Didier Galmiche
Dominique Larchey-Wendling (Eds.)

LNAI 8123

Automated Reasoning with Analytic Tableaux and Related Methods

22nd International Conference, TABLEAUX 2013
Nancy, France, September 2013
Proceedings

Springer

Didier Galmiche
Dominique Larchey-Wendling (Eds.)

Automated Reasoning with Analytic Tableaux and Related Methods

22nd International Conference, TABLEAUX 2013
Nancy, France, September 16-19, 2013
Proceedings

Springer

Volume Editors

Didier Galmiche
Université de Lorraine – LORIA UMR CNRS 7503
Campus Scientifique, BP 239
54 506 Vandœuvre-lès-Nancy, France
E-mail: didier.galmiche@loria.fr

Dominique Larchey-Wendling
LORIA – CNRS
Campus Scientifique, BP 239
54 506 Vandœuvre-lès-Nancy, France
E-mail: dominique.larchey-wendling@loria.fr

ISSN 0302-9743 e-ISSN 1611-3349
ISBN 978-3-642-40536-5 e-ISBN 978-3-642-40537-2
DOI 10.1007/978-3-642-40537-2
Springer Heidelberg New York Dordrecht London

Library of Congress Control Number: 2013946041

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

LNCS Sublibrary: SL 7 – Artificial Intelligence

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India

Printed on acid-free paper

Springer is part of Springer Science+Business Media (www.springer.com)

Preface

This volume contains the research papers presented at the International Conference on Automated Reasoning with Analytic Tableaux and Related Methods (TABLEAUX 2013) held September 16–19, 2013, in Nancy, France. This conference was the 22nd in a series of international meetings held since 1992 (listed on page VIII) and was co-located with FroCoS 2013, the International Symposium on Frontiers of Combining Systems.

The Program Committee of TABLEAUX 2013 received 38 submissions. Each paper was reviewed by at least three referees, and following a thorough and lively online discussion phase, 20 research papers including four system descriptions were accepted based on their originality, technical soundness, presentation, and relevance to the conference. We would like to sincerely thank both all the authors for their submissions and the members of the Program Committee and additional referees for their great effort and professional work in the review and selection process.

In addition to the contributed papers, the program included three keynote talks: Stéphane Demri (CIMS New York & LSV – ENS Cachan), common with FroCoS 2013, Sara Negri (University of Helsinki) and Tobias Nipkow (Technische Universität München). There were also two tutorials: "Noetherian Induction for First-Order Reasoning," by Sorin Stratulat (LITA, Université de Lorraine) and "Generating Tableau Provers Using MetTeL2," by Dmitry Tishkovsky and Renate A. Schmidt (School of Computer Science, University of Manchester). Two workshops were held in conjunction with TABLEAUX 2013: "Third Workshop of the Amadeus Project on Proof Compression," organized by Pascal Fontaine (INRIA Nancy – Grand-Est, LORIA, Université de Lorraine) and Bruno Woltzenlogel Paleo (Theory and Logic Group, Vienna University of Technology) and the Workshop "Logics for Resources, Processes and Programs," organized by David Pym (University of Aberdeen) and Didier Galmiche (LORIA – Université de Lorraine).

We would like to thank the members of the Organizing Committee for their much appreciated support and expertise, and the Conference Chair and Program Committee Chairs of FroCoS 2013 for our fruitful discussions and smooth coordination of the co-located events.

Finally, we would like to thank our sponsors for their generous and very welcome support: Région Lorraine, Communauté Urbaine du Grand Nancy, Université de Lorraine, CNRS, INRIA and LORIA. We would also like to acknowledge the `easychair.org` conference management system that greatly facilitated the smooth running of the review and selection process.

July 2013

Didier Galmiche
Dominique Larchey-Wendling

Organization

Program and Conference Chairs

Didier Galmiche	LORIA – Lorraine University, Nancy, France
Dominique Larchey-Wendling	LORIA – CNRS, Nancy, France

Program Committee

Carlos Areces	National University of Córdoba, Argentina
Arnon Avron	Tel Aviv University, Israel
Matthias Baaz	University of Technology, Vienna, Austria
Philippe Balbiani	IRIT – CNRS, Toulouse, France
Marta Cialdea Mayer	University of Rome Tre, Rome, Italy
Amy Felty	University of Ottawa, Canada
Ulrich Furbach	University of Koblenz-Landau, Germany
Valentin Goranko	Technical University of Denmark, Denmark
Rajeev Gore	Australian National University, Australia
Reiner Hähnle	Technical University of Darmstadt, Germany
George Metcalfe	University of Bern, Switzerland
Dale Miller	INRIA Saclay – LIX, Palaiseau, France
Neil Murray	State University of New York, USA
Nicola Olivetti	University of Marseille, France
Jens Otten	University of Potsdam, Germany
Lawrence C. Paulson	University of Cambridge, UK
Nicolas Peltier	LIG – CNRS, Grenoble, France
Renate Schmidt	University of Manchester, UK
Alex Simpson	University of Edinburgh, Scotland, UK
Viorica Sofronie-Stokkermans	University of Koblenz-Landau, Germany
Luca Vigano	University of Verona, Italy
Arild Waaler	University of Oslo, Norway

Additional Referees

James Brotherston	Martin Giese	Mohammad Khodadadi
Richard Bubel	Birte Glimm	Christoph Kreitz
Serenella Cerrito	Valentina Gliozzi	Roman Kuznets
Kaustuv Chaudhuri	Matthias Horbach	Ori Lahav
Ranald Clouston	Ullrich Hustadt	Espen H. Lian
Willem Conradie	Ran Ji	João Marcos
Stéphane Demri	Erisa Karafili	Andrew Matusiewicz

Daniel Méry	Ulrike Sattler	Alwen Tiu
Sara Negri	Claudia Schon	Marco Volpe
Fabio Papacchini	Lutz Strassburger	Anna Zamansky
Dirk Pattinson	Jimmy Thomson	
Gian Luca Pozzato	Dmitry Tishkovsky	

Steering Committee

Jens Otten (President)	University of Potsdam, Germany
Kai Brünnler	Innovation Process Tech. AG, Switzerland
Agata Ciabattoni	University of Technology, Vienna, Austria
Martin Giese	University of Oslo, Norway
Dominique Larchey-Wendling	LORIA – CNRS, Nancy, France
Angelo Montanari	University of Udine, Italy
Neil Murray	State University of New York, USA
Dale Miller	INRIA Saclay – LIX, Palaiseau, France

Sponsoring Institutions

INRIA, Institut National de Recherche en Informatique et en Automatique
LORIA, Laboratoire Lorrain de Recherche en Informatique et ses Applications
CNRS, Centre National de la Recherche Scientifique
UL, Université de Lorraine
CUGN, Communauté Urbaine du Grand Nancy
Région Lorraine

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)
2007 Aix-en-Provence, France
2008 Sydney, Australia (p.o. IJCAR)
2009 Oslo, Norway
2010 Edinburgh, UK (part of IJCAR)
2011 Bern, Switzerland
2012 Manchester, UK (part of IJCAR)

Table of Contents

Witness Runs for Counter Machines[⋆]
(Abstract)

Clark Barrett[1], Stéphane Demri[1,2], and Morgan Deters[1]

[1] New York University, USA
[2] LSV, CNRS, France

Abstract. We present recent results about the verification of counter machines by using decision procedures for Presburger arithmetic. We recall several known classes of counter machines for which the reachability sets are Presburger-definable as well as temporal logics with arithmetical constraints. We discuss issues related to flat counter machines, path schema enumeration and the use of SMT solvers.

Counter machines. Counter machines are well-known infinite-state systems that have many applications in formal verification. Their ubiquity stems from their use as operational models for several purposes, including for instance for broadcast protocols [FL02] and for logics for data words, see e.g. [BL10]. However, numerous model-checking problems for counter machines, such as reachability, are known to be undecidable [Min61, Min67]. Many subclasses of counter machines admit a decidable reachability problem, such as reversal-bounded counter automata [Iba78] and flat counter automata [FO97, CJ98, Boi99, FL02]. These two classes of systems admit reachability sets effectively definable in Presburger arithmetic (assuming some additional conditions, unspecified herein). Indeed, the set of reachable configurations can be effectively represented symbolically, typically by a formula in Presburger arithmetic for which satisfiability is known to be decidable, see e.g. [Pre29, BC96, Wol09, BBL09]. In general, computing the transitive closures of integer relations is a key step to solve verification problems on counter machines, see e.g. [BW94, CJ98, Fri00, BIK09].

Flatness. Flat counter machines are counter machines in which each control state belongs to at most one simple cycle (i.e. a cycle without any repetition of edges) [FO97, CJ98]. Several classes of such flat operational devices have been identified and reachability sets have been shown effectively Presburger-definable for many of them, see e.g. [CJ98, Boi99, Fri00, FL02, BIK09]. This provides a decision procedure for the reachability problem, given a prover for Presburger arithmetic validity. Effective semilinearity boils down to check that the effect of a loop can be characterized by a formula in Presburger arithmetic (or in any decidable fragment of first-order arithmetic). The results for flat counter machines can be then obtained by adequately composing formulae for loops and for finite paths. However, this approach, briefly described in this talk, suffers from at

[⋆] Work partially supported by the EU Seventh Framework Programme under grant agreement No. PIOF-GA-2011-301166 (DATAVERIF). An extended version can be found in [BDD13].

least two drastic limitations. First, flatness in counter machines remains a strong restriction on the control graph, though this has been relaxed by considering flattable counter machines, see e.g. [BFLS05, LS05, Ler13], where a machine may not itself be flat, but is known to have a flat unfolding with the same reachability set. The second limitation is due to the fact that reachability questions are not the only interesting ones and the verification of properties expressed in dedicated temporal logics is often desirable, see e.g. [DFGvD10, KF11].

Content of the talk. We present a selection of results about the verification of counter machines, at times assuming flatness, from reachability problems to model-checking problems with temporal logics (see e.g. [DDS12]). Systems and properties are a bit more general than those in [Fri00] but essentially we follow the same approach. We consider flattable counter machines and how to compute flat unfoldings by enumerating path schemas while invoking Satisfiability Modulo Theories (SMT) [BSST08] solvers to optimize this enumeration. This part of the talk presents preliminary results, and it will be the subject of a dedicated paper.

We provide several results from the literature and we emphasize that the generation of path schemas is a key problem for formal verification of Presburger counter systems (see also [Fri00, Ler03]). This is not really new (see e.g. [FO97, Boi99, Ler03]), but it is becoming an important issue, at least as important as the design of optimal decision procedures as far as worst-case complexity is concerned. The talk has been designed to shed some light on this problem. However, an efficient generation of path schemas means that redundant path schemas should be eliminated (via a subsumption test) as early as possible in the enumeration process. A comparison with the algorithm for the acceleration techniques in FAST [Ler03] or LASH [Boi99] will be in order.

Bounding the set of runs. Enumerating path schemas can also be viewed as a way to underapproximate the set of runs; this is similar to a standard approach to consider subclasses of runs by bounding some features and to search for 'bounded runs' that may satisfy a desirable or undesirable property. Examples include reversal-bounded counter machines (which have a bound on the number of reversals) [Iba78, GI81, BD11, HL11], context-bounded model-checking (where there is a bound on the number of context switches) [QR05], and of course bounded model-checking (BMC) (where there is a bound on the distance of the reached positions), see e.g. [BCC+03].

Subsumption & Presburger arithmetic. The notion of subsumption takes care of redundancy, and again subsumption can be checked by testing the satisfiability of a quantified Presburger formula. It is a challenge to deal with such quantified formulae in the framework of path schema enumeration since most SMT solvers do not behave so nicely with quantified formulae (see e.g. [dMB08, BCD+11]). Part of our future work is dedicated to design a path schema generation algorithm that invokes SMT solvers for quantified Presburger formulae. Deciding Presburger arithmetic fragments is essential to verify properties of programs; see e.g. [Sho79] and [SJ80] for an early use of Presburger arithmetic for formal verification. Most well-known SMT solvers can deal with quantifier-free linear integer formulae, also known as quantifier-free linear integer arithmetic (QF_LIA).

However, dealing with quantifiers is usually a difficult task for SMT solvers that are better tailored to theory reasoning. Many general-purpose SMT solvers do accept formulas with quantifiers and they handle them in roughly the same way, through heuristic instantiation. For instance, this includes Z3 [dMB08], CVC4 [BCD$^+$11] and Alt-Ergo [Con12], to cite a few of them, although Z3 additionally supports quantifier elimination for certain theories.

References

[BBL09] Boigelot, B., Brusten, J., Leroux, J.: A Generalization of Semenov's Theorem to Automata over Real Numbers. In: Schmidt, R.A. (ed.) CADE 2009. LNCS (LNAI), vol. 5663, pp. 469–484. Springer, Heidelberg (2009)

[BC96] Boudet, A., Comon, H.: Diophantine equations, Presburger arithmetic and finite automata. In: Kirchner, H. (ed.) CAAP 1996. LNCS, vol. 1059, pp. 30–43. Springer, Heidelberg (1996)

[BCC$^+$03] Biere, A., Cimatti, A., Clarke, E.M., Strichman, O., Zhu, Y.: Bounded model checking. Advances in Computers 58, 118–149 (2003)

[BCD$^+$11] Barrett, C., Conway, C.L., Deters, M., Hadarean, L., Jovanović, D., King, T., Reynolds, A., Tinelli, C.: CVC4. In: Gopalakrishnan, G., Qadeer, S. (eds.) CAV 2011. LNCS, vol. 6806, pp. 171–177. Springer, Heidelberg (2011)

[BD11] Bersani, M.M., Demri, S.: The complexity of reversal-bounded model-checking. In: Tinelli, C., Sofronie-Stokkermans, V. (eds.) FroCoS 2011. LNCS (LNAI), vol. 6989, pp. 71–86. Springer, Heidelberg (2011)

[BDD13] Barrett, C., Demri, S., Deters, M.: Witness runs for counter machines. In: FROCOS 2013. LNCS. Springer (to appear, 2013)

[BFLS05] Bardin, S., Finkel, A., Leroux, J., Schnoebelen, P.: Flat acceleration in symbolic model checking. In: Peled, D.A., Tsay, Y.-K. (eds.) ATVA 2005. LNCS, vol. 3707, pp. 474–488. Springer, Heidelberg (2005)

[BIK09] Bozga, M., Iosif, R., Konečný, F.: Fast acceleration of ultimately periodic relations. In: Touili, T., Cook, B., Jackson, P. (eds.) CAV 2010. LNCS, vol. 6174, pp. 227–242. Springer, Heidelberg (2010)

[BL10] Bojańczyk, M., Lasota, S.: An extension of data automata that captures XPath. In: LICS 2010, pp. 243–252. IEEE (2010)

[Boi99] Boigelot, B.: Symbolic methods for exploring infinite state spaces. PhD thesis, Université de Liège (1999)

[BSST08] Barrett, C., Sebastiani, R., Seshia, S., Tinelli, C.: Satisfiability Modulo Theories. Frontiers in Artificial Intelligence and Applications, vol. 185, ch. 26, pp. 825–885. IOS Press (2008)

[BW94] Boigelot, B., Wolper, P.: Verification with Periodic Sets. In: Dill, D.L. (ed.) CAV 1994. LNCS, vol. 818, pp. 55–67. Springer, Heidelberg (1994)

[CJ98] Comon, H., Jurski, Y.: Multiple counter automata, safety analysis and Presburger Arithmetic. In: Vardi, M.Y. (ed.) CAV 1998. LNCS, vol. 1427, pp. 268–279. Springer, Heidelberg (1998)

[Con12] Conchon, S.: SMT Techniques and their Applications: from Alt-Ergo to Cubicle. Habilitation à Diriger des Recherches, Université Paris-Sud (2012)

[DDS12] Demri, S., Dhar, A.K., Sangnier, A.: Taming Past LTL and Flat Counter Systems. In: Gramlich, B., Miller, D., Sattler, U. (eds.) IJCAR 2012. LNCS (LNAI), vol. 7364, pp. 179–193. Springer, Heidelberg (2012)

[DFGvD10] Demri, S., Finkel, A., Goranko, V., van Drimmelen, G.: Model-checking CTL^* over flat Presburger counter systems. JANCL 20(4), 313–344 (2010) [dMB08] de Moura, L., Bjørner, N.: Z3: An Efficient SMT Solver. In: Ramakrishnan, C.R., Rehof, J. (eds.) TACAS 2008. LNCS, vol. 4963, pp. 337–340. Springer, Heidelberg (2008)

[FL02] Finkel, A., Leroux, J.: How to compose Presburger accelerations: Applications to broadcast protocols. In: Agrawal, M., Seth, A.K. (eds.) FSTTCS 2002. LNCS, vol. 2556, pp. 145–156. Springer, Heidelberg (2002)

[FO97] Fribourg, L., Olsén, H.: Proving safety properties of infinite state systems by compilation into Presburger arithmetic. In: Mazurkiewicz, A., Winkowski, J. (eds.) CONCUR 1997. LNCS, vol. 1243, pp. 213–227. Springer, Heidelberg (1997)

[Fri00] Fribourg, L.: Petri nets, flat languages and linear arithmetic. In: 9th Workshop on Functional and Logic Programming (WFLP), pp. 344–365 (2000)

[GI81] Gurari, E., Ibarra, O.: The complexity of decision problems for finite-turn multicounter machines. In: ICALP 1981. LNCS, vol. 115, pp. 495–505. Springer (1981)

[HL11] Hague, M., Lin, A.W.: Model checking recursive programs numeric data types. In: Gopalakrishnan, G., Qadeer, S. (eds.) CAV 2011. LNCS, vol. 6806, pp. 743–759. Springer, Heidelberg (2011)

[Iba78] Ibarra, O.: Reversal-bounded multicounter machines and their decision problems. JACM 25(1), 116–133 (1978)

[KF11] Kuhtz, L., Finkbeiner, B.: Weak Kripke structures and LTL. In: Katoen, J.-P., König, B. (eds.) CONCUR 2011. LNCS, vol. 6901, pp. 419–433. Springer, Heidelberg (2011)

[Ler03] Leroux, J.: Algorithmique de la vérification des systèmes à compteurs. Approximation et accélération. Implémentation de l'outil FAST. PhD thesis, ENS de Cachan, France (2003)

[Ler13] Leroux, J.: Presburger Vector Addition Systems. In: LICS 2013, pp. 23–32. IEEE (2013)

[LS05] Leroux, J., Sutre, G.: Flat counter automata almost everywhere! In: Peled, D.A., Tsay, Y.-K. (eds.) ATVA 2005. LNCS, vol. 3707, pp. 489–503. Springer, Heidelberg (2005)

[Min61] Minsky, M.: Recursive unsolvability of Post's problems of 'tag' and other topics in theory of Turing machines. Annals of Mathematics 74(3), 437–455 (1961)

[Min67] Minsky, M.: Computation, Finite and Infinite Machines. Prentice Hall (1967)

[Pre29] Presburger, M.: Über die Vollständigkeit eines gewissen Systems der Arithmetik ganzer Zahlen, in welchem die Addition als einzige Operation hervortritt. In: Comptes Rendus du Premier Congrès de Mathématiciens des Pays Slaves, Warszawa, pp. 92–101 (1929)

[QR05] Qadeer, S., Rehof, J.: Context-bounded model checking of concurrent software. In: Halbwachs, N., Zuck, L.D. (eds.) TACAS 2005. LNCS, vol. 3440, pp. 93–107. Springer, Heidelberg (2005)

[Sho79] Shostak, R.: A practical decision procedure for arithmetic with function symbols. JACM 26(2), 351–360 (1979)

[SJ80] Suzuki, N., Jefferson, D.: Verification Decidability of Presburger Array Programs. JACM 27(1), 191–205 (1980)

[Tin07] Tinelli, C.: An Abstract Framework for Satisfiability Modulo Theories. In: Olivetti, N. (ed.) TABLEAUX 2007. LNCS (LNAI), vol. 4548, p. 10. Springer, Heidelberg (2007)

[Wol09] Wolper, P.: On the use of automata for deciding linear arithmetic. In: Giese, M., Waaler, A. (eds.) TABLEAUX 2009. LNCS, vol. 5607, p. 16. Springer, Heidelberg (2009)

On the Duality of Proofs and Countermodels in Labelled Sequent Calculi

Sara Negri

Department of Philosophy
PL 24, Unioninkatu 40 B
00014 University of Helsinki, Finland
sara.negri@helsinki.fi

The duality of proofs and counterexamples, or more generally, refutations, is ubiquitous in science, but involves distinctions often blurred by the rethoric of argumentation. More crisp distinctions between proofs and refutations are found in mathematics, especially in well defined formalized fragments.

Every working mathematician knows that finding a proof and looking for a counterexample are two very different activities that cannot be carried on simultaneusly. Usually the latter starts when the hope to find a proof is fading away, and the failed attempts will serve as an implicit guide to chart the territory in which to look for a counterexample. No general recipe is, however, gained from the failures, and a leap of creativity is required to find a counterxample, if such is at all obtained.

In logic, things are more regimented because of the possibility to reason within formal *analytic calculi* that reduce the proving of theorems to *automatic* tasks. Usually one can rest upon a completeness theorem that guarantees a perfect duality between proofs and countermodels. So in theory. In practice, we are encountered with obstacles: completeness proofs are often non-effective (non-constructive) and countermodels are artificially built from Henkin sets or Lindenbaum algebras, and thus far away from what we regard as counterexamples. Furthermore, the canonical countermodels provided by traditional completeness proofs may fall out of the intended classes and need a model-theoretic fine tuning with such procedures as unravelling and bulldozing.

The question naturally arises as to whether we can find in some sense "concrete" countermodels in the same automated way in which we find proofs. *Refutation calculi* (as those found in [5, 9, 20, 23]) produce refutations rather than proofs and can be used as a basis for building countermodels. These calculi are separate from the direct inferential systems, their rules are not invertible (root-first, the rules give only sufficient conditions of non-validity) and sometimes the decision method through countermodel constructions uses a pre-processing of formulas into a suitable normal form (as in [11]). As pointed out in [10] in the presentation of a combination of a derivation and a refutation calculus for bi-intuitionistic logic, these calculi often depart from Gentzen's original systems, because the sequent calculus LI or its contraction-free variant LJT [2] have rules that are not invertible; thus, while preserving validity, they do not preserve refutability. *Prefixed tableaux* in the style of Fitting, on the other hand, restrict the refutations to relational models, and countermodels can be read off from failed proof search. As remarked in [6], the tree structure inherent in these calculi makes them suitable to a relatively restricted family of logics and, furthermore, the non-locality of the rules makes the extraction of the countermodel not an immediate task.

We shall present a method for unifying proof search and countermodel construction that is a synthesis of a generation of calculi with internalized semantics (as presented in [14] and in chapter 11 of [19]), a Tait-Schütte-Takeuti style completeness proof [15] and, finally, a procedure to finitize the countermodel construction. This final part is obtained either through the search of a minimal, or irredundant, derivation (a procedure employed to establish decidability of basic modal logics in [14] and formalized in [7] for a labelled sequent system for intuitionistic logic), a pruning of infinite branches in search trees through a suitable syntactic counterpart of semantic filtration (a method employed in [1] for Priorean linear time logic and in [8] for multimodal logics) or through a proof-theoretic embedding into an appropriate provability logic that internalizes finiteness in its rules, as in [4].

The emphasis here is on the *methodology*, so we shall present the three stages in detail for the case of intuitionistic logic. Our starting point is **G3I**, a labelled contraction- and cut-free intuitionistic multi-succedent calculus in which *all rules are invertible.* The calculus is obtained through the internalization of Kripke semantics for intuitionistic logic: the rules for the logical constants are obtained by unfolding the inductive definition of truth at a world and the properties of the accessibility relation are added as rules, following the method of "axioms as rules" to encode axioms into a sequent calculus while preserving the structural properties of the basic logical calculus [18, 13]. The structural properties guarantee a root-first determinism, with the consequence that there is no need of backtracking in proof search. Notably for our purpose, all the rules of the calculus preserve countermodels because of invertibility, and thus any terminal node in a failed proof search gives a Kripke countermodel.

The methodology of generation of complete analytic countermodel-producing calculi covers in addition the following (families of) logics and extensions:

Intermediate Logics and Their Modal Companions: These are obtained as extensions of **G3I** and of the labelled calculus for basic modal logic **G3K** by the addition of geometric frame rules. Because of the uniformity of generation of these calculi, proofs of faithfulness of the modal translation between the respective logical systems are achieved in a modular and simple way [3].

Provability Logics: The condition of Noetherian frames, though not first order, is internalized through suitable formulations of the right rule for the modality. By choosing *harmonious* rules (as in [14]), a syntactic completeness proof for Gödel-Löb provability logic was obtained. Through a variant of the calculus obtained by giving up harmony, we achieve instead a semantic completeness proof which gives at the same time also decidability and the finite model property [17]. Completeness for Grzegorczyk provability logic **Grz** is obtained in a similar semantic way and prepares the ground for a syntactic embedding of **Int** into **Grz** and thus for an indirect decision procedure for intuitionistic logic [4].

Knowability Logic: This logic has been in the focus of recent literature on the investigation of paradoxes that arise from the principles of the verificationist theory of truth [21]. By the methods of proof analysis, it has been possible to pinpoint how the ground logic is responsible for the paradoxical consequences of these principles. A study focused on the well known Church-Fitch paradox brought forward a new challenge to the method of conversion of axioms into rules. The *knowability principle*, which states that whatever is true can be known, is rendered in a standard multimodal alethic/epistemic language by the axiom $A \supset \Diamond\mathcal{K}A$. This axiom corresponds, in turn, to the frame property

$$\forall x \exists y (xRy \,\&\, \forall z (yR_{\mathcal{K}}z \supset x \leqslant z))$$

Here R, $R_{\mathcal{K}}$, and $\leqslant$ are the alethic, epistemic, and intuitionistic accessibility relations, respectively. This frame property goes beyond the scheme of geometric implication and therefore the conversion into rules cannot be carried through with the usual rule scheme for geometric implications. In this specific case, we succeeded with a combination of two rules linked together by a side condition on the eigenvariable. The resulting calculus has all the structural properties of the ground logical system and leads to definite answers to the questions raised by the Church-Fitch paradox by means of a complete control over the structure of derivations for knowability logic [12].

Extensions Beyond Geometric Theories: The generalization and systematization of the method of system of rules allows the treatment of axiomatic theories and of logics characterized by frame properties expressible through *generalized geometric implications* that admit arbitrary quantifier alternations and a more complex propositional structure than that of geometric implications [16]. The class of generalized geometric implications is defined as follows: We start from a *geometric axiom* (i.e. a conjunct in the canonical form of a geometric implication [22], where the P_i range over a finite set of atomic formulas and all the M_j are conjunctions of atomic formulas and the variables y_j are not free in the P_i)

$$GA_0 \equiv \forall \overline{x}(\& P_i \supset \exists y_1 M_1 \vee \ldots \vee \exists y_n M_n)$$

We take GA_0 as the base case in the inductive definition of a *generalized geometric axiom.* We then define

$$GA_1 \equiv \forall \overline{x}(\,\& \, P_i \supset \exists y_1 \,\&\, GA_0 \vee \ldots \vee \exists y_m \,\&\, GA_0)$$

Next we define by induction

$$GA_{n+1} \equiv \forall \overline{x}(\,\& \, P_i \supset \exists y_1 \,\&\, GA_{k_1} \vee \ldots \vee \exists y_m \,\&\, GA_{k_m})$$

Here $\& \, GA_i$ denotes a conjunction of GA_i-axioms and $k_1, \ldots, k_m \leqslant n$.

Through an operative conversion to normal form, generalized geometric implications can also be characterized in terms of *Glivenko classes* as those first-order formulas that do not contain implications or universal quantifiers in their negative parts.

The equivalence, established in [13], between the axiomatic systems based on geometric axioms and contraction- and cut-free sequent systems with geometric rules, is extended by a suitable definition of *systems of rules* for generalized geometric axioms. Here the word “system” is used in the same sense as in linear algebra where there are systems of equations with variables in common, and each equation is meaningful and can be solved only if considered together with the other equations of the system. In the same way, the systems of rules considered in this context consist of rules connected to each other by some variables and subject in addition to the condition of appearing in a certain order in a derivation.

The precise form of system of rules, the structural properties for the resulting extensions of sequent calculus (admissibility of cut, weakening, and contraction), a generalization of Barr’s theorem, examples from axiomatic theories and applications to the proof theory of non-classical logics through a proof of completeness of the proof systems obtained, are all detailed in [16].

We shall conclude with some open problems and further directions.

References

1. Boretti, B., Negri, S.: Decidability for Priorean linear time using a fixed-point labelled calculus. In: Giese, M., Waaler, A. (eds.) TABLEAUX 2009. LNCS, vol. 5607, pp. 108–122. Springer, Heidelberg (2009)
2. Dyckhoff, R.: Contraction-free sequent calculi for intuitionistic logic. The Journal of Symbolic Logic 57, 795–807 (1992)
3. Dyckhoff, R., Negri, S.: Proof analysis in intermediate logics. Archive for Mathematical Logic 51, 71–92 (2012)
4. Dyckhoff, R., Negri, S.: A cut-free sequent system for Grzegorczyk logic, with an application to the Gödel-McKinsey-Tarski embedding. Journal of Logic and Computation (2013), doi:10.1093/logcom/ext036
5. Ferrari, M., Fiorentini, C., Fiorino, G.: Contraction-free linear depth sequent calculi for intuitionistic propositional logic with the subformula property and minimal depth counter-models. Journal of Automated Reasoning (2012), doi:10.1007/s10817-012-9252-7
6. Fitting, M.: Prefixed tableaus and nested sequents. Annals of Pure and Applied Logic 163, 291–313 (2012)
7. Galmiche, D., Salhi, Y.: Sequent calculi and decidability for intuitionistic hybrid logic. Information and Computation 209, 1447–1463 (2011)
8. Garg, G., Genovese, V., Negri, S.: Counter-models from sequent calculi in multi-modal logics. In: LICS 2012, pp. 315–324. IEEE Computer Society (2012)
9. Goranko, V.: Refutation systems in modal logic. Studia Logica 53, 299–324 (1994)
10. Goré, R., Postniece, L.: Combining derivations and refutations for cut-free completeness in bi-intuitionistic logic. Journal of Logic and Computation 20, 233–260 (2010)
11. Larchey-Wendling, D.: Combining proof-search and counter-model construction for deciding Gödel-Dummett logic. In: Voronkov, A. (ed.) CADE-18. LNCS (LNAI), vol. 2392, pp. 94–110. Springer, Heidelberg (2002)

12. Maffezioli, P., Naibo, A., Negri, S.: The Church-Fitch knowability paradox in the light of structural proof theory. Synthese (2012) (Online first), doi:10.1007/s11229-012-0061-7
13. Negri, S.: Contraction-free sequent calculi for geometric theories, with an application to Barr's theorem. Archive for Mathematical Logic 42, 389–401 (2003)
14. Negri, S.: Proof analysis in modal logic. Journal of Philosophical Logic 34, 507–544 (2005)
15. Negri, S.: Kripke completeness revisited. In: Primiero, G., Rahman, S. (eds.) Acts of Knowledge - History, Philosophy and Logic, pp. 247–282. College Publications (2009)
16. Negri, S.: Proof analysis beyond geometric theories: from rule systems to systems of rules (2012) (submitted)
17. Negri, S.: A terminating cut-free sequent system for Gödel-Löb provability logic. Manuscript (2012)
18. Negri, S., von Plato, J.: Cut elimination in the presence of axioms. The Bulletin of Symbolic Logic 4, 418–435 (1998)
19. Negri, S., von Plato, J.: Proof Analysis. Cambridge University Press (2011)
20. Pinto, L., Dyckhoff, R.: Loop-free construction of counter-models for intuitionistic propositional logic. In: Behara, et al. (eds.) Symposia Gaussiana, Conf. A, pp. 225–232. de Gruyter, Berlin (1995)
21. Salerno, J. (ed.): New Essays on the Knowability Paradox. Oxford University Press (2009)
22. Simpson, A.: Proof Theory and Semantics of Intuitionistic Modal Logic. Ph.D. thesis, School of Informatics, University of Edinburgh (1994)
23. Skura, T.: Refutation systems in propositional logic. In: Gabbay, D., Guenthner, F. (eds.) Handbook of Philosophical Logic, vol. 16, pp. 115–157. Springer (2011)

A Brief Survey of Verified Decision Procedures for Equivalence of Regular Expressions

Tobias Nipkow and Maximilian Haslbeck

Institut für Informatik, Technische Universität München

Abstract. We survey a number of decision procedures for the equivalence of regular expressions that have been formalised with the help of interactive proof assistant over the past few years.

1 Introduction

Equivalence of regular expressions is a perennial topic in computer science. Recently it has spawned a number of papers that have formalised and verified various different algorithm for this task in interactive theorem provers. One of the motivations is that such verified decision procedures can help to automate reasoning about binary relations: relation composition corresponds to concatenation, reflexive transitive closure to Kleene star, and $\cup$ to $+$. It can be shown [9] that an equivalence between two relation algebraic expressions holds if the corresponding two regular expressions are equivalent—the other direction holds too, provided the base type of the relations is infinite.

In this brief note we survey the different formalisations that have appeared over the last few years. We have reproduced most of them in the Isabelle proof assistant and compare some of them on this basis.

Braibant and Pous [4] where the first to verify an equivalence checker for regular expressions. The work was carried out in Coq. They followed the classical approach of translating the regular expressions into automata. The resulting theory was quite large and their algorithm efficient. Although they set the trend, the next four verified decision procedures all worked directly on regular expressions. The motivation is simplicity: regular expressions are a free data type which proof assistants and their users love because it means induction, recursion and equational reasoning, the core competence of proof assistants and functional languages.

The outer shell of all the decision procedures that operate directly on regular expressions is the same. Roughly speaking, there is always a function $\delta : regexp \times \Sigma \to regexp$ that extends canonically to words. Starting from some pair (r, s), all the pairs $(\delta(r, w), \delta(s, w))$ are enumerated; the setup guarantees there are only finitely many $\delta(r, w)$ for each r. If for all such pairs (r', s'), r' is final iff s' is final (for a suitable notion of finality), then $r \equiv s$ (and conversely). This is just an incremental computation of the product automaton or a bisimulation.

2 Derivatives of Regular Expressions

Brzozowski [5] had introduced derivatives of regular expressions in 1964 and had shown that modulo ACI of +, there are only finitely many derivatives of a regular expression, which correspond to the states of a DFA for that regular expression. In 2009, Owens *et al.* [13] used derivatives for scanner generators in ML. They wrote

> derivatives have been lost in the sands of time, and few computer scientists are aware of them.

As we shall see, they have certainly become better known by now.

In response to Braibant and Pous, Krauss and Nipkow [9] verified partial correctness an equivalence checker for regular expressions based on derivatives in Isabelle. The formalization is very small and elegant, although not very efficient for larger problems. Coquand and Siles [6] extended this work in Coq. The emphasis of their work is on the finiteness/termination proof in type theory.

Antimirov [1] introduced partial derivatives of regular expressions. They can be viewed as sets of derivatives, thus building in ACI of +. Moreira *et al.* [11] present an equivalence checker for regular expressions based on partial derivatives and show its total correctness in Coq—termination is proved by showing finiteness of the set of partial derivatives of an expression. We have formalized the same proofs in Isabelle. Moreover we have shown termination of a modified version of the equivalence checker by Krauss and Nipkow as follows. Instead of comparing derivatives normalised w.r.t. ACI of +, we convert them into partial derivatives before comparing them. Thus we can reuse finiteness of the set of partial derivatives to prove termination of the algorithm based on derivatives.

3 Marked Regular Expressions

Both McNaughton and Yamada [10] and Glushkov [8] marked the atoms in a regular expression with numbers in order to turn it into an automaton. Fischer *et al.* [7] realize the convenience of working directly with regular expressions in a functional programming setting. They present matching algorithms on regular expression with boolean marks indicating where in the regular expression the matching process has arrived. Independently, Asperti [2] verifies an equivalence checker for regular expressions via marked regular expressions in the Matita proof assistant. We have verified the basic algorithm by Fischer *et al.* and the one by Asperti in Isabelle and shown that they are closely related: the one by Fischer *et al.* marks the atoms that have just occurred, Asperti marks the atoms that can occur next.

4 Related Formalisations

Moreira *et al.* [12] formalise a decision procedure for Kleene Algebra with Tests as an extension of their earlier work [11] and show its application to program verification by encoding Hoare triples algebraically. Traytel and Nipkow [14] present verified decision procedures for monadic second order logics (MSO) on finite words based on derivatives of regular expressions extended with complementation and projection. Outside of the application area of equivalence checking, Wu *et al.* [15] verify thy Myhill-Nerode theorem in Isabelle using regular expressions. Berghofer and Reiter [3] verify a decision procedure for Presburger arithmetic via automata in Isabelle.

References

1. Antimirov, V.: Partial derivatives of regular expressions and finite automaton constructions. Theor. Comput. Sci. 155, 291–319 (1996)
2. Asperti, A.: A compact proof of decidability for regular expression equivalence. In: Beringer, L., Felty, A. (eds.) ITP 2012. LNCS, vol. 7406, pp. 283–298. Springer, Heidelberg (2012)
3. Berghofer, S., Reiter, M.: Formalizing the logic-automaton connection. In: Berghofer, S., Nipkow, T., Urban, C., Wenzel, M. (eds.) TPHOLs 2009. LNCS, vol. 5674, pp. 147–163. Springer, Heidelberg (2009)
4. Braibant, T., Pous, D.: An efficient Coq tactic for deciding Kleene algebras. In: Kaufmann, M., Paulson, L. (eds.) ITP 2010. LNCS, vol. 6172, pp. 163–178. Springer, Heidelberg (2010)
5. Brzozowski, J.: Derivatives of regular expressions. J. ACM 11, 481–494 (1964)
6. Coquand, T., Siles, V.: A decision procedure for regular expression equivalence in type theory. In: Jouannaud, J.-P., Shao, Z. (eds.) CPP 2011. LNCS, vol. 7086, pp. 119–134. Springer, Heidelberg (2011)
7. Fischer, S., Huch, F., Wilke, T.: A play on regular expressions. Functional pearl. In: Hudak, P., Weirich, S. (eds.) Proc. Int. Conf. Functional Programming, ICFP 2010, pp. 357–368 (2010)
8. Glushkov, V.M.: The abstract theory of automata. Russian Mathematical Surveys 16, 1–53 (1961)
9. Krauss, A., Nipkow, T.: Proof pearl: Regular expression equivalence and relation algebra. J. Automated Reasoning 49, 95–106 (2012) (published online March 2011)
10. McNaughton, R., Yamada, H.: Regular expressions and finite state graphs for automata. IRE Trans. on Electronic Comput. EC-9, 38–47 (1960)
11. Moreira, N., Pereira, D., de Sousa, S.M.: Deciding regular expressions (in-)equivalence in Coq. In: Kahl, W., Griffin, T.G. (eds.) RAMiCS 2012. LNCS, vol. 7560, pp. 98–113. Springer, Heidelberg (2012)
12. Moreira, N., Pereira, D., de Sousa, S.M.: Mechanization of an algorithm for deciding KAT terms equivalence. Tech. Rep. DCC-2012-04, Universidade do Porto (2012)
13. Owens, S., Reppy, J.H., Turon, A.: Regular-expression derivatives re-examined. J. Functional Programming 19, 173–190 (2009)
14. Traytel, D., Nipkow, T.: A verified decision procedure for MSO on words based on derivatives of regular expressions. In: Proc. Int. Conf. Functional Programming, ICFP 2013 (2013)
15. Wu, C., Zhang, X., Urban, C.: A formalisation of the Myhill-Nerode theorem based on regular expressions (Proof pearl). In: van Eekelen, M., Geuvers, H., Schmaltz, J., Wiedijk, F. (eds.) ITP 2011. LNCS, vol. 6898, pp. 341–356. Springer, Heidelberg (2011)

Dealing with Symmetries in Modal Tableaux

Carlos Areces and Ezequiel Orbe[⋆]

FaMAF, U. Nacional de Córdoba (Argentina) and CONICET

Abstract. We present a technique that is able to detect symmetries in formulas of the basic modal logic ($\mathcal{BML}$). Then we introduce a modal tableaux calculus for $\mathcal{BML}$ with a blocking mechanism that takes advantage of symmetry information about the input formula to restrict the application of the ($\Diamond$) rule. We prove completeness of the calculus. Finally, we empirically evaluate both, the detection technique and the blocking mechanism in different modal benchmarks.

1 Introduction

In this work we investigate symmetries in modal logics. We discuss how to detect them in modal formulas and how to exploit them in a modal tableaux prover.

In the context of automated reasoning a symmetry can be defined as a permutation of the variables (or literals) of a problem that preserves its structure and its set of solutions. Symmetries have been extensively investigated and successfully exploited for propositional satisfiability (SAT) since the introduction in [21] of symmetry inference rules to strengthen resolution-based proof systems for propositional logic which lead to much shorter proofs of certain difficult problems (e.g., the pigeonhole problem). Since then many articles discuss how to detect and exploit symmetries. Most of them can be grouped into two different approaches: static symmetry breaking [11,12,1] and dynamic symmetry breaking [7,8]. In the former, symmetries are detected and eliminated from the problem statement before the SAT solver is used, i.e., they work as a preprocessing step, whilst in the latter symmetries are detected and broken during the search space exploration. Despite their differences they share the same goal: to identify symmetric branches of the search space and guide the SAT solver away from symmetric branches already explored.

Research involving other logics has been done in the last years, e.g. [5,14]. To the best of our knowledge, symmetries remain largely unexplored in automated theorem proving for modal logics. We investigated the theoretical foundations to exploit symmetries in a number of modal logics in [4]. In this paper we present a graph-based method to detect *layered symmetries* for the basic modal logic

⋆ This work was partially supported by grants ANPCyT-PICT-2008-306, ANPCyT-PICT-2010-688, the FP7-PEOPLE-2011-IRSES Project "Mobility between Europe and Argentina applying Logics to Systems" (MEALS) and the Laboratoire International Associé "INFINIS". We thank Guillaume Hoffmann for useful comments and help with `HTab`.

$\mathcal{BML}$ that extend the method presented in [4]. A layered symmetry is a more flexible notion of symmetry that can be defined in modal logics that enjoy the tree model property. Then we show how to use symmetry information in a modal tableaux calculus by presenting a novel blocking mechanism for a modal tableaux called *symmetry blocking*, that blocks the application of the $(\Diamond)$ rule if it has been already applied to a symmetric $\neg\Box$-formula. Finally we evaluate empirically both the detection technique and the blocking mechanism.

Outline. Section 2 introduces the required definitions on modal language and symmetries. Section 3 presents a graph construction algorithm to detect symmetries in modal formulas. Section 4 presents a $\mathcal{BML}$ tableaux calculus with symmetry blocking and proves it correctness. Finally, Section 5 presents experimental results about the detection construction and the effects of symmetry blocking in modal benchmarks. Section 6 concludes with some final remarks.

2 Definitions

In this section we introduce the basic notions concerning modal languages and permutations. In what follows, we will assume basic knowledge of classical modal logics and refer the reader to [9,10] for technical details. Let $\mathsf{PROP} = \{p_1, p_2, \ldots\}$ be a countably infinite set of *propositional variables*. The set of (basic) modal formulas FORM is defined as

$$\mathsf{FORM} := p \mid \neg p \mid \varphi \vee \psi \mid \varphi \wedge \psi \mid \neg\Box\varphi \mid \Box\varphi,$$

where $p \in \mathsf{PROP}$, $\varphi, \psi \in \mathsf{FORM}$. We will discuss only the mono-modal case, but the results we present extend naturally to the multi-modal case.

A *propositional literal* l is either a propositional variable p or its negation $\neg p$. The set of literals over PROP is $\mathsf{PLIT} = \mathsf{PROP} \cup \{\neg p_i \mid p_i \in \mathsf{PROP}\}$. A set of literals L is *complete* if for each $p \in \mathsf{PROP}$ either $p \in L$ or $\neg p \in L$. It is *consistent* if for each $p \in \mathsf{PROP}$ either $p \notin L$ or $\neg p \notin L$. Any complete and consistent set of literals L defines a unique valuation $v \subseteq \mathsf{PROP}$ as $p \in v$ if $p \in L$ and $p \notin v$ if $\neg p \in L$. For $S \subseteq \mathsf{PROP}$, the *consistent and complete set of literals generated by* S (notation L_S) is $S \cup \{\neg p \mid p \in \mathsf{PROP} \backslash S\}$.

A modal formula is in *modal conjunctive normal form (modal CNF)* if it is a conjunction of modal CNF clauses. A *modal CNF clause* is a disjunction of propositional and *modal literals*. A *modal literal* is a formula of the form $\Box C$ or $\neg\Box C$ where C is a modal CNF clause. Every modal formula can be transformed into an equisatisfiable formula in modal CNF in polynomial time [24]. In what follows assume that modal formulas are in modal CNF. Also, we often represent a modal CNF formula as a set of modal CNF clauses (interpreted conjunctively), and each modal CNF clause as a set of propositional and modal literals (interpreted disjunctively). The set representation disregards order and multiplicity of clauses and literals in a formula.

Models and semantics for modal CNF formulas are defined in the usual way. A (pointed) model is a tuple $\langle w, W, R, V\rangle$ such that W is a non-empty set, $w \in W$,

$R \subseteq W \times W$ and $V(v) \subseteq \mathsf{PROP}$ for all $v \in W$. Let $\mathcal{M} = \langle w, W, R, V\rangle$ be a model. We define $\models$ for modal CNF formulas, clauses and literals as

$$\begin{array}{lll}
\mathcal{M} \models \varphi & \text{iff} & \text{for all clauses } C \in \varphi \text{ we have } \mathcal{M} \models C\\
\mathcal{M} \models C & \text{iff} & \text{there is some literal } l \in C \text{ such that } \mathcal{M} \models l\\
\mathcal{M} \models p & \text{iff} & p \in V(w) \ \text{ for } \ p \in \mathsf{PROP}\\
\mathcal{M} \models \neg p & \text{iff} & p \notin V(w) \ \text{ for } \ p \in \mathsf{PROP}\\
\mathcal{M} \models \Box C & \text{iff} & \langle w', W, R, V\rangle \models C, \text{ for all } w' \text{ s.t. } wRw'\\
\mathcal{M} \models \neg\Box C & \text{iff} & \mathcal{M} \not\models \Box C.
\end{array}$$

The modal depth of a formula φ in CNF (notation $\mathsf{md}(\varphi)$) is the maximum nesting of $\Box$ operators that occurs in φ.

A *permutation* is a bijective function $\rho : \mathsf{PLIT} \mapsto \mathsf{PLIT}$. We will define permutations using cyclic notation [16], e.g., $\rho = (p\ \neg q)(\neg p\ q)$ is the permutation that makes $\rho(p) = \neg q$, $\rho(\neg q) = p$, $\rho(\neg p) = q$ and $\rho(q) = \neg p$ and leaves unchanged all other literals; $\rho = (p\ q\ r)(\neg p\ \neg q\ \neg r)$ is the permutation $\rho(p) = q$, $\rho(q) = r$ and $\rho(r) = p$ and similarly for the negations. If L is a set of literals then $\rho(L) = \{\rho(l) \mid l \in L\}$. Given a modal CNF formula φ and a permutation ρ we define $\rho(\varphi)$ as the formula obtained by simultaneously replacing all literals l in φ by $\rho(l)$. We say that a permutation ρ is *consistent* if for every literal l, $\rho(\neg l) = \neg\rho(l)$. We say that a permutation ρ is a *symmetry* for φ if $\varphi = \rho(\varphi)$ when φ is represented using sets. For example, consider the formula $\varphi = (\neg p \vee r) \wedge (q \vee r) \wedge \Box(\neg p \vee q)$, then the permutation $\rho = (p\ \neg q)(\neg p\ q)$ is a consistent symmetry of φ.

In modal logics that enjoy the tree model property there is a direct correlation between the modal depth of the formula and the depth in a tree model satisfying it: in models, a notion of layer is induced by the depth (distance from the root) of the nodes, whereas in formulas, a notion of layer is induced by the nesting of the modal operators. A consequence of this correspondence is that propositional literals at different formula layers are semantically independent of each other (see [2] for details), i.e., at different layers the same propositional literal can be assigned a different value. This enables the definition of *layered permutations* [4], that let us define a different permutation at each modal depth.

A *finite permutation sequence* $\bar\rho$ is either the empty sequence $\bar\rho = \langle\rangle$ or $\bar\rho = \rho : \bar\rho_2$ with ρ a permutation and $\bar\rho_2$ a permutation sequence. We sometimes use the notation $\bar\rho = \langle\rho_1, \ldots, \rho_n\rangle$ instead of $\bar\rho = \rho_1{:}\ldots{:}\rho_n{:}\langle\rangle$. Let $|\bar\rho| = n$ be the length of $\bar\rho$ ($\langle\rangle$ has length 0). For $1 \leq i \leq n$, $\bar\rho_i$ is the sub-sequence that starts from the i^{th} element of $\bar\rho$ (in particular, $\bar\rho_i = \langle\rangle$ for $i \geq n$ and $\bar\rho_1 = \bar\rho$). $\bar\rho(i)$ is ρ_i if $1 \leq i \leq |\bar\rho|$ and $\bar\rho(i) = \rho_{Id}$ otherwise, for ρ_{Id} the identity permutation. A permutation sequence is *consistent* if all its permutations are consistent.

Let φ be a modal CNF formula and $\bar\rho$ a permutation sequence. Then $\bar\rho(\varphi)$ is defined recursively as

$$\begin{array}{rll}
\langle\rangle(\varphi) = \varphi & & \\
(\rho_1{:}\bar\rho_2)(l) = \rho_1(l) & & \text{for } l \in \mathsf{PLIT}\\
(\rho_1{:}\bar\rho_2)(\Box C) = \Box\bar\rho_2(C) & & \\
\bar\rho(C) = \{\bar\rho(A) \mid A \in C\} & & \text{for } C \text{ a clause or a formula.}
\end{array}$$

One benefit of using a different permutation at each modal depth is that this enables symmetries (layered symmetries) to be found, that would not be found otherwise. Consider the formula $\varphi = (p \vee \Box(p \vee \neg r)) \wedge (\neg q \vee \Box(\neg p \vee r))$. If we only consider non-layered symmetries then φ has none. However, the permutation sequence $\langle \rho_1, \rho_2 \rangle$ generated by $\rho_1 = (p\ \neg q)(\neg p\ q)$ and $\rho_2 = (p\ \neg r)(\neg p\ r)$ is a layered symmetry of φ.

Definition 1 ($\bar{\rho}$-simulation). *Let $\bar{\rho}$ be a permutation sequence. A $\bar{\rho}$-*simulation *between two pointed models $\mathcal{M} = \langle w, W, R, V \rangle$ and $\mathcal{M}' = \langle w', W', R', V' \rangle$ is a family of relations $Z_{\bar{\rho}_i} \subseteq W \times W'$, $1 \leq i$, that satisfies the following conditions:*

- ***Root:*** $wZ_{\bar{\rho}_1}w'$.
- ***Harmony:*** *If $wZ_{\bar{\rho}_i}w'$ then $l \in L_{V(w)}$ iff $\bar{\rho}_i(1)(l) \in L_{V'(w')}$.*
- ***Zig:*** *If $wZ_{\bar{\rho}_i}w'$ and wRv then $vZ_{\bar{\rho}_{i+1}}v'$ for some v' such that $w'R'v'$.*
- ***Zag:*** *If $wZ_{\bar{\rho}_i}w'$ and $w'R'v'$ then $vZ_{\bar{\rho}_{i+1}}v'$ for some v such that wRv.*

We say that two models $\mathcal{M}$ and $\mathcal{M}'$ are $\bar{\rho}$-similar (notation $\mathcal{M} \underline{\rightarrow}_{\bar{\rho}} \mathcal{M}'$), if there is a $\bar{\rho}$-simulation between them.

Notice that while $\mathcal{M} \underline{\rightarrow}_{\bar{\rho}} \mathcal{M}'$ implies $\mathcal{M}' \underline{\rightarrow}_{\bar{\rho}^{-1}} \mathcal{M}$, the relation $\underline{\rightarrow}_{\bar{\rho}}$ is not symmetric. From the definition of $\bar{\rho}$-simulations it intuitively follows that while they do not preserve validity of modal formulas (as is the case with bisimulations) they do preserve validity of *permutations* of formulas (see [4] for details).

Proposition 1. *Let $\bar{\rho}$ be a consistent layered permutation, φ a modal CNF formula and $\mathcal{M} = \langle w, W, R, V \rangle$, $\mathcal{M}' = \langle w', W', R', V' \rangle$ models such that $\mathcal{M} \underline{\rightarrow}_{\bar{\rho}} \mathcal{M}'$. Then $\mathcal{M} \models \varphi$ iff $\mathcal{M}' \models \bar{\rho}(\varphi)$.*

3 Detecting Modal Symmetries

We present a graph-based technique for the detection of symmetries in modal CNF formulas. In propositional logic, it is standard to reduce the problem of finding symmetries in formulas to the problem of finding automorphisms in colored graphs where graphs are constructed from formulas in such a way that its automorphism group is isomorphic to the symmetry group of the formula [1,11]

Although known to be an NP problem, it is not known if the graph automorphism problem is P or NP-complete [15]. Nevertheless there exist polynomial time algorithms for many special cases [23], and the are many efficient tools [13,20] capable of compute a set of generators [16] for the automorphism group of very large graphs efficiently.

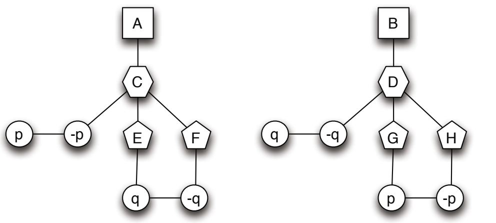
In [4] we extended the graph construction for propositional formulas presented in [1] to a wide range of modal logics. In this article we extend this construction to detect layered symmetries for $\mathcal{BML}$. In what follows, let $\mathit{Var}(\varphi, n)$ denote the set of variables occurring in φ at modal depth n. Clauses occurring at modal depth 0 are called *top clauses*, and clauses occurring in modal literals are called *modal clauses*.

Definition 2. *Let φ be a modal CNF formula of modal depth n. The colored graph $G(\varphi) = (V, E)$ corresponding to φ is constructed as follows:*

1. *For each propositional variable $p \in Var(\varphi, i)$ with $0 \leq i \leq n$:*
   (a) *Add two* literal *nodes of color 0: one labeled p_i and one labeled $\neg p_i$.*
   (b) *Add an edge between these two nodes to ensure Boolean consistency.*
2. *For each top clause C of φ add a* clause *node of color 1.*
3. *For each propositional literal l occurring in C, add an edge from C to the corresponding literal node $l_{\mathsf{md}(C)}$.*
4. *For each modal literal $\Box C'$ ($\neg\Box C'$) occurring in C:*
   (a) *Add a* clause *node of color 2 (color 3) to represent C'.*
   (b) *Add an edge from C to this node.*
   (c) *Repeat the process from point 3 for each literal (propositional or modal) occurring in C'.*

This construction creates a graph with 4 colors and at most $2|V| \times (\mathsf{md}(\varphi) + 1) + \#TopClauses + \#ModalClauses$ nodes.

Example 1. Let us consider the following modal CNF formula $\varphi = \neg\Box(\neg p \vee \Box q \vee \Box\neg q) \wedge \neg\Box(\neg q \vee \Box p \vee \Box\neg p)$. The associated 4-colored graph, $G(\varphi)$, is shown below (colors are represented by shapes in the figure).



$A = \neg\Box(\neg p \vee \Box q \vee \Box\neg q)$
$B = \neg\Box(\neg q \vee \Box p \vee \Box\neg p)$
$C = \neg p \vee \Box q \vee \Box\neg q$
$D = \neg q \vee \Box p \vee \Box\neg p$
$E = \Box q$
$F = \Box\neg q$
$G = \Box p$
$H = \Box\neg p$

Group Generators:
$$\bar{\rho}_1 = \langle \rho_{Id}, \rho_{Id}, (p\ \neg p)\rangle$$
$$\bar{\rho}_2 = \langle \rho_{Id}, \rho_{Id}, (q\ \neg q)\rangle$$
$$\bar{\rho}_3 = \langle \rho_{Id}, (p\ q)(\neg p\ \neg q), (p\ q)(\neg p\ \neg q)\rangle$$

Notice the way literals are handled during the construction of $G(\varphi)$: the construction duplicates literal nodes occurring at different modal depth. By doing this we incorporate the notion of layering introduced in Section 2. Also, modal literals $\Box C$ and $\neg\Box C$ are colored differently. This avoids spurious permutations mapping $\Box C$ literals to $\neg\Box C$ literals and the other way around. The following Proposition is proved in [4].

Proposition 2. *Let φ be a modal CNF formula and $G(\varphi) = (V, E)$ the colored graph obtained following the construction of Definition 2. Then every symmetry $\bar{\rho}$ of φ corresponds one-to-one to an automorphism π of $G(\varphi)$.*

4 Symmetry Blocking

We present a labeled tableaux calculus [18,10] for the basic modal logic $\mathcal{BML}$ and a dynamic blocking mechanism that uses symmetry information from the input formula. We will use standard tableaux prefixed rules and notation which we briefly review for completeness. Let PREF be an infinite, non-empty set of *prefixes*. Here we will set $\mathsf{PREF} = \mathbb{N}$. Given φ a modal CNF formula, C a modal clause and $\sigma \in \mathsf{PREF}$ we call $\sigma{:}\varphi$ and $\sigma{:}C$ *prefixed formulas*. The intended interpretation of a prefixed formula $\sigma{:}F$ is that F holds at the state denoted by σ. Given $\sigma, \sigma' \in \mathsf{PREF}$ we call $\sigma R \sigma'$ an *accessibility statement*. The intended interpretation of $\sigma R \sigma'$ is that the state denoted by σ' is accessible via R from σ.

A tableaux for $\varphi \in \mathcal{BML}$ is a tree whose nodes are decorated with prefixed formulas and accessibility statements, such that the root node is $0{:}\varphi$. Moreover, we require that additional nodes in the tree are created according to the following rules, where φ is a modal CNF formula and C a modal clause.

$$\frac{\sigma{:}\varphi}{\sigma{:}C_i}\,(\wedge) \quad \text{for all } C_i \in \varphi \qquad \frac{\sigma{:}C}{\sigma{:}l_1 \mid \ldots \mid \sigma{:}l_n}\,(\vee) \quad \text{for all } l_i \in C$$

$$\frac{\sigma{:}\neg\Box C}{\sigma R \sigma',\ \sigma'{:}{\sim}C}\,(\Diamond)^1 \qquad \frac{\sigma{:}\Box C, \sigma R \sigma'}{\sigma' : C}\,(\Box)$$

1 ${\sim}C$ is the CNF of the negation of C. The prefix σ' is new in the tableau.

The rules are interpreted as follows: if the antecedents of a rule appear in nodes in a branch of the tableaux, the branch is extended according to the formulas in the consequent. For the case of the $(\vee)$ rule, n immediate successors of the last node of the branch should be created. In all other cases only one successor is created, which is decorated with the indicated formulas. To ensure termination, we require that nodes are created only if they add at least one prefixed formula or accessibility statement that was not already in the branch. Moreover, the $(\Diamond)$ rule can only be applied once to each formula of the form $\sigma{:}\neg\Box C$ in a branch. A branch is *closed* if it contains both $\sigma{:}p$ and $\sigma{:}\neg p$, and it is *open* otherwise. A branch is *saturated* if no rule can be further applied in the branch (notice that the conditions we imposed on rule application lead to saturation after a finite number of steps).

Let $\mathsf{Tab}(\varphi)$ be the set of tableaux for φ whose branches are all saturated. The following classical result establishes that the tableaux calculus we just defined is a decision method for satisfiability of formulas in $\mathcal{BML}$.

Theorem 1. *For any $\varphi \in \mathcal{BML}$, any $\mathsf{T} \in \mathsf{Tab}(\varphi)$ is finite. Moreover T has a saturated open branch if and only if φ is satisfiable.*

This completes our brief introduction of the standard tableaux calculus for $\mathcal{BML}$. We are interested in further restricting the application of the $(\Diamond)$ rule so that it can be applied to a $\sigma{:}\neg\Box C$ formula only if it has not been applied to a symmetric formula before. Notice that this restriction cannot affect termination or soundness of the calculus.

Let $\mathsf{T} \in \mathsf{Tab}(\varphi)$, and let Θ be a branch of T. For $\sigma \in \mathsf{PREF}$ we will use $\mathsf{depth}(\sigma)$ for the distance from the root prefix (in particular, if σ is the prefix of φ then $\mathsf{depth}(\sigma) = 0$). Given a prefixed formula $\sigma{:}\varphi$ and a permutation sequence $\bar{\rho}$, we define $\bar{\rho}(\sigma{:}\varphi) = \sigma{:}\bar{\rho}_{\mathsf{depth}(\sigma)+1}(\varphi)$. Finally, given a modal formula φ, we define $\mathit{Var}(\varphi)$ as the set of propositional variables occurring in φ; for S a set of modal formulas, let $\mathit{Var}(S) = \bigcup_{\varphi\in S} \mathit{Var}(\varphi)$.

Symmetry Blocking: Let $\bar{\rho}$ be a layered symmetry of φ, and let Θ be a branch in a tableau of φ. The rule $(\Diamond)$ cannot be applied to $\sigma{:}\bar{\rho}(\neg\Box\psi)$ on Θ if it has been applied to $\sigma{:}\neg\Box\psi$ and $\mathit{Var}(\bar{\rho}(\neg\Box\psi)) \cap \mathit{Var}(\Gamma(\sigma)) = \emptyset$, for $\Gamma(\sigma) = \{\psi \mid \sigma : \Box\psi \in \Theta\}$ the set of $\Box$-formulas occurring at prefix σ[1].

Note that symmetry blocking is a dynamic condition: after being blocked, a $\neg\Box$-formula can be scheduled for expansion if the blocking condition fails in an expansion of the current branch. This can happen because the set $\Gamma(\sigma)$ increases monotonically as the tableau advances. From now on, we consider that branches in $\mathsf{Tab}(\varphi)$ are saturated using symmetry blocking.

4.1 Completeness

To prove completeness of our calculus with symmetry blocking we rely on the intuition that we can extend an incomplete model $\mathcal{M}^\Theta$, built from a saturated open branch Θ to a complete model even when symmetry blocking was used.

Definition 3. *Given an open saturated branch Θ of the tableaux $\mathsf{T} \in \mathsf{Tab}(\varphi)$, we define a model $\mathcal{M}^\Theta = \langle W^\Theta, R^\Theta, V^\Theta\rangle$ as:*

$$\begin{aligned} W^\Theta &= \{\sigma \mid \sigma \textit{ is a prefix on } \Theta\} \\ R^\Theta &= \{(\sigma,\sigma') \mid \sigma,\sigma' \in W^\Theta \textit{ and } \sigma R \sigma' \in \Theta\} \\ V^\Theta(\sigma) &= \{p \mid \sigma : p \in \Theta\}. \end{aligned}$$

Notice that $\mathcal{M}^\Theta$ is always a tree. Given a tree and a permutation sequence, we can construct a new model as follows.

Definition 4 ($\bar{\rho}$-*image* of a tree model). *Given a pointed tree model $\mathcal{M} = \langle w, W, R, V\rangle$ and a permutation sequence $\bar{\rho}$. Let $\mathsf{depth}(v)$ be the distance of v*

[1] A more strict symmetry blocking condition is possible, where instead of requiring $\mathit{Var}(\bar{\rho}(\neg\Box\psi)) \cap \mathit{Var}(\Gamma(\sigma)) = \emptyset$ we verify that the variables at each modal depth of $\bar{\rho}(\neg\Box\psi)$ are disjoint from those in $\Gamma(\sigma)$ (i.e., $\forall n.\, \mathit{Var}(\bar{\rho}(\neg\Box\psi), n) \cap \mathit{Var}(\Gamma(\sigma), n) = \emptyset$). But this more aggressive blocking condition did not have an impact in our experiments. It is easy to find cases where a $\neg\Box$-formula is only blocked under the more strict condition, but we did not find any such case in the test sets we investigated.

*to the root w (in particular $\mathsf{depth}(w) = 0$). Define $\mathcal{M}_{\bar{\rho}} = \langle w, W_{\bar{\rho}}, R_{\bar{\rho}}, V_{\bar{\rho}}\rangle$ (the $\bar{\rho}$-*image *of $\mathcal{M}$) as the model identical to $\mathcal{M}$ except that*

$$V_{\bar{\rho}}(v) = \bar{\rho}(\mathsf{depth}(v) + 1)(L_{V(v)}) \cap \mathsf{PROP}.$$

Given a model $\mathcal{M} = \langle W, R, V\rangle$ and an element $w \in W$, let $W[w]$ denote the set of all elements that are reachable from w (by the reflexive and transitive closure of the accessibility relation). Let $\mathcal{M}[w] = \langle w, W[w], R{\restriction}W[w], V{\restriction}W[w]\rangle$ denote the sub-model of $\mathcal{M}$ rooted at w.

Given Θ a saturated open branch, let Σ be the set of prefixes added to Θ by the application of the ($\Diamond$) rule to a $\neg\Box$-formula that has a symmetric $\neg\Box$-formula blocked by symmetry blocking. Intuitively, Σ contains the roots of the sub-models that need a symmetric counterpart in the completion of $\mathcal{M}^{\Theta}$.

Let $M[\Sigma] = \{\mathcal{M}^{\Theta}[\sigma] \mid \sigma \in \Sigma\}$. This set contains the sub-models to which we need to construct a symmetric sub-model. By $M[\Sigma]_{\bar{\rho}} = \{\mathcal{M}^{\Theta}[\sigma]_{\bar{\rho}} \mid \mathcal{M}^{\Theta}[\sigma] \in M[\Sigma]\}$ we denote the set of $\bar{\rho}$-images corresponding to the set of sub-models $M[\Sigma]$. Intuitively, $M[\Sigma]_{\bar{\rho}}$ is the set of models that we need to "glue" to the model $\mathcal{M}^{\Theta}$ to obtain a complete model.

Definition 5 (Symmetric Extension). *Given a saturated open branch Θ, a model $\mathcal{M}^{\Theta} = \langle W^{\Theta}, R^{\Theta}, V^{\Theta}\rangle$ and a set of symmetric pointed sub-models $M[\Sigma]_{\bar{\rho}}$. The symmetric extension of $\mathcal{M}^{\Theta}$ is the model $\mathcal{M}^{\Theta}_{\bar{\rho}} = \langle W^{\Theta}_{\bar{\rho}}, R^{\Theta}_{\bar{\rho}}, V^{\Theta}_{\bar{\rho}}\rangle$ where:*

$$\begin{aligned} W^{\Theta}_{\bar{\rho}} &= W^{\Theta} \uplus \biguplus W \\ R^{\Theta}_{\bar{\rho}} &= R^{\Theta} \uplus \biguplus R \cup \{(\sigma, \tau_{\sigma'}) \mid (\sigma, \sigma') \in R^{\Theta}\} \\ V^{\Theta}_{\bar{\rho}}(\sigma_i) &= V^{\Theta} \uplus \biguplus V \end{aligned}$$

for all $\langle \tau_{\sigma'}, W, R, V\rangle \in M[\Sigma]_{\bar{\rho}}$, where $\tau_{\sigma'}$ is the element corresponding to σ' in the disjoint union.

Note that we are *gluing* the symmetric sub-models to the original model by adding an edge from the element σ to the root, $\tau_{\sigma'}$, of the symmetric sub-model if there is an edge from σ to the root, σ', of the original sub-model. To be sure that this construction is sound we have to check that after adding these sub-models to the original model, the $\Box$-formulas holding at σ, $\Gamma(\sigma)$, still hold.

The following lemma is the key to prove completeness of our calculus. First recall the following well known result.

Proposition 3. *Let φ be a modal formula and* PROP *a set of propositional variables such that $\mathit{Var}(\varphi) \subseteq \mathsf{PROP}$. Let $\mathcal{M} = \langle W, R, V\rangle$ be a model such that $V : W \mapsto \mathcal{P}(\mathsf{PROP})$. Then $\mathcal{M}, w \models \varphi$ iff $\mathcal{M}{\restriction}\mathit{Var}(\varphi), w \models \varphi$.*

Lemma 1. *Let φ be a modal CNF formula, $\bar{\rho}$ a symmetry of φ and Θ be a saturated open branch of $\mathsf{T} \in \mathsf{Tab}(\varphi)$. Let σ be a prefix such that $\sigma{:}\neg\Box\psi \in \Theta$ and let $\sigma{:}\bar{\rho}(\neg\Box\psi) \in \Theta$ be a blocked formula. Then $\bar{\rho}(\psi) \bigwedge \Gamma(\sigma)$ is satisfiable.*

Proof. Given that Θ is a saturated open branch, we know that $\sigma R \sigma' \in \Theta$ and that $\sigma' : \psi \bigwedge \Gamma(\sigma) \in \Theta$. From Θ we can construct a model $\mathcal{M}^{\Theta} = \langle W^{\Theta}, R^{\Theta}, V^{\Theta}\rangle$ such that $\mathcal{M}^{\Theta}, 0 \models \varphi$ and, in particular, $\mathcal{M}^{\Theta}, \sigma' \models \psi \bigwedge \Gamma(\sigma)$ with $(\sigma, \sigma') \in R^{\Theta}$.

Let $\mathcal{M}^{\Theta}[\sigma'] = \langle \sigma', W', R', V' \rangle$ be the sub-model rooted at σ' with $W' = W^{\Theta}[\sigma']$, $R' = R^{\Theta}\restriction W^{\Theta}[\sigma']$ and $V' = V^{\Theta}\restriction W^{\Theta}[\sigma']$. Then $\mathcal{M}^{\Theta}[\sigma'] \models \psi \bigwedge \Gamma(\sigma)$. Let $\mathcal{N} = \mathcal{M}^{\Theta}[\sigma']\restriction \mathit{Var}(\psi) = \langle \sigma', W', R', V'_{\mathcal{N}} \rangle$ and $\mathcal{R} = \mathcal{M}^{\Theta}[\sigma']\restriction \mathit{Var}(\Gamma(\sigma)) = \langle \sigma', W', R', V'_{\mathcal{R}} \rangle$. By Proposition 3 $\mathcal{N} \models \psi$ and $\mathcal{R} \models \bigwedge \Gamma(\sigma)$.

Let $\mathcal{N}' = \bar{\rho}(\mathcal{N}) = \langle \sigma', W', R', V''_{\mathcal{N}} \rangle$. By construction, $\mathcal{N} \underline{\rightarrow}_{\bar{\rho}} \mathcal{N}'$ and therefore $\mathcal{N}' \models \bar{\rho}(\varphi)$. Finally, let $\mathcal{U} = \mathcal{N}' \cup \mathcal{R} = \langle \sigma, W', R', V''' \rangle$ where $V'''(w) = V''_{\mathcal{N}}(w) \cup V'_{\mathcal{R}}(w)$ for all $w \in W'$. By the symmetry blocking condition we know that $\mathit{Var}(\bar{\rho}(\psi)) \cup \mathit{Var}(\Gamma(\sigma)) = \emptyset$ and therefore $L_{V''_{\mathcal{N}}(w)} \cap L_{V'_{\mathcal{R}}(w)} = \emptyset$ for all $w \in W'$. It follows that no contradiction will arise when doing $V''_{\mathcal{N}}(w) \cup V'_{\mathcal{R}}(w)$ and hence that the valuation function $V'''(w)$ is well defined.

Now we have to prove that $\mathcal{U} \models \bar{\rho}(\psi) \bigwedge \Gamma(\sigma)$. First we prove that $\mathcal{U} \models \bar{\rho}(\psi)$. Take the restriction of $\mathcal{U}$ to $\mathit{Var}(\bar{\rho}(\psi))$, $\mathcal{U}\restriction \mathit{Var}(\bar{\rho}(\psi))$. By construction of $\mathcal{U}$, we know that $\mathcal{U} \restriction \mathit{Var}(\bar{\rho}(\psi)) = \mathcal{N}'$ and that $\mathcal{N}' \models \bar{\rho}(\psi)$. By Proposition 3, $\mathcal{U} \models \bar{\rho}(\psi)$. That $\mathcal{U} \models \bigwedge \Gamma(\sigma)$ holds, follows by the same argument using the model $\mathcal{R}$.

We are now ready to prove a correspondence between formulas in the branch Θ and truth in the symmetric extension of model built from it.

Lemma 2. *Let Θ be a saturated open branch of a tableau $\mathsf{T} \in \mathsf{Tab}(\varphi)$ and $\bar{\rho}$ a symmetry of φ. For any formula $\sigma : \psi \in \Theta$ we have that $\mathcal{M}^{\Theta}_{\bar{\rho}}, \sigma \models \psi$.*

Proof. The proof is by induction on the syntactic structure of ψ.

$[\psi = p]$ By definition, $\sigma \in V^{\Theta}_{\bar{\rho}}(p)$. This implies $\mathcal{M}^{\Theta}_{\bar{\rho}}, \sigma \models p$.

$[\psi = \neg p]$ Since Θ is open, $\sigma{:}p \notin \Theta$. Thus $\sigma \notin V^{\Theta}_{\bar{\rho}}(p)$, which implies $\mathcal{M}^{\Theta}_{\bar{\rho}}, \sigma \models \neg p$.

$[\psi = \chi \wedge \theta]$ and $[\psi = \chi \vee \theta]$ are trivial, by application of the corresponding tableau rules and the induction hypothesis.

$[\psi = \neg\Box\theta]$ We have to consider two cases: a) $\neg\Box\theta$ has been expanded by the application of the $(\Diamond)$ rule. By saturation of $(\Diamond)$, $\sigma R \sigma'$, $\sigma'{:}\theta \in \Theta$. By definition of $R^{\Theta}_{\bar{\rho}}$ and induction hypothesis: $(\sigma, \sigma') \in R^{\Theta}_{\bar{\rho}}$ and $\mathcal{M}^{\Theta}_{\bar{\rho}}, \sigma' \models \theta$. Combining this, we obtain $\mathcal{M}^{\Theta}_{\bar{\rho}}, \sigma \models \neg\Box\theta$, as required. b) $\neg\Box\theta$ has been blocked by the application of symmetry blocking. In this case, $\neg\Box\theta = \bar{\rho}(\neg\Box\chi) = \neg\Box\bar{\rho}(\chi)$. By saturation of $(\Diamond)$ we have that $\sigma R \sigma'$, $\sigma'{:}\chi \in \Theta$. Moreover, we have that $(\sigma, \sigma') \in R^{\Theta}$ and that $\mathcal{M}^{\Theta}, \sigma' \models \chi$. By definition of the symmetric extension of $\mathcal{M}^{\Theta}$ we have that $(\sigma, \tau_{\sigma'}) \in R^{\Theta}_{\bar{\rho}}$ and $\mathcal{M}^{\Theta}_{\bar{\rho}}, \tau_{\sigma'} \models \bar{\rho}(\chi)$. Which implies that $\mathcal{M}^{\Theta}_{\bar{\rho}}, \sigma \models \neg\Box\bar{\rho}(\chi) = \neg\Box\theta$.

$[\varphi = \Box\theta]$ If there is no state σ' such that $(\sigma, \sigma') \in R^{\Theta}_{\bar{\rho}}$ then this holds trivially. Otherwise, let σ' be such that $(\sigma, \sigma') \in R^{\Theta}_{\bar{\rho}}$. By definition of $R^{\Theta}_{\bar{\rho}}$ it must be the case that $\sigma{:}\neg\Box\chi \in \Theta$ and $\sigma R \sigma' \in \Theta$. We must consider two cases: a) if $\sigma{:}\neg\Box\chi$ has not a symmetric counterpart, i.e., it is not blocking a formula $\sigma : \neg\Box\bar{\rho}(\chi)$ then, given that $\sigma : \Box\theta \in \Theta$, by saturation of $(\Box)$, we have that $\sigma'{:}\theta \in \Theta$. By inductive hypothesis, we have that $\mathcal{M}^{\Theta}_{\bar{\rho}}, \sigma' \models \theta$. From this it follows that $\mathcal{M}^{\Theta}_{\bar{\rho}}, \sigma \models \Box\theta$ as required. b) If it is the case that $\sigma{:}\neg\Box\chi$ is blocking $\sigma{:}\neg\Box\bar{\rho}(\chi)$, then, by the definition of the symmetric extension $\mathcal{M}^{\Theta}_{\bar{\rho}}$ and Lemma 1, we have that $(\sigma, \tau_{\sigma'}) \in R^{\Theta}_{\bar{\rho}}$ and $\mathcal{M}^{\Theta}_{\bar{\rho}}, \tau_{\sigma'} \models \bar{\rho}(\chi) \wedge \Gamma(\sigma)$. Given that $\theta \in \Gamma(\sigma)$ then, $\mathcal{M}^{\Theta}_{\bar{\rho}}, \tau_{\sigma'} \models \theta$. From what it follows that $\mathcal{M}^{\Theta}_{\bar{\rho}}, \sigma \models \Box\theta$ as required.

Theorem 2. *The tableau calculus with symmetry blocking for the modal logic $\mathcal{BML}$ is sound and complete.*

Proof. The tableaux calculus we introduced is clearly sound for $\mathcal{BML}$ and symmetry blocking cannot affect soundness. For completeness, let Θ be an open saturated branch of the tableau $\mathsf{T} \in \mathsf{Tab}(\varphi)$. Since $0 : \varphi \in \Theta$, by Lemma 2 we get that φ is satisfiable.

5 Experimental Evaluation

In this section, we empirically evaluate the detection of symmetries and the effect of symmetry blocking in modal benchmarks.

For testing we use `HTab` [19], a tableaux prover developed in Haskell, for the hybrid logic $\mathcal{H}(:, \mathsf{E}, \mathsf{D}, \Diamond^{-}, \downarrow)$ with reflexive, transitive and symmetric modalities[2]. `HTab` includes a series of optimizations that are enabled by default, namely, semantic branching, dependency-directed backtracking, lazy branching, unit propagation and eager unit propagation. For symmetry detection we use Bliss [20], a graph automorphism tool. Bliss takes as input a graph specification and returns a set of generators for the automorphism group of the graph. For each formula the computed symmetries are saved in file. A formula together with its symmetry file are given as input to `HTab`. All tests are run on an Intel Core i7 2.93GHz with 16GB of RAM.

5.1 Symmetry Detection

In Section 3 we presented a graph construction to compute symmetries of modal formulas. It remains to test how often symmetries appear in modal benchmark and how hard it is to actually find them. Our testbed is made of 1134 structured instances: 378 instances from the Logics Workbench Benchmark for $\mathcal{BML}$ (LWB_K) [6] (distributed in 9 problem classes)[3] and 756 instances from the QBFLib Benchmarks [17] (distributed in 12 problem classes). Instances from the QBFLib benchmarks were translated to $\mathcal{BML}$ using a variation of Ladner's translation [22] that reduces the modal depth of the resulting formula. For each instance we generate the corresponding graph and feed it to Bliss.

Table 1 shows the results for the LWB_K and QBFLib benchmarks. Columns #Inst and #Sym are the number of instances and the number of instances with at least one symmetry, respectively. Column T is the total time, in seconds, needed to process all instances. The table clearly shows that both benchmarks are highly symmetric. Formulas in QBFLib are large (ranging into the 250 megabytes after translation into $\mathcal{BML}$) which explains the difference in the time required to process all instances.

[2] Download page: `http://www.glyc.dc.uba.ar/intohylo/htab.php`

[3] We use negate the formulas in LWB_K before constructing the tableaux. *_p classes are unsatisfiable, and *_n classes are satisfiable.

Table 1. Detected Symmetries

	#Inst	#Sym	T
LWB_K	378	208	10.2
QBFLib	756	746	16656

Table 2. Symmetries in LWB_K

Class	#Inst	#Sym	AvGen
`k_branch`	42	42	12
`k_d4`	42	0	0
`k_dum`	42	0	0
`k_grz`	42	42	4
`k_lin`	42	1	1
`k_path`	42	42	35
`k_ph`	42	39	1
`k_poly`	42	42	18
`k_t4p`	42	0	0

The LWB_K benchmark presents a behavior that coincide with our expectations: the existence of symmetries is driven by the codification used in each problem class. Table 2 shows detailed results for this benchmark. Column AvGen is the average number of generators. It shows that some problem classes (`k_branch`, `k_path`, `k_grz`, `k_ph` and `k_poly`) exhibit a large amount of symmetries while others exhibit none (`k_d4`, `k_dum`, `k_t4p`) or very few symmetries (`k_lin`). The QBFLib benchmark also exhibit a large amount of symmetries: 11 of 12 problem classes have a 100% of highly symmetric instances, although in this case, the translation from QBF to $\mathcal{BML}$ accounts for a large number of the detected symmetries (see [5] for details on symmetry detection for QBF formulas). With respect to efficiency, Table 1 shows that for the LWB_K benchmark the time required to compute the symmetries is negligible. For the QBFLib benchmark, it greatly varies depending on the problem class and is directly correlated to the size of the instances.

We also tested symmetry detection on a random testbed. The testbed contains 19000 formulas in CNF generated using hGen [3]. We fix the maximum modal depth of the formulas (D) to 3. Instances are distributed in 10 sets. For each set we fix the number of propositional variables (N) (from 10 to 500) and vary the number of clauses (L) to get different values of the ratio clauses-to-variables (L/N). This ratio is a good indicator of the satisfiability of the formula: formulas with smaller value of L/N are likely to be satisfiable, whilst formulas with greater values of L/N are often unsatisfiable. Each set contain 100 instances for 19 different values of the ratio L/N (from 0.2 to 35). Notice that formula size grows with L/N (larger values of L/N are obtained by the generation of a larger number of clauses). As expected, the number of symmetries diminish with the addition of new, random clauses.

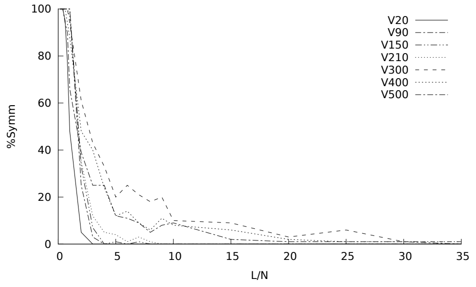


Fig. 1. % of random symmetric instances

Figure 1 shows the percentage of symmetric instances for each value of the ratio L/N. The figure shows that for small values of L/N we find many symmetric instances even in randomly generated formulas. As we increase the value of the ratio, the number of symmetric instances rapidly diminish. Again this coincides with expectations: large values of L/N results from a high number of clauses in the instances, reducing the possibility of symmetries.

5.2 Symmetry Blocking

The implementation of symmetry blocking (SB) in HTab is straightforward: whenever there is a $\neg\Box$-formula scheduled for expansion, the solver checks if there is a symmetric formula already expanded. If this is the case, it blocks the $\neg\Box$-formula and continues with the application of the remaining rules. The solver only verifies the blocking condition if it gets a saturated open branch. If the blocking condition holds for all blocked formulas the solver terminates. Otherwise it reschedules formulas for further expansion.

Our testbed includes 954 symmetric instances from the LWB_K and QBFLib benchmarks (for time constraints and space limitations, we do not report results on random formulas). Table 3 presents the results with and without symmetry blocking (HTab+SB and HTab, respectively). Columns T_1 and T_2 are total times, including symmetry computation, in seconds, on the complete testbed, including and excluding timeouts, respectively (timeout was set to 600 seconds). It shows that HTab+SB outperforms HTab: HTab+SB requires less time to solve all the instances and solves 7 instances more than HTab (HTab+SB is able to solve 9 instances that timeout with HTab, but timeouts in other 2 that HTab is able to solve). It also shows that there is a large number of formulas with timeouts (mostly from QBFLib's formulas)[4].

Table 3. Total Times with SB

Solver	#Suc	#TO	T_1	T_2
HTab+SB	318	636	9657	391167
HTab	311	643	10634	396434

Figure 2 presents a scatter plot of the running times for the 320 formulas that succeed in at least one of the configurations. The x axis gives the running times of HTab without symmetry blocking, whereas the y axis gives the running times of HTab+SB. Each point represents an instance and its horizontal and vertical coordinates represent the time necessary to solve it in seconds. Points on the rightmost and topmost edges

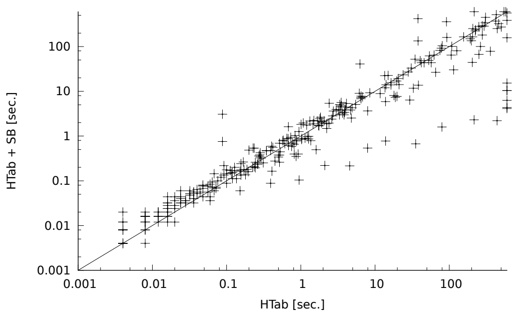


Fig. 2. Performance of HTab vs. HTab with symmetry blocking on all formulas

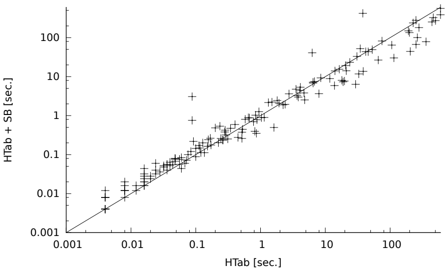
[4] Large formulas from QBFLib often resulted in timeouts. Time constraints put restrictions on the timeout value we could use. We are currently running further tests with larger timeouts.

represent timeout. Notice that a logscale is used, so that gain or degradation to the far right and far top are exponentially more relevant. Approximately half of the instances report a performance gain while the other half report a slight performance degradation. Degradation is due to the extra overhead imposed by the blocking mechanism on instances that never trigger SB. Nevertheless, degradation is negligible for most of the instances.

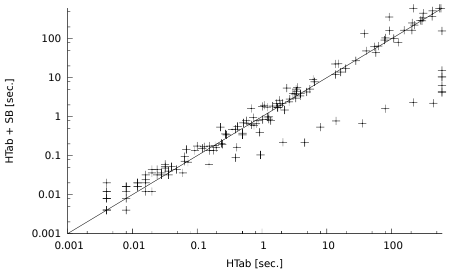
Table 4 shows information about the application of SB. Column #Trig is the number of instances that trigger SB at least once, columns B_1 and B_2 are the number of times that SB is triggered and the number of times that SB is not correct, respectively. For satisfiable instances, SB triggers many times but in most cases the blocking condition fails later in the branch (remember that the blocking condition is dynamic). Figure 3a) shows that there is still an improvement for several instances (44% for all instances, 59% for instances that triggered SB). It also shows that degradation is almost negligible. This tells us that even in cases where SB is not correct, delaying the processing of symmetric formulas is beneficial because the branch has more information available that can avoid branching or close the branch more rapidly.

Table 4. Applications of SB

Status	#Inst	#Trig	B_1	B_2
Satisfiable	157	73	6319	6278
Unsatisfiable	163	79	1038	87



a) Satisfiable formulas



b) Unsatisfiable formulas

Fig. 3. Performance of `HTab` vs. `HTab`+SB

For unsatisfiable instances, SB triggers less often than for satisfiable instances, but most of the blockings are correct (but notice that SB is not validated if the branch closes). Figure 3b) shows a great improvement for several instances. Also `HTab`+SB proves 7 more instances than `HTab`. Degradation is also more noticeable for some instances. A possible explanation is the following: if the blocked formula plays no role in the unsatisfiability of the problem, blocking it avoids unnecessary work resulting in a performance gain. If it plays a role in the unsatisfiability, the solver might be forced to process formulas that would not be processed otherwise, resulting in a performance degradation.

Table 5. Effect of SB on the LWB_K

Class	HTab+SB			HTab		
	n_{100}	n_{600}	T	n_{100}	n_{600}	T
`k_branch_p`	21	21	59.760	13	15	4402.130
`k_branch_n`	9	10	7010.200	8	10	7197.000
`k_grz_p`	21	21	0.508	21	21	0.276
`k_grz_n`	21	21	0.632	21	21	0.380
`k_path_p`	21	21	4.542	21	21	3.812
`k_path_n`	21	21	5.348	21	21	3.792
`k_ph_p`	7	8	8116.900	7	8	8095.48
`k_ph_n`	21	21	177.560	21	21	178.579
`k_poly_p`	21	21	29.068	21	21	22.949
`k_poly_n`	21	21	29.534	21	21	24.229

Table 5 presents detailed results for the LWB_K benchmark. We show the number of the largest formula that could be solved within a time limit of 100 (n_{100}) and a time limit of 600 (n_{600}), and the total time required to process all the class, including timeouts (T). It shows that the effectiveness of SB is highly dependent on the problem class. For some classes SB provide an important performance gain (e.g., `k_branch`). On the other hand, for several highly symmetric classes, SB does not make a difference as it never gets triggered. In other words, symmetric blocking only addresses a small part of the symmetries usually present in modal formulas.

6 Conclusions

In this paper we exploited symmetries in a modal tableuax. First we presented a method to detect symmetries for $\mathcal{BML}$ that extend the method presented in [4] to detect layered symmetries. Given a formula φ it generates a graph such that the automorphism group of the graph is isomorphic to the symmetry group of the formula. Layering is incorporated by duplicating the literal nodes occurring at different modal depth. Then we presented a blocking mechanism for a modal tableaux that uses symmetry information only. This mechanism, called *symmetry blocking*, blocks the application of the ($\Diamond$) rule if it has been already applied to a symmetric $\neg\Box$-formula. Finally we evaluated empirically both, the detection technique and the blocking mechanism. Experimental results shows that structured modal benchmarks are highly symmetric and that our detection algorithm is efficient at computing symmetries. In the case of symmetry blocking, results shows that the applicability of the blocking mechanism highly depends on the problem class at hand, and that important performance gains can be obtained in some classes while imposing only a reasonable overhead on problem classes not suited for this blocking mechanism. Further testing is needed, and also other approaches to exploiting modal symmetries like, e.g., symmetry breaking predicates [12].

References

1. Aloul, F., Ramani, A., Markov, I., Sakallah, K.: Solving difficult instances of Boolean satisfiability in the presence of symmetry. IEEE Trans. of CAD 22(9), 1117–1137 (2003)
2. Areces, C., Gennari, R., Heguiabehere, J., de Rijke, M.: Tree-based heuristics in modal theorem proving. In: Proc. of ECAI 2000, Berlin, pp. 199–203 (2000)
3. Areces, C., Heguiabehere, J.: hGen: A Random CNF Formula Generator for Hybrid Languages. In: Proc. of M4M-3, Nancy, France (2003)
4. Areces, C., Hoffmann, G., Orbe, E.: Symmetries in modal logics. In: Kesner, D., Viana, P. (eds.) Proc. of LSFA 2012, pp. 27–44 (2013)
5. Audemard, G., Mazure, B., Sais, L.: Dealing with symmetries in quantified boolean formulas. In: Proc. of SAT 2004, pp. 257–262 (2004)
6. Balsiger, P., Heuerding, A., Schwendimann, S.: A benchmark method for the propositional modal logics K, KT, S4. J. of Aut. Reas. 24(3), 297–317 (2000)
7. Benhamou, B., Sais, L.: Theoretical study of symmetries in propositional calculus and applications. In: Kapur, D. (ed.) CADE 1992. LNCS, vol. 607, pp. 281–294. Springer, Heidelberg (1992)
8. Benhamou, B., Sais, L.: Tractability through symmetries in propositional calculus. J. of Aut. Reas. 12(1), 89–102 (1994)
9. Blackburn, P., de Rijke, M., Venema, Y.: Modal Logic. Cambridge University Press (2001)
10. Blackburn, P., van Benthem, J., Wolter, F.: Handbook of Modal Logic. Studies in Logic and Practical Reasoning, vol. 3. Elsevier Science Inc., New York (2006)
11. Crawford, J.: A theoretical analysis of reasoning by symmetry in first-order logic. In: Proc. of AAAI 1992 Work. on Tractable Reasoning, San Jose, pp. 17–22 (1992)
12. Crawford, J., Ginsberg, M., Luks, E., Roy, A.: Symmetry-breaking predicates for search problems. In: Proc. of KR 1996, pp. 148–159 (1996)
13. Darga, P., Liffiton, M., Sakallah, K., Markov, I.: Exploiting structure in symmetry detection for CNF. In: Proc. of DAC 2004, pp. 530–534 (2004)
14. Déharbe, D., Fontaine, P., Merz, S., Woltzenlogel Paleo, B.: Exploiting symmetry in SMT problems. In: Bjørner, N., Sofronie-Stokkermans, V. (eds.) CADE 2011. LNCS, vol. 6803, pp. 222–236. Springer, Heidelberg (2011)
15. Fortin, S.: The graph isomorphism problem. Technical Report 96-20, University of Alberta, Edomonton, Alberta, Canada (1996)
16. Fraleigh, J., Katz, V.: A first course in abstract algebra. Addison-Wesley (2003)
17. Giunchiglia, E., Narizzano, M., Tacchella, A.: Quantified Boolean Formulas satisfiability library, QBFLIB (2001), `http://www.qbflib.org`
18. Goré, R.: Tableau methods for modal and temporal logics. In: D'Agostino, M., Gabbay, D., Hähnle, R., Posegga, J. (eds.) Handbook of Tableau Methods, pp. 297–396. Springer, Netherlands (1999)
19. Hoffmann, G., Areces, C.: Htab: A terminating tableaux system for hybrid logic. In: Proc. of M4M-5 (2007)
20. Junttila, T., Kaski, P.: Engineering an efficient canonical labeling tool for large and sparse graphs. In: Proc. of ALENEX 2007. SIAM (2007)
21. Krishnamurthy, B.: Short proofs for tricky formulas. Act. Inf. 22(3), 253–275 (1985)
22. Ladner, R.: The computational complexity of provability in systems of modal propositional logic. SIAM J. on Comp. 6(3), 467–480 (1977)
23. Luks, E.M.: Isomorphism of graphs of bounded valence can be tested in polynomial time. Journal of Computer and System Sciences 25(1), 42–65 (1982)
24. Patel-Schneider, P., Sebastiani, R.: A new general method to generate random modal formulae for testing decision procedures. J. of Art. Int. Res. 18, 351–389 (2003)

Tableaux for Verification of Data-Centric Processes

Andreas Bauer, Peter Baumgartner, Martin Diller, and Michael Norrish

NICTA[⋆] and Australian National University, Canberra, Australia

Abstract. Current approaches to analyzing dynamic systems are mostly grounded in propositional (temporal) logics. As a consequence, they often lack expressivity for modelling rich data structures and reasoning about them in the course of a computation. To address this problem, we propose a rich modelling framework based on first-order logic over background theories (arithmetics, lists, records, etc) and state transition systems over corresponding interpretations. On the reasoning side, we introduce a tableau calculus for bounded model checking of properties expressed in a certain fragment of CTL* over that first-order logic. We also describe a k-induction scheme on top of that calculus for proving safety properties, and we report on first experiments with a prototypical implementation.

1 Introduction

Current approaches to analyzing dynamic systems are mostly grounded in propositional (temporal) logics. As a consequence, they often lack expressivity for modelling rich data structures and reasoning about them in the course of a computation. To address this problem, we propose an expressive, sorted first-order logic to describe states, a fragment of CTL* to describe systems' evolution, and we introduce a tableau calculus for model checking in that logic. Our approach is based on

– *process fragments* that describe specific tasks of a larger process, inspired by what is known as *declarative business process modelling* [17]. As a result, users do not have to specify a single, large transition system with all possible task interleavings.
– *constraints* for limiting the interactions between the fragments. In this way, users can create many small process fragments whose interconnections are governed by rules that determine which executions are permitted.
– *first-order temporal logic*. Unlike [8], we choose to extend CTL*, *i.e.*, a branching time logic, rather than LTL, since process fragments are essentially annotated graphs and CTL* is, arguably, appropriate to express its properties (*cf.* [6]).
– *sorts* for JSON objects [7], where sorts are governed by a custom, static type system which models and preserves the type information of any input data. JSON objects allow for richly structured data types such as lists and records.

Tableau calculi have been long considered (*e.g.*, [10]) an appropriate and natural reasoning procedure for temporal logics. There is even a tableau procedure for propositional CTL* [18]. However, we are not aware of a first-order logic tableaux calculus

⋆ NICTA is funded by the Australian Government as represented by the Department of Broadband, Communications and the Digital Economy and the Australian Research Council through the ICT Centre of Excellence program.

that accommodates our requirements, hence we devise one (Section 4). We note that we circumvent the difficult problem of loop detection by working in a *bounded* model checking setting, where runs are artificially terminated when they become too long.

The high expressivity of our approach comes at the price of high undecidability, and so practical feasibility is an issue. Ultimately, all our reasoning problems reduce to first-order logic proof obligations, and hence automated reasoning in first-order logic becomes a crucial component. Although we focus on the core logic of our framework, we also report on first experiments with a prototypical implementation that integrates our tableau procedure with the state of the art SMT solver Z3 [14].

Related Work. In the area of business process modelling, the so-called "business artifact" approach pioneered the idea of making data a "first-class citizen" (Nigam and Caswell [16]). The *artifacts* of this approach are records of data values that can change over time due to the modifications performed by *services*, which are formalized using first-order logic. Process analysis answers the following question: given some artifact model, a database providing initial values, and a correctness property in terms of a first-order LTL formula, do all possible artifact changes over time satisfy the correctness property? For the constraints given in Damaggio *et al.* [8], this problem is decidable. We refer to this problem as "concrete model checking" since an initial state has to be given. We are also interested in the generalization thereof, where the set of possible initial states is unconstrained, the "general model checking" problem.

Schuele and Schneider [20] give a categorisation of temporal model checking problems. They differentiate between global model checking techniques, which are basically fix-point iterations, and local techniques, which are inference based and analyse a formula in a top-down fashion by inspecting its syntax tree. As such, both our concrete and general model checking problems fall under the local techniques category.

Bersani *et al.* [4] describe linking SMT-solvers to decide bounded model checking problems over temporal logic extensions. There, LTL with integer constraints is considered, which results in an undecidable satisfiability problem. However, by constraining the number of variables and length of paths, a decidable satisfiability and model checking problem is obtained.

Another example combining data and dynamics is Vianu [23]. This work uses an LTL in which the propositions can be replaced by a background theory statement, in particular FOL, to verify systems whose behaviour is expressible as sequence of database updates. Since the latter results in an infinite-state system, Vianu imposes several restrictions on the database properties, and obtains a PSpace model checking algorithm. In the area of description logics, Hariri *et al.* [12] and Chang *et al.* [5] both present systems that allow for rich queries over dynamically evolving systems. Entities within the systems can be related to one another and characterized in a first-order style, but there is no scope for the use of types such as numbers and lists as in our work.

In Ghilardi *et al.* [9], fragments of first-order linear-time temporal logic with background theories are considered. While the general satisfiability problem of such a logic is necessarily undecidable, the authors identify the quantifier-free fragment, which can be decided in PSpace, given that the background constraints can. Work on temporalizing description logics heads in a similar direction. The challenge there is to determine fragments of, *e.g.*, LTL over description logics so that the desired reasoning services become decidable. See, *e.g.*, Baader *et al.* [1] for recent work.

2 Preliminaries

We work with sorted signatures Σ consisting of a non-empty set *sorts* and function and predicate symbols of fixed arities over these sorts. We assume infinite supplies of variables, one for each sort. A *constant* is a 0-ary function symbol. The (well-sorted Σ-) terms and atoms are defined as usual. We assume Σ contains a predicate symbol $=_s$ (equality) of arity $s \times s$, for every sort $s \in sorts$. Equational atoms, or just *equations*, are written infix, usually without the subscript s, as in $1 + 1 = 2$. We write $\theta[\boldsymbol{x}]$ to indicate that every free variable in the formula θ is among the list $\boldsymbol{x}$ of variables, and we write $\theta[\boldsymbol{t}]$ for the formula obtained from $\theta[\boldsymbol{x}]$ by replacing all its free variables $\boldsymbol{x}$ by the corresponding terms in the list $\boldsymbol{t}$.

We assume a sufficiently rich set of Boolean connectives (such as $\{\neg, \wedge\}$) and the quantifiers $\forall$ and $\exists$. The *well-sorted Σ-formulas*, or just *(FO) formulas* are defined as usual. We are particularly interested in signatures containing (linear) integer arithmetic. For that, we assume $\mathbb{Z} \in sorts$, the $\mathbb{Z}$-sorted constants $0, \pm 1, \pm 2, \ldots$, the function symbols $+$ and $-$, and the predicate symbol $>$, each of the expected arity over $\mathbb{Z}$.

The semantics of our logic is the usual one: a *Σ-interpretation $\mathcal{I}$* consists of non-empty, disjoint sets, called *domains*, one for each sort. We require that the domain for $\mathbb{Z}$ is the set of integers, and that every arithmetic function and predicate symbol (including $=_{\mathbb{Z}}$) is mapped to its obvious function or relation, respectively, over the integers. Indeed, we will later see that we treat other sorts, such as lists and other JSON types as "built-in" (see Section 3). In brief, our modelling framework supports the use of (finite) lists, arrays and records in a monomorphically sorted setting. Thus, we further require that Σ-interpretations interpret the function and predicate symbols associated with these sorts in a way consistent with the intended semantics, which can be given axiomatically. This is consistent with the de-facto TPTP standard [21], so that compliant theorem provers can be applied.

A *(variable) assignment α* is a mapping from the variables into their corresponding domains. Given a formula θ and a pair $(\mathcal{I}, \alpha)$ we say that $(\mathcal{I}, \alpha)$ *satisfies* θ, and write $(\mathcal{I}, \alpha) \models \theta$, iff θ evaluates to true under $\mathcal{I}$ and α in the usual sense (the component α is needed to evaluate the free variables in θ). If θ is closed then α is irrelevant and we can write $\mathcal{I} \models \theta$ instead of $(\mathcal{I}, \alpha) \models \theta$. We say that a closed sentence θ is *valid* (*satisfiable*) iff $\mathcal{I} \models \theta$ for all (some) interpretations $\mathcal{I}$.

Processes in our framework are modeled as state transition systems. A *state transition system* is a tuple $M = (S, I, R)$ where S is a set of *states*, $I \subseteq S$ are the *initial states*, and $R \subseteq S \times S$ the *transition relation*.[1] Throughout this paper, states are mappings from the variables into their corresponding domains, i.e., every state $s \in S$ is an assignment (but in general not every assignment α is a state in S).

Our query language is a fragment of CTL* over first-logic, which we refer to as CTL*(FO). Its syntax is given by the following grammar:

$$\phi ::= \theta \mid \neg\phi \mid \phi \wedge \phi \mid \mathsf{A}\psi \mid \mathsf{E}\psi \qquad \psi ::= \phi \mid \neg\psi \mid \psi \wedge \psi \mid \mathsf{X}\psi \mid \overline{\mathsf{X}}\psi \mid \psi\,\mathsf{U}\,\psi \mid \psi\,\mathsf{R}\,\psi$$

[1] Notice we do not require R to be (left-) total, as runs may be *finite*.

where θ refers to a FO formula, ϕ is called a *state formula* and ψ a *path formula*. The operator $\overline{\mathsf{X}}$ is called "weak next". A CTL*(FO) formula is *pure FO* iff it does not contain any path quantifier and does not contain any temporal operator.

Let $M = (S, I, R)$ be a state transition system as stated above and $s_0 \in S$. A *run r (of M) from* s_0 is a possibly infinite sequence $s_0\, s_1\, s_2\, \cdots$ of states such that $(s_i, s_{i+1}) \in R$. Let $r[i] = s_i$, and r^i the *truncated run* $s_i\, s_{i+1}\, \cdots$. By $|r|$ we denote the number of elements in r or ∞, if r is infinite. Obviously, $r^0 = r$.

For any state formula $\phi \in$ CTL*(FO), interpretation $\mathcal{I}$, and state $s_0 \in S$ we define a satisfaction relation $\models$. It differs somewhat from the usual definition (cf. [6]) in that it is implicitly parametric in a set of *admissible runs (of M)*. We identify the set of admissible runs with its closure under truncation of runs.

A finite run $s_0 \cdots s_n$ is called *finished* if there is no $s \in S$ such that $(s_n, s) \in R$. That is, finished runs do not stop prematurely. The set of *standard runs (of M)* consists of all infinite runs and all finished runs. Unless stated otherwise we assume standard runs.

For any state formula $\phi \in$ CTL*(FO), interpretation $\mathcal{I}$, and state $s_0 \in S$, the satisfaction relation $(\mathcal{I}, s_0) \models \phi$ is defined as follows,

$$\begin{array}{ll}
(\mathcal{I}, s_0) \models \theta & \text{iff } (\mathcal{I}, s_0) \models \theta \\
(\mathcal{I}, s_0) \models \neg\phi & \text{iff } (\mathcal{I}, s_0) \not\models \phi \\
(\mathcal{I}, s_0) \models \phi_1 \wedge \phi_2 & \text{iff } (\mathcal{I}, s_0) \models \phi_1 \text{ and } (\mathcal{I}, s_0) \models \phi_2 \\
(\mathcal{I}, s_0) \models \mathsf{A}\,\psi & \text{iff } (\mathcal{I}, r) \models \psi \text{ for all runs } r \text{ from } s_0 \\
(\mathcal{I}, s_0) \models \mathsf{E}\,\psi & \text{iff } (\mathcal{I}, r) \models \psi \text{ for some run } r \text{ from } s_0,
\end{array}$$

where the satisfaction relation $(\mathcal{I}, r) \models \psi$ for path formulas ψ and admissible r is

$$\begin{array}{ll}
(\mathcal{I}, r) \models \phi & \text{iff } (\mathcal{I}, r[0]) \models \phi \\
(\mathcal{I}, r) \models \neg\psi & \text{iff } (\mathcal{I}, r) \not\models \psi \\
(\mathcal{I}, r) \models \psi_1 \wedge \psi_2 & \text{iff } (\mathcal{I}, r) \models \psi_1 \text{ and } (\mathcal{I}, r) \models \psi_2 \\
(\mathcal{I}, r) \models \mathsf{X}\,\psi & \text{iff } |r| > 1 \text{ and } (\mathcal{I}, r^1) \models \psi \\
(\mathcal{I}, r) \models \overline{\mathsf{X}}\,\psi & \text{iff } |r| \leq 1, \text{ or } |r| > 1 \text{ and } (\mathcal{I}, r^1) \models \psi \\
(\mathcal{I}, r) \models \psi_1\,\mathsf{U}\,\psi_2 & \text{iff there exists a } j \geq 0 \text{ such that } |r| > j \text{ and } (\mathcal{I}, r^j) \models \psi_2, \\
 & \quad \text{and } (\mathcal{I}, r^i) \models \psi_1 \text{ for all } 0 \leq i < j \\
(\mathcal{I}, r) \models \psi_1\,\mathsf{R}\,\psi_2 & \text{iff } (\mathcal{I}, r^i) \models \psi_2 \text{ for all } i < |r|, \text{ or there exists a } j \geq 0 \text{ such that} \\
 & \quad |r| > j,\ (\mathcal{I}, r^j) \models \psi_1 \text{ and } (\mathcal{I}, r^i) \models \psi_2 \text{ for all } 0 \leq i \leq j.
\end{array}$$

We assume the usual "syntactic sugar", which can easily be defined in terms of the above set of operators in the expected way. Note that we distinguish a strong next operator, X, from a weak next operator, $\overline{\mathsf{X}}$, as described in [2]. This gives rise to the following equivalences: $\psi_1\,\mathsf{R}\,\psi_2 \equiv \psi_2 \wedge (\psi_1 \vee \overline{\mathsf{X}}\,(\psi_1\,\mathsf{R}\,\psi_2))$ and $\psi_1\,\mathsf{U}\,\psi_2 \equiv \psi_2 \vee (\psi_1 \wedge \mathsf{X}\,(\psi_1\,\mathsf{U}\,\psi_2))$ as one can easily verify by using the above semantics. This choice is motivated by our bounded model checking algorithm, which has to evaluate CTL*(FO) formulas over finite traces as opposed to infinite ones. For example, when evaluating a safety formula, such as $\mathsf{G}\,\psi$, we want a trace of length n that satisfies ψ in all positions $i \leq n$ to be a model of this formula. On the other hand, if there is no position $i \leq n$, such that ψ is satisfied, we don't want this trace to be a model for $\mathsf{F}\,\psi$. This is achieved in our logic as $\mathsf{G}\,\psi \equiv \psi \wedge \overline{\mathsf{X}}\,\mathsf{G}\,\psi$ and $\mathsf{F}\,\psi \equiv \psi \vee \mathsf{X}\,\mathsf{F}\,\psi$ hold. Note also that $\neg\mathsf{X}\,\psi \not\equiv \mathsf{X}\,\neg\psi$, but $\neg\mathsf{X}\,\psi \equiv \overline{\mathsf{X}}\,\neg\psi$.

3 The Specification Language

We provide a specification language to define processes and the data they manipulate. A concrete specification consists of the sections `TYPES`, `SIGNATURE`, `DEFINITIONS`, `CONSTRAINTS`, and `DIGRAPH`. We explain each section in turn, including sample extracts in each explanation from a business process modelling domain.

Logic for Individual States. Our specification logic stratifies into two levels. The first level uses non-temporal first-order formulas to describe individual states of a system. Formulas at this level are richly *typed*, and may refer to user-defined logical notions.

JSON Values, `TYPES` and `SIGNATURE`. Users capture the states of their systems with *JSON values*. JSON [7] is an untyped framework for writing structured data, including base types such as strings and integers, as well as structure through records (field names coupled with values) and arrays. The JSON syntax is rich enough to represent complex states while remaining human-readable. We layer a simple type-system over JSON, ultimately providing a connection between these types and the sorts of CTL*(FO).

The atomic types of our specification language are `String`, `Bool` and `Integer`. In addition, users can define new types that are built up from these atomic types, the type operators `Array[_]`, `List[_]`, and a syntax for record types, *i.e.*, a list of field names coupled with types for those fields. Types may occur within other type definitions, as long as there are no recursive loops. This restriction means that users cannot specify their own recursive types (such as trees). This restriction does not seem too onerous in practice and makes the axiomatic characterization of the types straightforward.

```
DB = {
  stock: Array[Stock],
  nrStockItems: Integer,
  open: List[Integer],
  gold: Boolean,
  invoice: Bool,
  paid: Bool,
  shipped: Bool }
Stock = {
  ident: String,
  price: Integer,
  available: Integer }
```

The types from the purchase order example are shown on the right. The `DB` type corresponds to the entire system state. The `stock` array holds information about stock items, for each item number 0..`nrStockItems`-1. The `Stock.available` field is the number of items in stock, per item number. The `open` list contains the open order item numbers, those that have not been packed yet. The `gold` bit says whether the customer is a gold customer. The `invoice` filed says whether an invoice has been generated. The `paid` and `shipped` fields control the composition of "process fragments", see below.

There are naturally various operations over terms of the corresponding JSON types that our logic must support. Thus we support arithmetic function and relation symbols (+, <, *etc.*). Support for JSON record types includes functions for accessing fields of objects (the concrete syntax is the familiar "dot notation"; *e.g.*, `s.value`) and for creating new record values by updating field values of old values ("functional record update"). Depending on the nature of the back-end reasoning tool, elements of the signature may be characterized directly in FOL, as is done for the record functions. By contrast, we expect backend reasoning tools to directly support arithmetic, arrays and lists, with the usual operators on them, freeing us from providing a FOL axiomatization for the latter (this is not a critical limitation). We refer to this extended language as *JSON Logic*, and talk of *JSON sentences* and *JSON terms etc.*

In addition, users can declare and define their own functions, predicates and relations over these types. Those entities without definition will be uninterpreted. All such, whether or not they are later defined, are listed in the `SIGNATURE` section along with their types. For example, the `completed` predicate on `Status` arguments is given in this section with the syntax `completed: [Status] -> Bool`.

`DEFINITIONS` and `CONSTRAINTS`. The `DEFINITIONS` section consists of a set of FO JSON sentences, providing the semantics for (some of) the free predicate and function symbols declared in the `SIGNATURE`. Let *defs* be the image of the `DEFINITIONS` section under translation into CTL*(FO).

The `CONSTRAINTS` sections consists of a set of JSON CTL*(FO) path formulas. Unlike `DEFINITIONS`, the free variable `DB`-sorted variable *db* is permitted. It represents the database at the current time point. The intention is to provide additional constraints on the runs considered in the reasoning problems below. Let *constraints* be the image of the `CONSTRAINTS` section under translation into CTL*(FO).

Examples of definitions and constraints occur in Figure 1. One such is the definition of the mentioned `completed` predicate over system statuses. The sample constraint is a temporal property using the "weak until" W operator. It encodes the rule that customers without "gold" status can never have their order shipped before they have paid.

Adding Dynamics. Above the state-based level of the previous section, we allow users to define a "process fragment"-based dynamics for their systems by means of *process graphs*. Formally, a *process graph* G is a directed labeled graph (N, E), where N is a set of *nodes* and $E \subseteq N \times N$ is a set of *edges*. Exactly one node must be labelled as an "init node". Each node can be labelled as an "entry node" or "exit node" (or both). A *guard* is a FO formula with free variables at most $\{db\}$; an *update term* is a FO term with free variables at most $\{db\}$. Entry nodes and edges always have both a guard and an update term attached to them, which are denoted by $guard(n)$ and $upd(n)$ for entry nodes n, respectively, and analogously for edges.

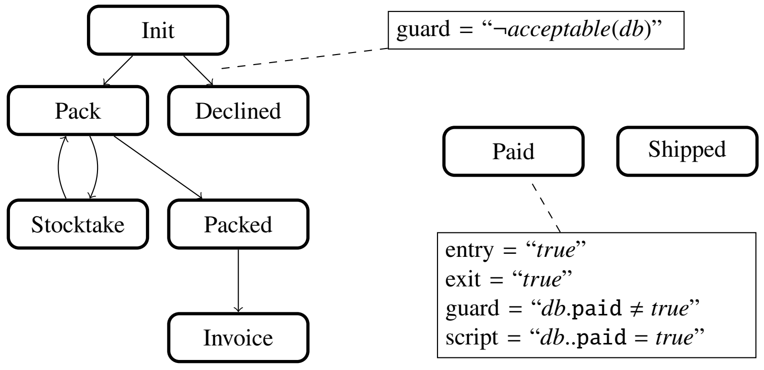
The concrete syntax for process graphs should be obvious from our running example. Every `script`, a sequence of assignments, is taken as an update term. The semantics of entry/exit nodes is defined by implicitly putting an edge between every exit node and every entry node and using the entry node's guard and script for the edge.

We capture this intuition formally and in a uniform way by defining a *labelled edge relation* consisting exactly of the quadruples $m \xrightarrow{\gamma,u} n$ such that $m, n \in N$ and either $(m, n) \in E$, $\gamma = guard(m, n)$ and $u = upd(m, n)$, or m is an exit node, n is an entry node, $\gamma = guard(n)$ and $u = upd(n)$.

The labelled edge relation induces a state transition system $M = (S, I, R)$ as follows. The states S are all assignments s of the form $\{\ell \mapsto n,\ db \mapsto d\}$ where $n \in N$ and d is a domain element of sort `DB`. Notice that ℓ and db are fixed. Then,

$$I \overset{\text{def}}{=} \{s \in S \mid s(\ell) = n_0\}$$

$$R \overset{\text{def}}{=} \{(s, s') \in S \times S \mid s(\ell) \xrightarrow{\gamma,u} s'(\ell),\ (\mathcal{I}, \{db \mapsto s(db)\}) \models \gamma[db], \text{ and } s'(db) = (\mathcal{I}, \{db \mapsto s(db)\})(u[db])$$


Init
guard = "¬*acceptable*(*db*)"
Pack
Declined
Paid
Shipped
Stocktake
Packed
entry = "*true*"
exit = "*true*"
guard = "*db*.paid ≠ *true*"
script = "*db*..paid = *true*"
Invoice

Definitions:
completed: $\forall db{:}\texttt{DB}\ .\ (completed(db) \Leftrightarrow (db.\texttt{paid} = true \wedge db.\texttt{shipped} = true))$
acceptable: $\forall db{:}\texttt{DB}\ .\ (acceptable(db) \Leftrightarrow db.\texttt{open} \neq [|\,|])$
Constraints:
nongold: $(db.\texttt{gold} = false \Rightarrow (db.\texttt{shipped} = false\ \mathsf{W}\ db.\texttt{paid} = true))$

Fig. 1. Model of a purchase order system as process fragments and definitions

Notice the transition relation R depends on the interpretation $\mathcal{I}$, which is fixed at the outset. If a guard evaluates to false under $\mathcal{I}$ and the current state, then the edge it is on is just "not there". Otherwise the state must be updated as specified by the update term.

We can now explain the dynamics of our running example, a system for handling purchase orders. The purpose of the modelled system is to accept incoming purchase orders and process them further (packing, shipping, etc.), or to decline them straight away if there are problems. The dynamics of the model is depicted as a graph in Fig. 1. It is comprised of three *(process) fragments*: the biggest fragment on the left, and beside it the two one-node fragments labelled "Paid" and "Shipped".

The depicted model's initial node ("Init") is where it waits for a purchase order to arrive. Subsequently, the system can either start to pack (*i.e.*, enter node "Pack"), or decline the order (*i.e.*, enter node "Declined"). An order can be declined if the depicted guard (¬*acceptable*(*db*)) in the annotation of the edge is satisfied. The predicate *acceptable* is defined in the `DEFINITIONS` section of our input specification.

If the order is not declined, an attempt will be made to pack its constituents. As long as the `open` list is not empty, the loop between "Pack" and "Stocktake" packs all items one after the other. Not all guards and scripts are depicted in Figure 1. For example, there is an edge from "Stocktake" to "Pack" labelled with

guard = "$db.\texttt{stock}[head(db.\texttt{open})].\texttt{available} > 0$"
script = "$db.\texttt{stock}[head(db.\texttt{open})].\texttt{available} =$
$db.\texttt{stock}[head(db.\texttt{open})].\texttt{available} - 1;\ db.\texttt{open} = tail(db.\texttt{open})$"

Upon completion, the "Invoice" state is reached, followed by composition with the fragments "Paid" and "Shipped". The "Shipped" fragment has a guard and script analogously to that of "Paid". The `guards` make it impossible to compose these fragments repeatedly, otherwise their composition is subject only to the "nongold" constraint. The intended final states are those that satisfy the "completed" predicate.

Reasoning Problems. Assume as given a specification. Let Σ be the induced signature with sorts *sorts*. Let *defs*, *constraints* and $M = (S, I, R)$ be as defined above. In terms of our specification language we are interested in the following reasoning problems. In each of them, let $\psi[db]$ be a path formula, in this context called the *query*.

Concrete satisfiability problem: Given an initial state $s_0 \in I$.
Is there a Σ-interpretation $\mathcal{I}$ such that $(\mathcal{I}, s_0) \models \mathsf{E}\,(defs \wedge constraints \wedge \psi)$ holds?

General satisfiability problem: Is there an initial state $s_0 \in I$ and a Σ-interpretation $\mathcal{I}$ such that $(\mathcal{I}, s_0) \models \mathsf{E}\,(defs \wedge constraints \wedge \psi)$ holds?

That is, the concrete *vs.* general dimension distinguishes whether an initial assignment is fixed or not. The concrete problems are interesting for implementing deployed systems and runtime verification. For, if the definitions and constraints are sufficiently restricted (*e.g.*, non-recursive definitions and constraints whose quantifiers range over finite domains) all state transitions can be effectively executed. See Section 6 for examples of reasoning problems.

4 Tableaux for CTL*(FO)

In this section we introduce a tableau calculus for the reasoning problems in Section 3. Without loss of generality it suffices to consider the general satisfiability problem only. (Pragmatics aside, any concrete satisfiability problem can be encoded as a general one as a set of equations in the `CONSTRAINTS` section). With the abbreviation $\psi_0 = defs \wedge constraints \wedge \psi$ the reasoning problem, hence, is to ask whether $(\mathcal{I}, s_0) \models \mathsf{E}\,\psi_0$ holds for some $s_0 \in I$ and Σ-interpretation $\mathcal{I}$. In fact, Ψ_0 can be any path formula in the free variable *db*.

It comes in handy to assume the signature Σ contains a `DB`-sorted constant db, representing the initial database, and that Σ contains a distinguished sort "Node" and the nodes N from the process graph as constants. We assume $\mathcal{I}(n) = n$ for every $n \in N$.

We formulate the calculus' inference rules as operators on sets of sequents. A *sequent* is an expression of the form $(n, t, l) \vdash_Q \Phi$ where $n \in N$, t is a ground term of sort `DB`, $l \geq 0$ is an integer, $Q \in \{\mathsf{E}, \mathsf{A}\}$ is a path quantifier, and Φ is a (possibly empty) set of CTL*(FO) formulas in negation normal form with free variables at most $\{db\}$. When we write $s \vdash_Q \Phi$ we mean $(n, t, l) \vdash_Q \Phi$ for some $(n, t, l) = s$, and when we write $s \vdash_Q \phi, \Phi$ we mean $s \vdash_Q \{\phi\} \cup \Phi$.

Informally, the sequent $(n, t, d) \vdash_Q \Phi$ means that the computation has reached after l steps ("length") into a run the graph node n with a database represented by t and that database satisfies $Q\,\Phi$. For example, t could be $\mathsf{db}\{\mathsf{open} = [|1, 3, 2|]\}$ (in sugared notation) which stands for an update of the initial database db updated on its open-field with the list $[|1, 3, 2|]$, and $Q\,\Phi$ could be the formula $\mathsf{A}\,\mathsf{G}\,db.\mathsf{open} \neq [|\,|]$. The calculus analyses a given sequent by decomposing its formula Φ according to its boolean operators, path quantifiers and temporal operators. An additional implicit boolean structure is given by reading the formulas Φ in $s \vdash_{\mathsf{E}} \Phi$ conjunctively, and reading the formulas Φ in $s \vdash_{\mathsf{A}} \Phi$ disjunctively.[2] The purpose is to derive a set of sequents with only classical, i.e., pure FO formulas in Φ, so that a first-order satisfiability check results.

[2] These structures are in general not decomposable, as A does not distribute over "or" and E does not distribute over "and", and so the calculus needs to deal with that explicitly.

We are using notions around tableau calculi in a standard way, and so it suffices to summarize the key points. The nodes in our tableaux are labelled with sets Σ of sequents or the special sign FAIL, which indicates branch closure. Logically, FAIL is taken as an (any) unsatisfiability set of sequents, e.g., $\{(n_0, \mathsf{db}, 0) \vdash_{\mathsf{A}} \emptyset\}$. We often write $\sigma;\Sigma$ instead of $\{\sigma\} \cup \Sigma$, and we often identify the node with its label. A *derivation $\mathcal{D}$ (from a path formula ψ_0)* is a sequence of tableaux, starting from a root node only tableau labelled with $\{(n_0, \mathsf{db}, 0) \vdash_{\mathsf{E}} \psi_0\}$. A successor tableaux is obtained by applying one of the inference rules below to a non-FAIL leaf of the current tableau and branching out with the conclusions. A *refutation (of ψ_0)* is a derivation from ψ_0 of a tableau whose leaves are all FAIL. We suppose a notion of *fair* derivations as commonly used with tableau calculi. Intuitively, a derivation is *fair* iff it is a refutation or no inference rule application is deferred forever.

In the inference rules below we use the following notions. A formula is *classical* iff it contains no path quantifier and no temporal operator. A formula is a *modal atom* iff its top-level operator is a path quantifier or a temporal operator. A sequent $s \vdash_Q \Phi$ is *classical* if all formulas in Φ are classical. We define $\mathrm{form}_{\mathsf{A}}(\Phi) \overset{\mathrm{def}}{=} \mathsf{A}(\bigvee \Phi)$ and $\mathrm{form}_{\mathsf{E}}(\Phi) \overset{\mathrm{def}}{=} \mathsf{E}(\bigwedge \Phi)$ in order to reflect the disjunctive/conjunctive reading of Φ depending on a path quantifier context. If all formulas in Φ are classical then the path quantifier is semantically irrelevant and omitted from $\mathrm{form}_Q(\Phi)$.

Boolean rules

$$\text{E-}\wedge\ \frac{s \vdash_{\mathsf{E}} \phi \wedge \psi, \Phi; \Sigma}{s \vdash_{\mathsf{E}} \phi, \psi, \Phi; \Sigma} \qquad \text{E-}\vee\ \frac{s \vdash_{\mathsf{E}} \phi \vee \psi, \Phi; \Sigma}{s \vdash_{\mathsf{E}} \phi, \Phi; \Sigma \quad s \vdash_{\mathsf{E}} \psi, \Phi; \Sigma}$$

$$\text{A-}\vee\ \frac{s \vdash_{\mathsf{A}} \phi \vee \psi, \Phi; \Sigma}{s \vdash_{\mathsf{A}} \phi, \psi, \Phi; \Sigma} \qquad \text{A-}\wedge\ \frac{s \vdash_{\mathsf{A}} \phi \wedge \psi, \Phi; \Sigma}{s \vdash_{\mathsf{A}} \phi, \Phi; s \vdash_{\mathsf{A}} \psi, \Phi; \Sigma}$$

if ϕ is not classical or ψ is not classical (no need to break classical formulas apart).

Rules to separate classical sequents

$$\text{E-Split}\ \frac{s \vdash_{\mathsf{E}} \Phi; \Sigma}{s \vdash_{\mathsf{E}} \Gamma; s \vdash_{\mathsf{E}} \Phi \backslash \Gamma; \Sigma} \qquad \text{A-Split}\ \frac{s \vdash_{\mathsf{A}} \Phi; \Sigma}{s \vdash_{\mathsf{A}} \Gamma; \Sigma \qquad s \vdash_{\mathsf{A}} \Phi \backslash \Gamma; \Sigma}$$

if Γ consists of all classical formulas in Φ and Γ is not empty.

Rules to eliminate path quantifiers

$$\text{E-Elim}\ \frac{s \vdash_{\mathsf{E}} Q\phi, \Phi; \Sigma}{s \vdash_Q \phi; s \vdash_{\mathsf{E}} \Phi; \Sigma} \qquad \text{A-Elim}\ \frac{s \vdash_{\mathsf{A}} Q\phi, \Phi; \Sigma}{s \vdash_Q \phi; \Sigma \qquad s \vdash_{\mathsf{A}} \Phi; \Sigma}$$

The above rules apply also if Φ is empty. In this case Φ represents the empty conjunction in $s \vdash_{\mathsf{E}} \Phi$, a sequent that is satisfied by every $\mathcal{I}$, and the empty disjunction in $s \vdash_{\mathsf{A}} \Phi$, a sequent that is satisfied by no $\mathcal{I}$.

When applied exhaustively, the rules so far lead to sequents that all have the form $s \vdash_Q \Phi$ such that (a) Φ consists of classical formulas only, or (b) Φ consists of modal atoms only with top-level operators from $\{\mathsf{U}, \mathsf{R}, \mathsf{X}, \overline{\mathsf{X}}\}$.

Rules to expand U *and* R *formulas*

$$\mathsf{U\text{-}Exp}\ \frac{s \vdash_Q (\phi \mathbin{\mathsf{U}} \psi), \Phi; \Sigma}{s \vdash_Q \psi \lor (\phi \land \mathsf{X}(\phi \mathbin{\mathsf{U}} \psi)), \Phi; \Sigma} \qquad \mathsf{R\text{-}Exp}\ \frac{s \vdash_Q (\phi \mathbin{\mathsf{R}} \psi), \Phi; \Sigma}{s \vdash_Q (\psi \land (\phi \lor \overline{\mathsf{X}}(\phi \mathbin{\mathsf{R}} \psi))), \Phi; \Sigma}$$

The above rules perform one-step expansions of modal atoms with U and R operators.

When applied exhaustively, the rules so far lead to sequents that all have the form $s \vdash_Q \Phi$ such that (a) Φ consists of classical formulas only, or Φ consists of modal atoms only with top-level operators from $\{\mathsf{X}, \overline{\mathsf{X}}\}$.

Rules to simplify X *and* $\overline{\mathsf{X}}$ *formulas.* Below we define inference rules for one-step expansions of sequents of the form $s \vdash_Q \mathsf{X}\,\phi$ and $\vdash_Q \overline{\mathsf{X}}\,\phi$. The following inference rules prepare their application.

$$\mathsf{E\text{-}X\text{-}Simp}\ \frac{s \vdash_{\mathsf{E}} \mathsf{X}\,\phi_1, \ldots, \mathsf{X}\,\phi_n, \overline{\mathsf{X}}\,\psi_1, \ldots, \overline{\mathsf{X}}\,\psi_m; \Sigma}{s \vdash_{\mathsf{E}} Y(\phi_1 \land \cdots \land \phi_n \land \psi_1 \land \cdots \land \psi_m); \Sigma}$$

if $n+m > 1$, where $Y = \overline{\mathsf{X}}$ if $n = 0$ else $Y = \mathsf{X}$. Intuitively, if just one of the modal atoms in the premise is an X-formula then a successor state must exist to satisfy it, hence the X-formula in the conclusion. Similarly:

$$\mathsf{A\text{-}X\text{-}Simp}\ \frac{s \vdash_{\mathsf{A}} \mathsf{X}\,\phi_1, \ldots, \mathsf{X}\,\phi_n, \overline{\mathsf{X}}\,\psi_1, \ldots, \overline{\mathsf{X}}\,\psi_m; \Sigma}{s \vdash_{\mathsf{A}} Y(\phi_1 \lor \cdots \lor \phi_n \lor \psi_1 \lor \cdots \lor \psi_m); \Sigma}$$

if $n + m > 1$, where $Y = \mathsf{X}$ if $m = 0$ else $Y = \overline{\mathsf{X}}$.

To summarize, with the rules so far, all sequents can be brought into one of the following forms: (a) $s \vdash_Q \Gamma$, where Γ consists of classical formulas only, (b) $s \vdash_Q \mathsf{X}\,\phi$, or (c) $s \vdash_Q \overline{\mathsf{X}}\,\phi$.

Rules to expand X *and* $\overline{\mathsf{X}}$ *formulas.*

$$\mathsf{E\text{-}\overline{X}\text{-}Exp}\ \frac{(m, t, l) \vdash_{\mathsf{E}} \overline{\mathsf{X}}\,\phi; \Sigma}{(n_1, u_1[t], l+1) \vdash_{\mathsf{E}} \gamma_1[t] \land \phi; \Sigma \quad \cdots \quad \begin{array}{c}(n_k, u_k[t], l+1) \vdash_{\mathsf{E}} \gamma_k[t] \land \phi; \Sigma \\ (m, t, l) \vdash_{\mathsf{E}} \neg\gamma_1[t] \land \cdots \land \neg\gamma_k[t]; \Sigma\end{array}}$$

if there is a $k \geq 0$ such that $m \xrightarrow{\gamma_i, u_i} n_i$ are all labelled edges emerging from m, where $1 \leq i \leq k$. Notice that the case $k = 0$ is possible. In this case there is only one conclusion, which is equivalent to Σ.

This rule binds the variable *db* in the guards to the term t, which represents the current database. The variable *db* in $\overline{\mathsf{X}}\,\Phi$ refers to the databases in a later state and hence cannot be bound to t.

There is also a rule E-X-Exp whose premise sequent is made with the X operator instead of $\overline{\mathsf{X}}$. It differs from the E-$\overline{\mathsf{X}}$-Exp rule only by leaving away the rightmost conclusion. We do not display it here for space reasons. Dually,

$$\text{A-X-Exp}\ \frac{(m,t,l) \vdash_{\mathsf{A}} \mathsf{X}\,\phi;\Sigma}{(n_1,u_1[t],l+1) \vdash_{\mathsf{A}} \neg\gamma_1[t] \lor \phi; \cdots (n_k,u_k[t],l+1) \vdash_{\mathsf{A}} \neg\gamma_k[t] \lor \phi; (m,t,l) \vdash_{\mathsf{E}} \gamma_1[t] \lor \cdots \lor \gamma_k[t];\Sigma}$$

if there is a $k \geq 0$ such that $m \xrightarrow{\gamma_i,u_i} n_i$ are all labelled edges emerging from m, where $1 \leq i \leq k$.

The conclusion sequent $(m,t,l) \vdash_{\mathsf{E}} \gamma_1[t] \lor \cdots \lor \gamma_k[t]$ forces that at least one guard is true. Analogously to above, there is also a rule A-$\overline{\mathsf{X}}$-Exp for the $\overline{\mathsf{X}}$ case, which does not include this sequent. This reflects that $\overline{\mathsf{X}}$ formulas are true in states without successor.

These rules are the only one that increase the length counter l.

Rule to close branches

$$\text{Close}\ \frac{(m_1,t_1,l_1) \vdash_{Q_1} \Phi_1; \cdots ; (m_n,t_n,l_n) \vdash_{Q_n} \Phi_n}{\text{FAIL}}$$

if every formula in every Φ_i is classical and $F = \bigwedge_{i=1,\ldots,n} \mathrm{form}_{Q_i}(\Phi_i[t_i])$ is unsatisfiable (not satisfied by any interpretation $\mathcal{I}$).

Notice that F is a classical formula, It is meant to be passed to a first-order theorem prover for checking unsatisfiability.

Let us now turn to analyzing the calculus' theoretical properties. To this end, we equip sequents with a formal semantics within the temporal logic framework in Section 2. In that framework, a state is a mapping from variables to domain elements. In our case the (relevant) variables are fixed, which are the Node-sorted variable ℓ and the DB-sorted variable db. Given an interpretation $\mathcal{I}$, we associate to the triple (n,t,l) the state $\mathrm{state}_{\mathcal{I}}(n,t,l) \overset{\text{def}}{=} \{\ell \mapsto n,\ db \mapsto I(t)\}$ (the length l has no relevant meaning for that).

Definition 4.1 (Tableau node semantics). *Let $\mathcal{I}$ be an interpretation. We say that $\mathcal{I}$* satisfies *a sequent $s \vdash_Q \Phi$, written as $\mathcal{I} \models s \vdash_Q \Phi$, iff $(\mathcal{I}, \mathrm{state}_{\mathcal{I}}(s)) \models \mathrm{form}_Q(\Phi)$. We say that $\mathcal{I}$* satisfies *a set Σ of sequents, written as $\mathcal{I} \models \Sigma$, iff $\mathcal{I}$ satisfies every sequent in Σ.*

The following lemma expresses the correctness of our inference rules.[3]

Lemma 4.2. *Let $\mathcal{I}$ be an interpretation and Σ a set of sequents. For every tableau rule inference with premise Σ and conclusions $\Sigma_1,\ldots,\Sigma_n$ it holds that $\mathcal{I} \models \Sigma$ if and only if $\mathcal{I} \models \Sigma_j$, for some $1 \leq j \leq n$.*

Theorem 4.3 (Soundness). *Given a state transition system $M = (S, I, R)$ as described in Section 3 and a path formula $\Psi_0[db]$. If there is a refutation of Ψ_0 then for no interpretation $\mathcal{I}$ and no $s_0 \in I$ it holds $(\mathcal{I}, s_0) \models \mathsf{E}\,\Psi_0$.*

We are now turning to completeness. In its simplest form, the completeness statement reads as "if for no interpretation $\mathcal{I}$ and no $s_0 \in I$ it holds $(\mathcal{I}, s_0) \models \mathsf{E}\,\Psi_0$ then there is a refutation". For efficiency in practice, one should exploit confluence properties of the

[3] Proofs are in the long version of this paper, see `http://www.nicta.com.au/pub?id=6988`

inference rules and work with fair derivations instead. To this end, we demand that the inference rules are applied in the order given above, with decreasing priority. (In the bounded setting described below this is indeed a fair strategy.) Additionally, we would like to get a stronger model-completeness result saying that a non-refutation leads not only to a model of Ψ_0 but also delivers the corresponding run.

However, the infinite-state model checking problems we are dealing with make any general completeness result impossible. Our pragmatic solution is to use a form of bounded model checking by limiting runs to user-given length, as follows.

Let $l_{\max} \geq 0$ be an integer, the *length bound*. We define *bounded versions* of the rules to expand X and $\overline{\mathsf{X}}$ formulas by taking $k = 0$ whenever $l = l_{\max}$, otherwise the rule is applied as stated. That is, after $l_{\max}$ expansions of X or $\overline{\mathsf{X}}$ formulas the bounded versions of the inference rules pretend that the underlying run (of length $l_{\max}$) has stopped. The *bounded version* of the tableau calculus uses that bounded rules.

We need to reflect the bounded version of the calculus at the semantics level. Given a state transition system $M = (S, I, R)$ and $l_{\max} \geq 0$, let the admissible runs of M consist of all runs r from each $s_0 \in I$ such that $|r| \leq l_{\max}$ and if $|r| < l_{\max}$ then r is finished. We qualify the resulting satisfaction relation of Section 2 by "wrt. runs of length $l_{\max}$".

Theorem 4.4 (Bounded tableau calculus completeness). *Given a state transition system $M = (S, I, R)$ as described in Section 3, $l_{\max} \geq 0$ a length bound, and a path formula $\Psi_0[db]$. Let $\mathcal{D}$ be a fair derivation from Ψ_0 in the bounded version of the calculus.*

Then, D is finite, every non-FAIL leaf Σ consists of classical sequents only, and for every model $\mathcal{I}$ of Σ it holds $(\mathcal{I}, s_0) \models \mathsf{E}\,\Psi_0$ wrt. runs of length $l_{\max}$.

Conversely, for every interpretation $\mathcal{I}$ such that $(\mathcal{I}, s_0) \models \mathsf{E}\,\Psi_0$ wrt. runs of length $l_{\max}$ there is a non-FAIL leaf Σ such that $\mathcal{I}$ satisfies Σ (model completeness).

Thanks to the tableau calculus maintaining the history of expanding formulas, it is easy to extract from the branches leading to the leaves Σ the corresponding runs. Moreover, the formula Σ represents the weakest condition on $\mathcal{I}$ and this way provides more valuable feedback than, say, a fully specified concrete database.

But notice that in order to exploit Theorem 4.4 in practice, one has to establish satisfiability of the non-FAIL leaf node Σ. In general this is impossible, and the first-order proof problems we are dealing with are highly undecidable (Π^1_1-complete [19]), as the `DEFINITIONS` section may contain arbitrary FO sentences over integer arithmetics with free function symbols [11].

5 Inductive Proofs of Safety Properties

Verifying a safety property $\mathsf{A}\,\mathsf{G}\,\phi$ of a state transition system M (under given *constraints*) is especially problematic when using bounded model checking. The complexity of model checking will in most cases be prohibitive in case ϕ is in fact invariant and the failure to find a counterexample trace of a given length does not entail invariance.

A relatively simple method for verifying safety properties that has been shown to often work well in practice in the context of SAT and SMT based model checking is the *k-induction principle* [22,15,13]. It attempts to prove an invariant ϕ by iteratively increasing a parameter $k \geq 1$, the maximal length of considered traces, until a counterexample trace for the base case is found, k-induction succeeds, or some predetermined bound for k is reached. In our setting the the k-induction principle reads as follows:

Base Case: There does *not* exist a Σ interpretation $\mathcal{I}$ and an assignment α_0 with $\alpha_0(l) = n_0$ such that $(\mathcal{I}, \alpha_0) \models \mathsf{E}((constraints \land defs) \land \neg(\phi \land \overline{\mathsf{X}}\phi \land ... \land \overline{\mathsf{X}}^{k-1}\phi))$.

Induction Step: There does *not* exist a Σ interpretation $\mathcal{I}$ and an assignment α_0 with $\alpha_0(l) = n_i$ for some $n_i \in N$ such that $(\mathcal{I}, \alpha_0) \models \mathsf{E}\ (defs \land \neg((\phi \land \mathsf{X}\phi \land ... \land \mathsf{X}^{k-1}\phi) \rightarrow \mathsf{X}^{k-1}\overline{\mathsf{X}}\phi))$.

Constraints (e.g. of the form $\mathsf{A}\,\mathsf{G}\,\psi$) can be used for traces starting at the initial states, but in general not for the inductive step. An upper bound for k can sometimes be computed from the structure of ϕ and *constraints*. In general, k-induction based model checking is incomplete because a counterexample trace to the inductive step for some k may start at a state which is unreachable from an intitial state. While the objective of increasing k is precisely to avoid such "spurious" counterexamples, some properties are not k-inductive for any k. Strengthening the property to be verified [15,13] is one means of attempting to avoid that problem. We have adapted the strengthening strategy presented in [15] to our framework, though more work is required to make this approach practical.

6 Implementation and Experiments

We have implemented the modelling language of Section 3, the tableau calculus of Section 4, and the k-induction scheme of Section 5 on top of it.[4] The implementation, in Scala, is in a prototypical stage and is intended as a testbed for rapidly trying out ideas. As the first-order logic theorem prover for the Close rule we coupled Microsoft's SMT-solver Z3 [14]. Z3 accepts quantified formulas, which are treated by instantiation heuristics. Moreover, Z3 natively supports integers, arrays, and lists. For JSON record types we have to supply axioms explicitly. Non-recursive definitions are passed on as "functions" to Z3, recursive ones as "constraints". The coupling of Z3 is currently rather inefficient, through a file interface using the SMT2 language.

The lack of further improvements currently limits our implementation to problems that do not require too much combinatorial search induced by a process' dynamics. But, in fact, we are currently mostly interested in investigating the usefulness and limits of currently available first-order theorem proving technology in an expressive verification framework as ours (recall from Theorems 4.3 and 4.4 and the accompanying discussions how critically our approach depends on that).

A basic query is $\mathsf{F}\ completed(db)$, which checks whether or not it is possible to fully execute an (acceptable) order into a completed state. Such "planning" queries are useful, *e.g.*, for flexible process configuration from fragments during runtime, but also for static analysis during design time. Our prover can be instructed to exhaust all branches under

[4] Our implementation supports concrete reasoning problems (Section 3) by evaluation of `scripts` with a Groovy interpreter and a tailored, model elimination based proof procedure for guards, but we do not discuss this here.

inference rule applications and extract all runs represented by non-FAIL leaves. With a length bound $l_{\max} = 8$ it returns the runs

$$\begin{array}{l}
\mathit{Init} \rightarrow \mathit{Pack} \rightarrow \mathit{Stocktake} \rightarrow \mathit{Pack} \rightarrow \mathit{Invoice} \rightarrow \mathit{Shipped} \rightarrow \mathit{Paid} \\
\mathit{Init} \rightarrow \mathit{Pack} \rightarrow \mathit{Stocktake} \rightarrow \mathit{Pack} \rightarrow \mathit{Stocktake} \rightarrow \mathit{Pack} \rightarrow \mathit{Invoice} \rightarrow \mathit{Shipped} \rightarrow \mathit{Paid} \\
\mathit{Init} \rightarrow \mathit{Pack} \rightarrow \mathit{Stocktake} \rightarrow \mathit{Pack} \rightarrow \mathit{Invoice} \rightarrow \mathit{Paid} \rightarrow \mathit{Shipped} \\
\mathit{Init} \rightarrow \mathit{Pack} \rightarrow \mathit{Stocktake} \rightarrow \mathit{Pack} \rightarrow \mathit{Stocktake} \rightarrow \mathit{Pack} \rightarrow \mathit{Invoice} \rightarrow \mathit{Paid} \rightarrow \mathit{Shipped}
\end{array}$$

which are exactly the expected ones. In total, 223 branches have been closed, with 912 inference rule applications, and Z3 was called 529. The total runtime is 30 seconds, the time spent in Z3 was negligible. A variation is the query $(\mathsf{F}\, \mathit{completed}(\mathit{db})) \land (\mathit{db}.\mathtt{shipped} = \mathit{true}\ \mathsf{R}\ \mathit{db}.\mathtt{paid} = \mathit{false})$ ("Is there a completed state that has shipment before payment?") which returns the first two of the above runs. We also experimented with unsatisfiable variants, e.g., by adding the `CONSTRAINT` $\mathit{db}.\mathtt{gold} = \mathit{false}$ to the latter query. All these queries can be answered in comparable or shorter time.

Let us now turn to safety properties. They typically occur during design time, and are clearly general (as opposed to concrete) problems. Here are some examples, stated in non-negated form:

$\mathsf{A}\,\mathsf{G}\,(\forall i{:}\mathtt{Integer}.((0 \le i \land i < \mathit{db}.\mathtt{nrStockItems}) \Rightarrow \mathit{db}.\mathtt{stock}[i].\mathtt{available} \ge 0))$
("The number of available stock items is non-negative")

$\mathsf{A}\,\mathsf{G}\,((\mathit{db}.\mathtt{paid} = \mathit{true} \land \mathit{db}.\mathtt{shipped} = \mathit{false}) \Rightarrow \mathsf{F}\,\mathit{db}.\mathtt{shipped} = \mathit{true})$
("Orders that have been paid for but not yet shipped will be shipped eventually")

$\mathsf{A}\,\mathsf{G}\,((\mathit{db}.\mathtt{gold} = \mathit{false} \land \mathit{db}.\mathtt{shipped} = \mathit{true}) \Rightarrow \mathit{db}.\mathtt{paid} = \mathit{true})$
(Follows from non-gold `CONSTRAINT`)

$\mathsf{A}\,\mathsf{G}\,\mathtt{inRange}(\mathit{db}.\mathtt{open}, \mathit{db}.\mathtt{nrStockItems})$
where `inRange` is defined as $\forall l{:}\mathtt{List[Integer]}.\forall n{:}\mathtt{Integer}.(\mathtt{inRange}(l, n) \Leftrightarrow$
$(l = \mathtt{[|\ \ |]} \lor (0 \le \mathtt{head}(l) \land \mathtt{head}(l) < n \land \mathtt{inRange}(\mathtt{tail}(l), n))))$
("The open list contains valid item numbers only, in the range $0 \ldots \mathit{db}.\mathtt{nrStockItems}$")

The first property requires the additional `CONSTRAINT` $\mathit{db}.\mathtt{nrStockItems} \ge 0 \land (\forall i{:}\mathtt{Integer}.((0 \le i \land i < \mathit{db}.\mathtt{nrStockItems}) \Rightarrow \mathit{db}.\mathtt{stock}[i].\mathtt{available} \ge 0))$, which asserts that that property holds true initially. The proof of that is found with $k = 1$. The second property needs $k = 3$, both proven within seconds.

The third property is problematic. Although valid, it cannot be proven by k-induction because it admits spurious counterexamples due to ignoring *constraints* in the induction step. The fourth property is again valid after adding $\mathit{db}.\mathtt{nrStockItems} \ge 0 \land \mathtt{inRange}(\mathit{db}.\mathtt{open}, \mathit{db}.\mathtt{nrStockItems})$ to `CONSTRAINTS`, which asserts the property holds initially. It is provable by k-induction for $k = 2$. There is a caveat, though: as said, our prover tries $k = 1, 2, \ldots$ in search for a proof by k-induction. Here, for $k = 1$ an unprovable (satisfiable) proof obligation in the induction step turned up on which Z3 did not terminate. Z3, like other SMT-solvers, does not reliable detect countersatisfiability for non-quantifier free problems. This is a general problem and can be expected to show up as soon as datatypes with certain properties (like `inRange`) are present. Our workaround for now is to use time limits and pretend countersatisfiability in case of inconclusive results. Notice that this preserves the soundness of our k-induction procedure.

7 Conclusions and Future Work

In this paper we proposed an expressive modelling framework based on first-order logic over background theories (arithmetics, lists, records, etc) and state transition systems over corresponding interpretations. The framework is meant to smoothly support a wide range of practical applications, in particular those that require rich data structures and declarative process modelling by fragments and constraints governing their composition. On the reasoning side, we introduced a tableau calculus for bounded model checking of properties expressed in a certain fragment of CTL* over that first-order logic. To our knowledge, the tableau calculus as such and our soundness and completeness results are novel. First experiments with our implementation suggests that bounded model checking is already quite useful in the business domain we considered, in particular in combination with k-induction.

From another point of view, this paper is meant as an initial exploration into using general first-order logic theorem provers as back-ends for dynamic system verification. Developing such systems that natively support quantified formulas over built-in theories has been become an active area of research. Improvements here directly carry over to a stronger system on our side. For instance, we plan to integrate the prover described in [3].

We also plan to work on some conceptual improvements. Among them are blocking mechanisms to detect recurring nodes, partial-order reduction to break symmetries among fragment compositions, and cone of influence reduction. Each of these reduces, ultimately, to first-order logic proof problems, which again emphasizes the role of first-order logic theorem proving in our context.

References

1. Baader, F., Liu, H., ul Mehdi, A.: Verifying properties of infinite sequences of description logic actions. In: ECAI, pp. 53–58 (2010)
2. Bauer, A., Leucker, M., Schallhart, C.: Comparing LTL semantics for runtime verification. Logic and Computation 20(3), 651–674 (2010)
3. Baumgartner, P., Waldmann, U.: Hierarchic superposition with weak abstraction. In: Bonacina, M.P. (ed.) CADE 2013. LNCS (LNAI), vol. 7898, pp. 39–57. Springer, Heidelberg (2013)
4. Bersani, M.M., Cavallaro, L., Frigeri, A., Pradella, M., Rossi, M.: SMT-based verification of LTL specification with integer constraints and its application to runtime checking of service substitutability. In: Fiadeiro, J.L., Gnesi, S., Maggiolo-Schettini, A. (eds.) SEFM, pp. 244–254. IEEE Computer Society (2010)
5. Chang, L., Shi, Z., Gu, T., Zhao, L.: A family of dynamic description logics for representing and reasoning about actions. J. Autom. Reasoning 49(1), 1–52 (2012)
6. Clarke, E.M., Grumberg, O., Peled, D.A.: Model Checking. The MIT Press, Cambridge (1999)
7. Crockford, D.: RFC 4627—The application/json media type for JavaScript Object Notation (JSON). Technical report, IETF (2006)
8. Damaggio, E., Deutsch, A., Hull, R., Vianu, V.: Automatic verification of data-centric business processes. In: Rinderle-Ma, S., Toumani, F., Wolf, K. (eds.) BPM 2011. LNCS, vol. 6896, pp. 3–16. Springer, Heidelberg (2011)

9. Ghilardi, S., Nicolini, E., Ranise, S., Zucchelli, D.: Combination methods for satisfiability and model-checking of infinite-state systems. In: Pfenning, F. (ed.) CADE 2007. LNCS (LNAI), vol. 4603, pp. 362–378. Springer, Heidelberg (2007)
10. Goré, R.: Tableau methods for modal and temporal logics. In: D'Agostino, M., Gabbay, D., Hähnle, R., Posegga, J. (eds.) Handbook of Tableau Methods, ch. 6, pp. 297–396. Kluwer Academic Publishers (1999)
11. Halpern, J.: Presburger Arithmetic With Unary Predicates is Π_1^1-Complete. Journal of Symbolic Logic 56(2), 637–642 (1991)
12. Hariri, B.B., Calvanese, D., Giacomo, G.D., Masellis, R.D., Felli, P., Montali, M.: Verification of description logic knowledge and action bases. In: Raedt, L.D., Bessière, C., Dubois, D., Doherty, P., Frasconi, P., Heintz, F., Lucas, P.J.F. (eds.) ECAI. Frontiers in Artificial Intelligence and Applications, vol. 242, pp. 103–108. IOS Press (2012)
13. Kahsai, T., Tinelli, C.: Pkind: A parallel k-induction based model checker. In: Barnat, J., Heljanko, K. (eds.) PDMC. EPTCS, vol. 72, pp. 55–62 (2011)
14. de Moura, L., Bjørner, N.: Z3: An efficient SMT solver. In: Ramakrishnan, C.R., Rehof, J. (eds.) TACAS 2008. LNCS, vol. 4963, pp. 337–340. Springer, Heidelberg (2008)
15. de Moura, L., Rueß, H., Sorea, M.: Bounded model checking and induction: From refutation to verification. In: Hunt Jr., W.A., Somenzi, F. (eds.) CAV 2003. LNCS, vol. 2725, pp. 14–26. Springer, Heidelberg (2003)
16. Nigam, A., Caswell, N.S.: Business artifacts: An approach to operational specification. IBM Systems Journal 42(3), 428–445 (2003)
17. Pesic, M., van der Aalst, W.M.P.: A declarative approach for flexible business processes management. In: Eder, J., Dustdar, S. (eds.) BPM 2006 Workshops. LNCS, vol. 4103, pp. 169–180. Springer, Heidelberg (2006)
18. Reynolds, M.: A tableau for CTL*. In: Cavalcanti, A., Dams, D. (eds.) FM 2009. LNCS, vol. 5850, pp. 403–418. Springer, Heidelberg (2009)
19. Rogers Jr., H.: Theory of Recursive Functions and Effective Computability. The MIT Press, Cambridge (1987)
20. Schuele, T., Schneider, K.: Global vs. local model checking: A comparison of verification techniques for infinite state systems. In: SEFM, pp. 67–76. IEEE Computer Society, Washington, Dc (2004)
21. Sutcliffe, G., Schulz, S., Claessen, K., Baumgartner, P.: The TPTP typed first-order form with arithmetic. In: Bjørner, N., Voronkov, A. (eds.) LPAR-18. LNCS, vol. 7180, pp. 406–419. Springer, Heidelberg (2012)
22. Sheeran, M., Singh, S., Stålmarck, G.: Checking safety properties using induction and a SAT-solver. In: Hunt Jr., W.A., Johnson, S.D. (eds.) FMCAD 2000. LNCS, vol. 1954, pp. 108–125. Springer, Heidelberg (2000)
23. Vianu, V.: Automatic verification of database-driven systems: a new frontier. In: Fagin, R. (ed.) ICDT. ACM International Conference Proceeding Series, vol. 361, pp. 1–13. ACM (2009)

Bounded Proofs and Step Frames

Nick Bezhanishvili[1,⋆], Silvio Ghilardi[2],

[1] Utrecht University, Utrecht, The Netherlands
[2] Università degli Studi di Milano, Milano, Italy

Abstract. The longstanding research line investigating free algebra constructions in modal logic from an algebraic and coalgebraic point of view recently lead to the notion of a one-step frame [14], [8]. A one-step frame is a two-sorted structure which admits interpretations of modal formulae without nested modal operators. In this paper, we exploit the potential of one-step frames for investigating proof-theoretic aspects. This includes developing a method which detects when a specific rule-based calculus Ax axiomatizing a given logic L has the so-called bounded proof property. This property is a kind of an analytic subformula property limiting the proof search space. We define conservative one-step frames and prove that every finite conservative one-step frame for Ax is a p-morphic image of a finite Kripke frame for L iff Ax has the bounded proof property and L has the finite model property. This result, combined with a 'one-step version' of the classical correspondence theory, turns out to be quite powerful in applications. For simple logics such as **K**, **T**, **K4**, **S4**, etc, establishing basic metatheoretical properties becomes a completely automatic task (the related proof obligations can be instantaneously discharged by current first-order provers). For more complicated logics, some ingenuity is needed, however we successfully applied our uniform method to Avron's cut-free system for **GL** and to Goré's cut-free system for **S4.3**.

1 Introduction

The method of describing free algebras of modal logics by approximating them with finite partial algebras is longstanding. The key points of this method are that every free algebra is approximated by partial algebras of formulas of modal complexity n, for $n \in \omega$, and that dual spaces of these approximants can be described explicitly [1], [16]. The basic idea of this construction can be traced back to [15]. In recent years there has been a renewed interest in this method e.g., [6], [8], [9], [14], [17]. In this paper we apply the ideas originating from this line of research to investigate proof-theoretic aspects of modal logics. In particular, we will concentrate on the *bounded proof property.* An axiomatic system Ax has the bounded proof property (the bpp, for short) if every formula ϕ of modal complexity at most n derived in Ax from some set Γ containing only formulae of modal complexity at most n, can be derived from Γ in Ax by only

⋆ Supported by the Dutch NWO grant 639.032.918 and the Rustaveli Science Foundation of Georgia grant FR/489/5-105/11.

using formulae of modal complexity at most n. The bounded proof property is a kind of an analytic subformula property limiting the proof search space. This property holds for proof systems enjoying the subformula property (the latter is a property that usually follows from cut elimination). The bounded proof property depends on an axiomatization of a logical system. That is, one axiomatization of a logic may have the bpp and the other not. Examples of such axiomatizations will be given in Section 5 of the paper.

The main tools of our method are the one-step frames introduced in [14] and [8]. A one-step frame is a two-sorted structure which admits interpretations of modal formulae without nested modal operators. We show that an axiomatic system Ax axiomatizing a logic L has the bpp and the finite model property (the fmp) iff every one step-frame validating Ax is a p-morphic image of a finite standard (aka Kripke) frame for L. This gives a purely semantic characterization of the bpp. The main advantage of this criterion is that it is relatively easy to verify. In the next subsection we give an example explaining the details of our machinery step-by-step. Here we just list the main ingredients. Given an axiom of a modal logic, we rewrite it into a one-step rule, that is, a rule of modal complexity 1. One-step rules can be interpreted on one-step frames. We use an analogue of the classical correspondence theory, to obtain a first-order condition (or a condition of first-order logic enriched with fixed-point operators) for a one-step frame corresponding to the one-step rule. Finally, we need to find a standard frame p-morphically mapped onto any finite one-step frame satisfying this first-order condition. This part is not automatic, but we have some standard templates. For example, we define a procedure modifying the relation of a one-step frame so that the obtained frame is standard. In easy cases, e.g., for modal logics such as **K**, **T**, **K4**, **S4**, this frame is a frame of the logic and is p-morphically mapped onto the one-step frame. The bpp and fmp for these logics follow by our criterion. For more complicated systems such as **S4.3** and **GL**, we show using our method that Avron's cut-free system for **GL** [2] and Goré's cut-free system for **S4.3** [19] provide axiomatic systems with the bpp.

A Worked Out Example. In order to explain the basic idea of our technique, we proceed by giving a rather simple (but still significant) example. Consider the modal logic obtained by adding to the basic normal modal system **K** the 'density' axiom:

$$\Box\Box x \to \Box x. \tag{1}$$

First Step: we replace (1) by equivalent derived rules having modal complexity 1. The obvious solution is to replace the modalized subformulae occurring inside the modal operator by an extra propositional variable. Thus the first candidate is the rule $y \leftrightarrow \Box x / \Box y \to \Box x$. A better solution (suggested by the proof of Proposition 1) is to take advantage of the monotonicity and to use instead the rule

$$\frac{y \to \Box x}{\Box y \to \Box x} \tag{2}$$

Often, the method suggested by the proof of Proposition 1 gives 'good' rules, but for more complicated logics one needs some ingenuity to find the right system of derived rules replacing the axioms (this is substantially the kind of ingenuity needed to find rules leading to cut eliminating systems).

Second Step: this step may or may not succeed, but it is entirely algorithmic. It relies on a light modification of the well-known modal correspondence machinery. We first observe that inference rules having modal complexity 1 can be interpreted in the so-called *one-step frames.* A one-step frame is a quadruple $\mathcal{S} = (W_1, W_0, f, R)$, where W_0, W_1 are sets, $f : W_1 \to W_0$ is a map and $R \subseteq W_1 \times W_0$ is a relation between W_1 and W_0. In the applications, we need two further requirements (called *conservativity* requirements) on such a one-step frame $\mathcal{S}$: for the purpose of the present discussion, we may ignore the second requirement and keep only the first one, which is just the surjectivity of f. Formulae of modal complexity 1 (i.e., without nested modal operators) can be interpreted in one-step frames as follows: propositional variables are interpreted as subsets of W_0; when we apply modal operators to subsets of W_0, we produce subsets of W_1 using the modal operator $\Box_R$ canonically induced by R. In particular, for $y \subseteq W_0$ the operator $\Box_R$ is defined as $\Box_R y = \{w \in W_1 \mid R(w) \subseteq y\}$, where $R(w) = \{v \in W_0 \mid (w,v) \in R\}$. Whenever we need to compare, say y and $\Box_R x$, we apply the inverse image f (denoted by f^*) to y in order to obtain a subset of W_1. Thus, a one-step frame $\mathcal{S} = (W_1, W_0, f, R)$ validates (2) iff we have

$$\forall x, y \subseteq W_0 \ (f^*(y) \subseteq \Box_R x \Rightarrow \Box_R y \subseteq \Box_R x).$$

The standard correspondence machinery for Sahlqvist formulae shows that in the two-sorted language of one-step frames this condition has the following first-order equivalent:

$$\forall w \forall v \ (wRv \Rightarrow \exists k \ (wRf(k) \ \& \ kRv)). \tag{3}$$

In relational composition notation this becomes $R \subseteq R \circ f^o \circ R$, where f^o is the binary relation such that $wf^o v$ iff $f(w) = v$. We may call (3) the *step-density* condition. In fact, notice that for standard frames, where we have $W_1 = W_0$ and $f = id$, step-density condition becomes the customary density condition, see (6) below.

Third Step: our main result states that both the finite model property and bounded proof property (for the global consequence relation) are guaranteed provided we are able to show that *any finite conservative one-step frame validating our inference rules is a p-morphic image of a standard finite frame for our original logic.* The formal definition of a p-morphic image for one-step frames will be given in Definition 5. Here we content ourselves to observing that, in our case, in order to apply the above result and obtain the fmp and bpp, we need to prove that, given a conservative finite step-dense frame $\mathcal{S} = (W_1, W_0, f, R)$, there are a finite dense frame $\mathfrak{F} = (V, S)$ and a surjective map $\mu : V \longrightarrow W_1$ such that $R \circ \mu = f \circ \mu \circ S$. In concrete examples, the idea is to take $V := W_1$ and $\mu := id_{W_1}$. So the whole task reduces to that of finding $S \subseteq W_1 \times W_1$ such that $R = f \circ S$. That is, S should satisfy

$$\forall w \forall v \ (wRv \ \Leftrightarrow \ \exists w' \ (wSw' \ \& \ f(w') = v)). \tag{4}$$

Some ingenuity is needed in the general case to find the appropriate S (indeed our problem looks quite similar to the problem of finding appropriate filtrations case-by-case). As in the case of filtrations, there are standard templates that often work for the cases of arbitrary relations, transitive relations, etc. The basic template for the case of an arbitrary relation is that of taking S to be $f^o \circ R$, namely

$$\forall w \forall w' \ (wSw' \Leftrightarrow \exists v \ (wRv \ \& \ f(w') = v)). \tag{5}$$

Notice that what we need to prove in the end is that, assuming (3), the so-defined S satisfies (4) and

$$\forall w \forall v \ (wSv \Rightarrow \exists k \ (kSv \ \& \ wSk)). \tag{6}$$

Thus, taking into consideration that f is also surjective, i.e.,

$$\forall v \ \exists w \ f(w) = v, \tag{7}$$

(because $\mathcal{S}$ is conservative), we need the validity of the implication

$$(7) \ \& \ (3) \ \& \ (5) \ \Rightarrow \ (4) \ \& \ (6).$$

The latter is a deduction problem in first-order logic that can be solved affirmatively along the lines indicated in Section 5. The problem can be efficiently discharged by provers like SPASS, E, Vampire.[1]

In summary, the above is a purely algorithmic procedure, that may or may not succeed (in case it does not succeed, one may try to invent better solutions for the derived rules of Step 1 and/or for defining the relation S in Step 3). In case the procedure succeeds, we really obtain quite a lot of information about our logic, because we get altogether: (i) completeness via the finite model property; (ii) decidability; (iii) the bounded proof property; (iv) first-order definability; (v) canonicity (as a consequence of (i)+(iv), via known results in modal logic). Further applications concern the step-by-step descriptions of finitely generated free algebras (following the lines of [14] and [8]). But we will not deal with free algebras in this paper.

The large amount of information that one can obtain from successful runs of the method might suggest that the event of success is quite rare. This is true in essence, but we shall see in the paper that (besides simple systems such as $\mathbf{K}, \mathbf{T}, \mathbf{K4}, \mathbf{S4}$) the procedure can be successfully applied to more interesting case studies such as the linear system **S4.3** and the Gödel-Löb system **GL**. In the latter case we have definability not in first-order logic, but in first-order logic enriched with fixed-point operators. However, for finite one-step frames (as for finite standard **GL**-frames), this condition boils down to a first-order condition.

The paper is organized as follows. In Section 2 we recall the basic definitions of logics and decision problems. In Section 3 we introduce one-step frames and state our main results. In Section 4 we discuss the correspondence theory for one-step frames. In Section 5 we supply illustrative examples and case studies. Section 6

[1] SPASS `http://www.spass-prover.org/` (in the default configuration) took less than half a second to solve the above problem with a 47-line proof.

provides concluding remarks and discusses future work. For space reasons, we can only limit ourselves to giving the definitions, main results and important examples. For all the other details (especially the proofs), the reader is referred to the accompanying online Technical Report [7].

2 Logics and Decision Problems

Modal formulae are built from propositional variables $x, y, \dots$ by using the Booleans $(\neg, \wedge, \vee, 0, 1)$ and a modal operator $\Diamond$ (further connectives such as $\rightarrow, \Box$ are defined in the standard way). Underlined letters stand for tuples of unspecified length formed by distinct elements. Thus, we may use $\underline{x}$ for a tuple $x_1, \dots, x_n$. When we write $\phi(\underline{x})$ we want to stress that ϕ contains at most the variables $\underline{x}$. The same convention applies to sets of formulae: if Γ is a set of formulae and we write $\Gamma(\underline{x})$, we mean that all formulae in Γ are of the kind $\phi(\underline{x})$. The modal complexity of a formula ϕ counts the maximum number of nested modal operators in ϕ (the precise definition is by an obvious induction). The polarity (positive/negative) of an occurrence of a subformula in a formula ϕ is defined inductively: ϕ is positive in ϕ, the polarity is preserved through all connectives, except $\neg$ that reverses it. When we say that a propositional variable is positive (negative) in ϕ we mean that all its occurrences are such.

A *logic* is a set of modal formulae containing tautologies, Aristotle's principle (namely $\Box(x \rightarrow y) \rightarrow (\Box x \rightarrow \Box y)$) and closed under uniform substitution, modus ponens and necessitation (namely $\phi/\Box\phi$).

We are interested in the *global consequence relation decision problem* for modal logics [20, Ch. 3.1]. This can be formulated as follows: given a logic L, a finite set $\Gamma = \{\phi_1, \dots, \phi_n\}$ of formulae and a formula ψ, decide whether $\Gamma \vdash_L \psi$. Here the notation $\Gamma \vdash_L \psi$ means that there is a proof of ψ using tautologies, Aristotle's principle and the formulae in Γ, as well as necessitation, modus ponens and substitution instances of formulae from a set of axioms for L (notice that uniform substitutions cannot be applied to formulae in Γ).

In proof theory, logics are specified via axiomatic systems consisting of inference rules (axioms are viewed as 0-premises rules). Formally, an *inference rule* is an $n+1$-tuple of formulae, written in the form

$$\frac{\phi_1(\underline{x}), \dots, \phi_n(\underline{x})}{\psi(\underline{x}).} \tag{8}$$

An *axiomatic system* Ax is a set of inference rules. We write $\vdash_{\mathrm{Ax}} \phi$ to mean that ϕ has a proof using tautologies and Aristotle's principle as well as modus ponens, necessitation and inferences from Ax. When we say that a proof uses an inference rule such as (8), we mean that the proof can introduce at any step i a formula of the kind $\psi\sigma$ provided it already introduced in the previous steps $j_1, \dots, j_n < i$ the formulae $\phi_1\sigma, \dots, \phi_n\sigma$, respectively. Here σ is a substitution and notation $\psi\sigma$ denotes the application of the substitution σ to ψ.

Given a finite set $\Gamma = \{\phi_1, \dots, \phi_n\}$, an inference system Ax and a formula ψ, we write $\Gamma \vdash_{\mathrm{Ax}} \psi$ to mean that ψ has a proof using tautologies, Aristotle's principle and elements from Γ as well as modus ponens, necessitation and inferences from Ax (again notice that uniform substitution cannot be applied to members of Γ). We need some care when replacing a logic L with an inference system Ax, because we want global consequence relation to be preserved, in the sense of Proposition 1(ii) below. To this aim, we need to use derivable rules: the rule (8) is *derivable* in a logic L iff $\{\phi_1, \ldots, \phi_n\} \vdash_L \psi$. We say that the inference rule (8) is *reduced* iff (i) the formulae $\phi_1, \ldots, \phi_n, \psi$ have modal complexity at most 1; (ii) every propositional variable occuring in (8) occurs within a modal operator[2] An axiomatic system is *reduced* iff all inference rules in it are reduced.

Definition 1. *An axiomatic system* Ax *is* adequate for a logic L (*or* Ax *is an axiomatic system* for L) *iff* (*i*) *it is reduced;* (*ii*) *all rules in* Ax *are derivable in* L*;* (*iii*) $\vdash_{\mathrm{Ax}} \phi$ *for all* $\phi \in L$.

Proposition 1. (*i*) *For any modal logic L, there always exists an axiomatic system* Ax*, which is adequate for L.*
(*ii*) *If* Ax *is an axiomatic system for L, then $\Gamma \vdash_{\mathrm{Ax}} \psi$ iff $\Gamma \vdash_L \psi$ for all Γ, ψ.*

Proof. We just indicate how to prove (i) by sketching an algorithm replacing every rule (8) by one or more reduced rules. Applying exhaustively this algorithm to the formulae in L (or just to a set of axioms for L) viewed as zero-premises rules, we obtain the desired axiomatic system for L. Notice that the algorithm has a large degree of non determinism, so proof-theoretic properties of the outcome may be influenced by the way we run it.

Take a formula α having modal complexity at least one and take an occurrence of it located inside a modal operator in (8). We can obtain an equivalent rule by replacing this occurrence by a new propositional variable y and by adding as a further premise $\alpha \to y$ (resp. $y \to \alpha$) if the occurrence of α is positive within ψ or negative within one of the ϕ_i's (resp. if the occurrence of α is negative within ψ or positive within one of the ϕ_i's). Continuing in this way, in the end, only formulae of modal complexity at most 1 will occur in the rule. If a variable x does does not occur inside a modal operator in (8), one can add $\Box(x \vee \neg x)$ as a further premise (alternatively, one can show that x is eliminable from (8)). $\dashv$

Thus by Proposition 1(ii), the global consequence relation $\Gamma \vdash_{\mathrm{Ax}} \phi$ does not depend on an axiomatic system Ax chosen for a given logic L. However, deciding $\Gamma \vdash_{\mathrm{Ax}} \phi$ is easier for 'nicer' axiomatic systems. In particular, the bounded proof property below may hold only for 'nice' axiomatic systems for a logic L.

When we write $\Gamma \vdash^n_{\mathrm{Ax}} \phi$ we mean that ϕ can be proved from Ax, Γ (in the above sense) by using a proof in which only formulae of modal complexity at most n occur.

Definition 2. *We say that* Ax *has the* bounded proof property (*the bpp, for short*) *iff for every formula ϕ of modal complexity at most n and for every Γ containing only formulae of modal complexity at most n, we have*

$$\Gamma \vdash_{\mathrm{Ax}} \phi \quad \Rightarrow \quad \Gamma \vdash^n_{\mathrm{Ax}} \phi.$$

[2] Requirement (ii) is just to avoid possible misunderstanding in Definition 4.

It should be clear that the bpp for a finite axiom system Ax which is adequate for L implies the decidability of the global consequence relation problem for L. This is because we have a bounded search space for possible proofs: in fact, there are only finitely many non-provably equivalent formulae containing a given finite set of variables and having the modal complexity bounded by a given n. Notice that in a proof witnessing $\Gamma(\underline{x}) \vdash^n_{\mathrm{Ax}} \phi(\underline{x})$ we can freely suppose that only the variables $\underline{x}$ occur, because extra variables can be uniformly replaced by, say, a tautology.

3 Step Frames

The aim of this section is to supply a semantic framework for investigating proofs and formulae of modal complexity at most 1. We first recall the definition of one-step frames from [14] and [8], and define conservative one-step frames.

Definition 3. *A* one-step frame *is a quadruple* $\mathcal{S} = (W_1, W_0, f, R)$*, where* W_0, W_1 *are sets,* $f : W_1 \to W_0$ *is a map and* $R \subseteq W_1 \times W_0$ *is a relation between* W_1 *and* W_0*. We say that* $\mathcal{S}$ *is* conservative *iff* f *is surjective and the following condition is satisfied for all* $w_1, w_2 \in W_1$ *:*

$$f(w_1) = f(w_2) \ \& \ R(w_1) = R(w_2) \quad \Rightarrow \quad w_1 = w_2. \tag{9}$$

We shall use the notation $\mathcal{S}^*$ to indicate the so-called complex algebra (one-step modal algebra in the terminology of [7, 8]) formed by the 4-tuple

$$\mathcal{S}^* = (\wp(W_0), \wp(W_1), f^*, \Diamond_R),$$

where f^* is the Boolean algebra homomorphism given by inverse image along f and $\Diamond_R$ is the semilattice morphism associated with R. The latter is defined as follows: for $A \subseteq W_0$, we have $\Diamond_R(A) = \{w \in W_1 \mid R(w) \cap A \neq \emptyset\}$.

Notice that a one-step frame $\mathcal{S} = (W_1, W_0, f, R)$, where $W_0 = W_1$ and $f = id$ is just an ordinary Kripke frame. For clarity, we shall sometimes call Kripke frames *standard* frames.

We spell out what it means for a one-step frame to validate a reduced axiomatic system Ax. Notice that only formulae of modal complexity at most 1 are involved.

An $\mathcal{S}$*-valuation* $\mathtt{v}$ on a one-step frame $\mathcal{S} = (W_1, W_0, f, R)$ is a map associating with each variable x an element $\mathtt{v}(x) \in \wp(W_0)$. For every formula ϕ of complexity 0, we define $\phi^{\mathtt{v}0} \in \wp(W_0)$ inductively as follows:

$$x^{\mathtt{v}0} = \mathtt{v}(x) \quad \text{(for every variable } x\text{);}$$
$$(\phi \wedge \psi)^{\mathtt{v}0} = \phi^{\mathtt{v}0} \cap \psi^{\mathtt{v}0}; \qquad (\phi \vee \psi)^{\mathtt{v}0} = \phi^{\mathtt{v}0} \cup \psi^{\mathtt{v}0}; \qquad (\neg\phi)^{\mathtt{v}0} = W_0 \setminus (\phi^{\mathtt{v}0}).$$

For each formula ϕ of complexity 0, we define $\phi^{\mathtt{v}1} \in \wp(W_1)$ as $f^*(\phi^{\mathtt{v}0})$. For ϕ of complexity 1, $\phi^{\mathtt{v}1} \in \wp(W_1)$ is defined inductively as follows:

$$(\Diamond\phi)^{\mathtt{v}1} = \Diamond_R(\phi^{\mathtt{v}0}); \ (\phi\wedge\psi)^{\mathtt{v}1} = \phi^{\mathtt{v}1}\cap\psi^{\mathtt{v}1}; \ (\phi\vee\psi)^{\mathtt{v}1} = \phi^{\mathtt{v}1}\cup\psi^{\mathtt{v}1}; \ (\neg\phi)^{\mathtt{v}1} = W_1\setminus(\phi^{\mathtt{v}1}).$$

Definition 4. *We say that $\mathcal{S}$* validates the inference rule (8) *iff for every $\mathcal{S}$-valuation* $\mathtt{v}$,*we have that*

$$\phi_1^{\mathtt{v}1} = W_1, \ldots, \phi_n^{\mathtt{v}1} = W_1, \ \textit{imply} \ \psi^{\mathtt{v}1} = W_1.$$

We say that $\mathcal{S}$ validates an axiomatic system Ax (*written* $\mathcal{S} \models$ Ax) *iff $\mathcal{S}$ validates all inferences from* Ax.

Notice that it might well be that $\mathrm{Ax}_1, \mathrm{Ax}_2$ are both adequate for the same logic L, but that only one of them is validated by a given $\mathcal{S}$ (see Section 5).

We can specialize the notion of a valuation to standard frames $\mathfrak{F} = (W, R)$ and obtain well-known definitions from the literature. In particular, given a valuation $\mathtt{v}$, for any formula ϕ (of any modal complexity) we can define $\phi^{\mathtt{v}}$ by

$$\begin{aligned} &x^{\mathtt{v}} = \mathtt{v}(x) \quad \text{(for every variable } x);\\ &(\Diamond\phi)^{\mathtt{v}} = \Diamond_R(\phi^{\mathtt{v}}); \quad (\phi \wedge \psi)^{\mathtt{v}} = \phi^{\mathtt{v}} \cap \psi^{\mathtt{v}};\\ &(\phi \vee \psi)^{\mathtt{v}} = \phi^{\mathtt{v}} \cup \psi^{\mathtt{v}}; \quad (\neg\phi)^{\mathtt{v}} = W \setminus (\phi^{\mathtt{v}}). \end{aligned}$$

We say that $\mathfrak{F}$ *is a frame for L* [10,20] iff $\phi^{\mathtt{v}} = W$ for all $\mathtt{v}$ and all $\phi \in L$.

We now introduce morphisms of one-step frames. In the definition below, we use $\circ$ to denote relational composition: for $R_1 \subseteq X \times Y$ and $R_2 \subseteq Y \times Z$, we have $R_2 \circ R_1 := \{(x, z) \in X \times Z \mid \exists y \in Y \ ((x, y) \in R_1 \ \& \ (y, z) \in R_2)\}$. Notice that the relational composition applies also when one or both of R_1, R_2 are functions.

Definition 5. *A* p-morphism *between step frames $\mathcal{F}' = (W_1', W_0', f', R')$ and $\mathcal{F} = (W_1, W_0, f, R)$ is a pair of surjective maps $\mu : W_1' \longrightarrow W_1$, $\quad \nu : W_0' \longrightarrow W_0$ such that*

$$f \circ \mu = \nu \circ f' \qquad \text{and} \qquad R \circ \mu = \nu \circ R'. \tag{10}$$

Notice that, when $\mathcal{F}'$ is standard (i.e., $W_1' = W_0'$ and $f' = id$), ν must be $f \circ \mu$ and (10) reduces to

$$R \circ \mu = f \circ \mu \circ R'. \tag{11}$$

In the next section we formulate a semantic criterion for an axiomatic system to enjoy the bounded proof property in terms of one-step frames. For this we need to recall extensions of one step-frames [8].

Definition 6. *A* one-step extension *of a one-step frame $\mathcal{S}_0 = (W_1, W_0, f_0, R_0)$ is a one-step frame $\mathcal{S}_1 = (W_2, W_1, f_1, R_1)$ satisfying $R_0 \circ f_1 = f_0 \circ R_1$. A class K of one-step frames has the* extension property *iff every conservative one-step frame $\mathcal{S}_0 = (W_1, W_0, f_0, R_0)$ in K has an extension $\mathcal{S}_1 = (W_2, W_1, f_1, R_1)$ such that f_1 is surjective and $\mathcal{S}_1$ is also in K.*

Theorem 1. *An axiomatic system* Ax *has the bpp iff the class of finite one-step frames validating* Ax *has the extension property.*

We point out again that the proofs of this and other results of this paper can be found in [7]. The characterization of the bpp obtained in Theorem 1 may not be easy to handle, because in concrete examples one would like to avoid managing one-step extensions and would prefer to work with standard frames instead. This is possible, if we combine the bpp with the finite model property.

Definition 7. *A logic L has the (global)* finite model property*, the* fmp *for short, if for every finite set of formulae Γ and for every formula ϕ we have $\Gamma \not\vdash_L \phi$ iff there exist a finite frame $\mathfrak{F} = (W, R)$ for L and a valuation* v *on $\mathfrak{F}$ such that $(\bigwedge \Gamma)^{\mathrm{v}} = W$ and $\phi^{\mathrm{v}} \neq W$.*

We are ready to state our main result.

Theorem 2. *Let L be a logic and* Ax *an axiomatic system adequate for L. The following two conditions are equivalent:*

(i) Ax *has the bpp and L has the fmp;*
(ii) *Every finite conservative one-step frame validating* Ax *is a p-morphic image of some finite frame for L.*

4 One-Step Correspondence

In this section we develop the correspondence theory for one-step frames based on the classical correspondence theory for standard frames.

We will start by reformulating Definition 4. Notice that a one-step frame $\mathcal{S} = (W_1, W_0, f, R)$ is a two-sorted structure for the language $\mathcal{L}_f$ having a unary function and a binary relation symbol. The complex algebra $\mathcal{S}^* = (\wp(W_0), \wp(W_1), f^*, \Diamond_R)$ is also a two-sorted structure for the first-order language $\mathcal{L}_a$ having two sorts, Boolean operations for each of them, and two-sorted unary function symbols that we call i and $\Diamond$ (they are interpreted in $\mathcal{S}^*$ as f^* and $\Diamond_R$, respectively). As a first step, we reformulate the validity of inference in terms of truth of a formula in the language $\mathcal{L}_a$. We need to turn modal formulae ϕ of complexity at most 1 into $\mathcal{L}_a$-terms. This is easily done as follows: just replace every occurrence of a variable x *which is not located inside a modal connective* in ϕ by $i(x)$. Let us call $\tilde{\phi}$ the result of such replacement. The following fact is then clear.

Proposition 2. *A step frame $\mathcal{S}$ validates a reduced inference rule of the kind* (8) *iff considering $\mathcal{S}^*$ as a two-sorted $\mathcal{L}_a$-structure, we have*

$$\mathcal{S}^* \models \forall \underline{x}\, (\tilde{\phi}_1 = 1 \;\&\cdots\&\; \tilde{\phi}_n = 1 \to \tilde{\psi} = 1). \tag{12}$$

If we rewrite (12) in terms of the $\mathcal{L}_f$-structure $\mathcal{S}$, we realize that this is a truth relation regarding a *second order* formula, because the quantifiers $\forall \underline{x}$ range over tuples of subsets. The idea (borrowed from correspondence theory) is to perform symbolic manipulations on (12) and to convert it into a first-order $\mathcal{L}_f$-condition. This procedure works for many concrete examples, although there are cases where it fails. We follow a long line of research, e.g., [3–5, 12, 13, 18, 22] (see also [10, 20]). Similarly to these papers, our basic method is to perform symbolic manipulations on the algebraic language $\mathcal{L}_a$.

We start by enriching $\mathcal{L}_a$. The enrichment comes from the following observations. Let $\mathcal{F} = (W_0, W_1, f, R)$ be a one-step frame. First of all, the morphism $i := f^* : \wp(W_0) \to \wp(W_1)$ has a left i^* and a right adjoint $i_!$. In fact i^* is the direct image along f and $i_!$ is $\neg i^* \neg$. The operator $\Diamond : \wp(W_0) \to \wp(W_1)$ (we skip the index R) also has a right adjoint, which is the Box operator $\blacksquare$ induced by the converse relation R^o of R. We shall make use also of the related Diamond $\blacklozenge$ defined as $\neg\blacksquare\neg$. Thus we enrich $\mathcal{L}_a$ with extra unary function symbols $i^*, i_!, \blacksquare, \blacklozenge$ of appropriate sorts. In addition, we shall make use of the letters w_i^0, w_i^1 to denote *nominals*, namely quantified variables ranging over atoms (i.e., singleton subsets) of $\wp(W_0), \wp(W_1)$, respectively. For simplicity and for readability, we shall avoid the superscript $(-)^1, (-)^0$ indicating the sort of nominals. However, we shall adopt the convention of using preferably the variables $w, w_0, w', \dots$ for nominals of sort 1, the variables $v, v_0, v', \dots$ for nominals of sort 0 and the letters $u, u_0, u', \dots$ for nominals of unspecified sort (i.e., for nominals that might be of both sorts, which are useful in preventing, e.g., rule duplications). We call $\mathcal{L}_a^+$ the enriched language.

The idea is the following. We want to analyze validity of the inference rule (8) in a one-step frame $\mathcal{F}$. We initialize our procedure to:

$$\forall \underline{x}\ (1 \leq \tilde{\phi}_1 \ \&\cdots\&\ 1 \leq \tilde{\phi}_n \to 1 \leq \tilde{\psi}). \tag{13}$$

Here and below, we use abbreviations such as $\alpha \leq \beta$ to mean $\alpha \to \beta = 1$. Usually, we omit external quantifiers $\forall \underline{x}$ and use sequent notation, so that (13) is written as

$$1 \leq \tilde{\phi}_1,\ \dots,\ 1 \leq \tilde{\phi}_n \Rightarrow 1 \leq \tilde{\psi}. \tag{14}$$

We then try to find a sequence of applications of the rules below ending with a formula where *only quantifiers for nominals* occur (that is, the variables $\underline{x}$ have been eliminated). If we succeed, the standard translation can easily and automatically *convert the final formula into a first-order formula in the language* $\mathcal{L}_f$. It is possible to characterize syntactic classes (e.g., Sahlqvist-like classes and beyond) where the procedure succeeds, but for the purposes of this paper we are not interested in the details of such characterizations. They can be obtained in a straightforward way by extending the well-known characterizations, see e.g., [3,13,18]). The rules we use are divided into three groups:

(a) Any set of invertible rules in classical first-order sequent calculus. We refer the reader to proof-theory textbooks such as [21] for more details on this;
(b) Rules for managing nominal quantifiers (see Table 1);
(c) Adjunction rules (see Table 2);
(d) Ackermann rules (see Table 3).

Rules (a)-(b)-(c) are local, in the sense that they can be applied simply by replacing the formula above the line by the formula below the line (or vice versa).

Rules (d) to the contrary require checking global monotonicity conditions at the whole sequent level. Ackermann rules eliminate the quantified variables $\underline{x}$ one by one in successful runs.

When we start from a logic L, we first need to convert the axioms into reduced inference rules. The method indicated in the proof of Proposition 1 has the big advantage of *introducing new quantified variables that can be easily eliminated via the adjunction and the Ackermann rules*, as is shown in the example below.

Table 1. Nominals Rules

$\dfrac{\tilde{\phi} \leq \tilde{\psi}}{\forall u\ (u{\leq}\tilde{\phi}\ \to\ u{\leq}\tilde{\psi})}$	$\dfrac{u \leq \tilde{\psi}_1{\wedge}\tilde{\psi}_2}{u{\leq}\tilde{\psi}_1\ \&\ u{\leq}\tilde{\psi}_2}$	$\dfrac{u \leq \tilde{\psi}_1{\vee}\tilde{\psi}_2}{u{\leq}\tilde{\psi}_1\ \text{or}\ u{\leq}\tilde{\psi}_2}$
$\dfrac{u \leq \neg\tilde{\psi}}{u{\not\leq}\tilde{\psi}}\quad\dfrac{u{\not\leq}\tilde{\psi}}{\tilde{\psi}{\leq}\neg u}$	$\dfrac{w \leq \Diamond\tilde{\psi}}{\exists v\ (w{\leq}\Diamond v\ \&\ v{\leq}\tilde{\psi})}$	$\dfrac{v \leq \blacklozenge\tilde{\psi}}{\exists w\ (v{\leq}\blacklozenge w\ \&\ w{\leq}\tilde{\psi})}$
$\dfrac{u \leq 1}{\top}$	$\dfrac{u \leq 0}{\bot}$	$\dfrac{v \leq i^*(\tilde{\psi})}{\exists w\ (v{\leq}i^*(w)\ \&\ w{\leq}\tilde{\psi})}$

Table 2. Adjunction Rules

$\dfrac{\tilde{\phi} \leq \Box\tilde{\psi}}{\blacklozenge\tilde{\phi} \leq \tilde{\psi}}$	$\dfrac{\tilde{\phi} \leq \blacksquare\tilde{\psi}}{\Diamond\tilde{\phi} \leq \tilde{\psi}}$
$\dfrac{\tilde{\phi} \leq i(\tilde{\psi})}{i^*(\tilde{\phi}) \leq \tilde{\psi}}$	$\dfrac{\tilde{\phi} \leq i_!(\tilde{\psi})}{i(\tilde{\phi}) \leq \tilde{\psi}}$

Table 3. Ackermann Rules

$\dfrac{\Gamma,\ x{\leq}\tilde{\phi} \Rightarrow \Delta}{\Gamma(\tilde{\phi}/x) \Rightarrow \Delta(\tilde{\phi}/x)}$	(x is not in ϕ, is positive in all Γ, negative in all Δ)
$\dfrac{\Gamma,\ \tilde{\phi}{\leq}x \Rightarrow \Delta}{\Gamma(\tilde{\phi}/x) \Rightarrow \Delta(\tilde{\phi}/x)}$	(x is not in ϕ, is negative in all Γ, positive in all Δ)

Example 1. Let us consider the system **K4** that is axiomatized by the axiom $\Box x \to \Box\Box x$. Since this axiom does not have modal complexity 1, we turn it into the inference rule

$$\frac{\Box x \leq y}{\Box x \leq \Box y} \tag{15}$$

following the algorithm in the proof of Proposition 1. We then initialize our procedure to $\Box x \leq i(y) \Rightarrow \Box x \leq \Box y$. By adjunction rules, we obtain

$$i^*(\Box x) \leq y \Rightarrow \Box x \leq \Box y.$$

We can immediately eliminate y via the Ackermann rules and get $\Box x \leq \Box i^*(\Box x)$. We now use nominals rules together with rules (a) (i.e., invertible rules in classical sequent calculus) and get $w \leq \Box x \Rightarrow w \leq \Box i^*(\Box x)$ (notice that the nominal variable w is implicitly universally quantified here). By adjointness we obtain a sequent $\blacklozenge w \leq x \Rightarrow w \leq \Box i^*(\Box x)$ to which the Ackermann rules apply yielding:

$$w \leq \Box i^*(\Box \blacklozenge w).$$

This is a condition involving only (one) quantified variable for nominals. Thus, in the language $\mathcal{L}_f$ for one-step frames it is first-order definable (to do the unfolding, it is sufficient to notice that the nominal w stands in fact for the set $\{w' \in W_1 \mid w' = w\}$). After appropriate simplifications, we obtain

$$\forall w\, \forall v\ (R(w,v) \to \exists w_1\ (f(w_1) = v\ \&\ R(w_1) \subseteq R(w))). \tag{16}$$

5 Examples and Case Studies

In this section we show how to apply Theorem 2 first to basic, and later to more elaborate examples. The methodology is the following. We have three steps, as pointed out in Section 1:

– starting from a logic L, we produce an equivalent axiomatic system Ax_L with reduced rules (there is a default procedure for that, see the proof of Proposition 1);
– we apply the correspondence machinery of Section 4 and try to obtain a first-order formula α_L in the two-sorted language $\mathcal{L}_f$ of one-step frames characterizing the one-step frames validating Ax_L;
– we apply Theorem 2 and try to prove that conservative finite one-step frames satisfying α_L are p-mophic images of standard frames for L.

If we succeed, we obtain both the fmp and bpp for L. In examples, given a finite conservative one-step frame $\mathcal{F} = (X, Y, f, R)$ satisfying α_L, the finite frame required by Theorem 2 is often based on X and the p-morphism is the identity. Thus one must simply define a relation S on X in such a way that (11) holds (with $R' = S$). Condition (11), taking into consideration that μ is the identity, reduces to (4). There are standard templates for S. We give some examples below where the procedure succeeds, supplying also the relevant hints for the definition of the right S.

- $\boxed{L = \mathbf{K}}$: this is the basic normal modal logic. To obtain the appropriate S, we take $S := f^o \circ R$, i.e., we put wSw' iff $f(w') \in R(w)$.
- $\boxed{L = \mathbf{T}}$: this is the logic axiomatized by $\Box x \to x$. The one-step correspondence gives $f \subseteq R$ as the semantic condition equivalent to being a one-step frame for L. To obtain the appropriate S we again take $S := f^o \circ R$.
- $\boxed{L = \mathbf{K4}}$: this is the logic axiomatized by $\Box x \to \Box\Box x$. As we know, this axiom can be turned into the equivalent rule (15) and the one-step correspondence gives (16) as the semantic condition equivalent to being a one-step frame for **K4**. We take S to be $(f^o \circ R) \cap \geq_R$ (where $w \geq_R w'$ is defined as $R(w) \supseteq R(w')$); this is the same as saying that wSw' holds iff $R(w) \supseteq \{f(w')\} \cup R(w')$.
- $\boxed{L = \mathbf{S4}}$ Here one can combine the previous two cases. However, the definition of S as $(f^o \circ R) \cap \geq_R$ simplifies to $\geq_R$ by reflexivity.

The details required to justify the claims are straightforward but sometimes a bit involved (they considerably simplify by using a relational formalism), see [7, Sec. 8] for the details. All the claims we need are easy for current provers (the SPASS prover for instance solves each of the above problems in less than half a second on a common laptop).

Remark 1. Notice that the definition of a conservative finite one-step frame (Definition 3) has two conditions. However, only the first one (namely surjectivity of f) is used in the computations above. In fact, it is not clear whether Theorem 2 holds if we drop the second condition (9) in the definition of a one-step conservative frame.

A Case Study: S4.3. As a more elaborated example we take **S4.3**, which is **S4** plus the axiom

$$\Box(\Box x \to y) \vee \Box(\Box y \to x).$$

Applying the algorithm from the proof of Proposition 1, we obtain a rule which is 'bad' (the bpp fails for the related axiomatic system, see [7, Sec. 9] for details).

Instead of a rule obtained by the procedure of Proposition 1, we axiomatize **S4.3** by using the reflexivity axiom for **T** and the following infinitely many rules proposed by R. Goré [19]:

$$\frac{\cdots \Box y \to x_j \vee \bigvee_{j \neq i} \Box x_i \cdots}{\Box y \to \bigvee_{i=1}^{n} \Box x_i} \tag{17}$$

The rules are indexed by n and the n-th rule has n premises, according to the values $j = 1, \dots, n$. If we collectively do the correspondence theory on these rules, we obtain the following condition on *finite* one-step frames:

$$\forall w\, \forall S \subseteq R(w)\, \exists v \in S\, \exists w' \,(f(w') = v \;\&\; S \subseteq R(w') \subseteq R(w)). \tag{18}$$

This condition is sufficient to prove that a finite one-step frame (X, Y, f, R) satisfying (18) can be extended to a finite frame (X', R') which is a frame for **S4.3**. In the proof, we do not take X' to be X, but we define X' via a specific construction (see [7, Thm. 3]). Summing everything up, we obtain:

Theorem 3. **S4.3** *axiomatized by Goré's rules* (17) *has the bpp and fmp.*

A Case Study: GL. The Gödel-Löb modal logic **GL** can be axiomatized by the axiom $\Box(\Box x \to x) \to \Box x$. This system is known to have the fmp and to be complete with respect to the class of finite irreflexive transitive frames. From the proof-theoretic side, the following rule

$$\frac{x \wedge \Box x \wedge \Box y \to y}{\Box x \to \Box y} \tag{19}$$

has been proposed by Avron in [2], and shown to lead to a cut-eliminating system. We now analyze the axiomatic system for **GL** consisting of the only rule (19). If we analyze the validity of rule (19) in a *finite* one-step frame, we obtain the following condition

$$\forall w \; (R(w) \subseteq \{f(w') \mid R(w') \subset R(w)\}). \tag{20}$$

Notice that condition (20) implies the one-step transitivity condition (16). Using Theorem 2 and a specific construction of the p-morphic extension (see [7, Thm. 4] for details), we obtain:

Theorem 4. **GL** *axiomatized by Avron's rule* (19) *has the bpp and fmp.*

We conclude by mentioning that it is possible to apply our results for showing that the axiomatic system for **GL** obtained by adding to the transitivity rule (15) the well-known Löb rule $\Box x \to x/x$ is indeed unsatisfactory from a proof-theoretic point of view, because the bpp fails for it [7, Ex. 3].

6 Conclusions and Future Work

We have developed a uniform semantic method for analyzing proof systems of modal logics. The method relies on p-morphic extensions of finite one-step frames. In simple cases, by a one-step version of the classical correspondence theory, the application of our methodology is completely algorithmic. This is a concrete step towards *mechanizing the metatheory* of propositional modal logic.

We also analyzed our approach in two nontrivial cases, namely for the cut-free axiomatizations of **S4.3** and **GL** known from the literature. We succeeded in both cases in proving the fmp and bpp by our methods. The proofs are not entirely mechanical, but from the details given in [7] it emerges that they are still based on a common feature: an induction on the cardinality of accessible worlds in finite one-step frames.

For future, it will be important to see whether this method can fruitfully apply to complicated logics arising in computer science applications (such as dynamic logic, linear or branching time temporal logics, the modal μ-calculus, etc.). Another important series of questions concerns the clarification of the relationship between our techniques and standard techniques employed in filtrations and analytic tableaux. Finally, a comparison with the algebraic approach (in a non-distributive context) to cut elimination via MacNeille completions for the full Lambek calculus **FL** developed in [11] might bring further fruitful consequences.

References

1. Abramsky, S.: A Cook's tour of the finitary non-well-founded sets. In: Essays in Honour of Dov Gabbay, pp. 1–18. College Publications (2005)
2. Avron, A.: On modal systems having arithmetical interpretations. J. Symbolic Logic 49(3), 935–942 (1984)
3. van Benthem, J.: Modal logic and classical logic. Indices: Monographs in Philosophical Logic and Formal Linguistics, III. Bibliopolis, Naples (1985)
4. van Benthem, J.: Modal frame correspondences and fixed-points. Studia Logica 83(1-3), 133–155 (2006)
5. van Benthem, J., Bezhanishvili, N., Hodkinson, I.: Sahlqvist correspondence for modal mu-calculus. Studia Logica 100, 31–60 (2012)
6. Bezhanishvili, N., Gehrke, M.: Finitely generated free Heyting algebras via Birkhoff duality and coalgebra. Log. Methods Comput. Sci. 7(2:9), 1–24 (2011)
7. Bezhanishvili, N., Ghilardi, S.: Bounded proofs and step frames. Technical Report 306, Department of Philosophy, Utrecht University (2013)
8. Bezhanishvili, N., Ghilardi, S., Jibladze, M.: Free modal algebras revisited: the step-by-step method. In: Leo Esakia on Duality in Modal and Intuitionistic Logics. Trends in Logic. Springer (to appear, 2013)
9. Bezhanishvili, N., Kurz, A.: Free modal algebras: A coalgebraic perspective. In: Mossakowski, T., Montanari, U., Haveraaen, M. (eds.) CALCO 2007. LNCS, vol. 4624, pp. 143–157. Springer, Heidelberg (2007)
10. Chagrov, A., Zakharyaschev, M.: Modal Logic. The Clarendon Press (1997)
11. Ciabattoni, A., Galatos, N., Terui, K.: Algebraic proof theory for substructural logics: cut-elimination and completions. Ann. Pure Appl. Logic 163(3), 266–290 (2012)
12. Conradie, W., Ghilardi, S., Palmigiano, A.: Unified correspondence. In: Essays in Honour of J. van Benthem (to appear)
13. Conradie, W., Goranko, V., Vakarelov, D.: Algorithmic correspondence and completeness in modal logic. I. The core algorithm SQEMA. Log. Methods Comput. Sci. 2(1:5), 1–26 (2006)
14. Coumans, D., van Gool, S.: On generalizing free algebras for a functor. Journal of Logic and Computation 23(3), 645–672 (2013)
15. Fine, K.: Normal forms in modal logic. Notre Dame J. Formal Logic 16, 229–237 (1975)
16. Ghilardi, S.: An algebraic theory of normal forms. Annals of Pure and Applied Logic 71, 189–245 (1995)
17. Ghilardi, S.: Continuity, freeness, and filtrations. J. Appl. Non-Classical Logics 20(3), 193–217 (2010)
18. Goranko, V., Vakarelov, D.: Elementary canonical formulae: extending Sahlqvist's theorem. Ann. Pure Appl. Logic 141(1-2), 180–217 (2006)
19. Goré, R.: Cut-free sequent and tableaux systems for propositional diodorean modal logics. Technical report, Dept. of Comp. Sci., Univ. of Manchester (1993)
20. Kracht, M.: Tools and techniques in modal logic. Studies in Logic and the Foundations of Mathematics, vol. 142. North-Holland Publishing Co. (1999)
21. Negri, S., von Plato, J.: Structural proof theory. Cambridge University Press, Cambridge (2001)
22. Sambin, G., Vaccaro, V.: A new proof of Sahlqvist's theorem on modal definability and completeness. Journal of Symbolic Logic 54, 992–999 (1989)

Compression of Propositional Resolution Proofs by Lowering Subproofs

Joseph Boudou[1,⋆] and Bruno Woltzenlogel Paleo[2,⋆⋆]

[1] Université Paul Sabatier, Toulouse
joseph.boudou@matabio.net
[2] Vienna University of Technology
bruno@logic.at

Abstract. This paper describes a generalization of the `LowerUnits` algorithm [8] for the compression of propositional resolution proofs. The generalized algorithm, called `LowerUnivalents`, is able to lower not only units but also subproofs of non-unit clauses, provided that they satisfy some additional conditions. This new algorithm is particularly suited to be combined with the `RecyclePivotsWithIntersection` algorithm [8]. A formal proof that `LowerUnivalents` always compresses more than `LowerUnits` is shown, and both algorithms are empirically compared on thousands of proofs produced by the SMT-Solver veriT.

1 Introduction

Propositional resolution is among the most successful proof calculi for automated deduction in propositional logic available today. It provides the foundation for DPLL- and CDCL-based Sat/SMT-solvers [4], which perform surprisingly well in practice [10], despite the NP-completeness of propositional satisfiability [5] and the theoretical difficulty associated with NP-complete problems.

Resolution refutations can also be output by Sat/SMT-solvers with an acceptable efficiency overhead and are detailed enough to allow easy implementation of efficient proof checkers. They can, therefore, be used as certificates of correctness for the answers provided by these tools in case of unsatisfiability.

However, as the refutations found by Sat/SMT-solvers are often redundant, techniques for compressing and improving resolution proofs in a post-processing stage have flourished. Algebraic properties of the resolution operation that might be useful for compression were investigated in [7]. Compression algorithms based on rearranging and sharing chains of resolution inferences have been developed in [1] and [12]. Cotton [6] proposed an algorithm that compresses a refutation by repeteadly splitting it into a proof of a heuristically chosen literal ℓ and a proof of $\overline{\ell}$, and then resolving them to form a new refutation. The `Reduce&Reconstruct` algorithm [11] searches for locally redundant subproofs that can be rewritten into subproofs of stronger clauses and with fewer resolution steps. In [2] two

⋆ Supported by the Google Summer of Code 2012 program.
⋆⋆ Supported by the Austrian Science Fund, project P24300.

linear time compression algorithms are introduced. One of them is a partial regularization algorithm called `RecyclePivots`. An enhanced version of this latter algorithm, called `RecyclePivotsWithIntersection` (`RPI`), is proposed in [8], along with a new linear time algorithm called `LowerUnits`. These two last algorithms are complementary and better compression can easily be achieved by sequentially composing them (i.e. executing one after the other).

In this paper, the new algorithm `LowerUnivalents`, generalizing `LowerUnits`, is described. Its achieved goals are to compress more than `LowerUnits` and to allow fast *non-sequential* combination with `RPI`. While in a sequential combination one algorithm is simply executed after the other, in a non-sequential combination, both algorithms are executed simultaneously when the proof is traversed. Therefore, fewer traversals are needed.

The next section introduces the propositional resolution calculus using notations that are more convenient for describing proof transformation operations. It also describes the new concepts of *active literals* and *valent literals* and proves basic but essential results about them. Section 3 briefly describes the `LowerUnits` algorithm. In Sect. 4 the new algorithm `LowerUnivalents` is introduced and it is proved that it always compresses more than `LowerUnits`. Section 5 describes the non-sequential combination of `LowerUnivalents` and `RPI`. Lastly, experimental results are discussed in Sect. 6.

2 Propositional Resolution Calculus

A *literal* is a propositional variable or the negation of a propositional variable. The *complement* of a literal ℓ is denoted $\overline{\ell}$ (i.e. for any propositional variable p, $\overline{p} = \neg p$ and $\overline{\neg p} = p$). The set of all literals is denoted $\mathcal{L}$. A *clause* is a set of literals. $\bot$ denotes the *empty clause.*

Definition 1 (Proof). *A directed acyclic graph $\langle V, E, \Gamma\rangle$, where V is a set of nodes and E is a set of edges labeled by literals (i.e. $E \subset V \times \mathcal{L} \times V$ and $v_1 \xrightarrow{\ell} v_2$ denotes an edge from node v_1 to node v_2 labeled by ℓ), is a proof of a clause Γ iff it is inductively constructible according to the following cases:*

1. *If Γ is a clause, $\widehat{\Gamma}$ denotes some proof $\langle \{v\}, \varnothing, \Gamma\rangle$, where v is a new node.*
2. *If ψ_L is a proof $\langle V_L, E_L, \Gamma_L\rangle$ and ψ_R is a proof $\langle V_R, E_R, \Gamma_R\rangle$ and ℓ is a literal such that $\overline{\ell} \in \Gamma_L$ and $\ell \in \Gamma_R$, then $\psi_L \odot_\ell \psi_R$ denotes a proof $\langle V, E, \Gamma\rangle$ s.t.*

$$V = V_L \cup V_R \cup \{v\}$$
$$E = E_L \cup E_R \cup \left\{ v \xrightarrow{\overline{\ell}} \rho(\psi_L), v \xrightarrow{\ell} \rho(\psi_R) \right\}$$
$$\Gamma = \left(\Gamma_L \setminus \{\overline{\ell}\}\right) \cup \left(\Gamma_R \setminus \{\ell\}\right)$$

where v is a new node and $\rho(\varphi)$ denotes the root node of φ. □

If $\psi = \varphi_L \odot_\ell \varphi_R$, then φ_L and φ_R are *direct subproofs* of ψ and ψ is a *child* of both φ_L and φ_R. The transitive closure of the direct subproof relation is

```
Input: a proof $\varphi$
Input: $D$ a set of subproofs
Output: a proof $\varphi'$ obtained by deleting the subproofs in $D$ from $\varphi$
if $\varphi \in D$ or $\rho(\varphi)$ has no premises then
    return $\varphi$ ;
else
    let $\varphi_L$, $\varphi_R$ and $\ell$ be such that $\varphi = \varphi_L \odot_\ell \varphi_R$ ;
    let $\varphi'_L$ = delete($\varphi_L$,$D$) ;
    let $\varphi'_R$ = delete($\varphi_R$,$D$) ;
    if $\varphi'_L \in D$ then
        return $\varphi'_R$ ;
    else if $\varphi'_R \in D$ then
        return $\varphi'_L$ ;
    else if $\overline{\ell} \notin \Gamma_{\varphi'_L}$ then
        return $\varphi'_L$ ;
    else if $\ell \notin \Gamma_{\varphi'_R}$ then
        return $\varphi'_R$ ;
    else
        return $\varphi'_L \odot_\ell \varphi'_R$ ;
```

Algorithm 1. `delete`

the *subproof* relation. A subproof which has no direct subproof is an *axiom* of the proof. Contrary to the usual proof theoretic conventions but following the actual implementation of the data structures used by `LowerUnivalents`, edges are directed from children (resolvents) to their parents (premises). V_ψ, E_ψ and Γ_ψ denote, respectively, the nodes, edges and proved clause (conclusion) of ψ.

Definition 2 (Active literals). *Given a proof ψ, the set of active literals $A_\psi(\varphi)$ of a subproof φ are the labels of edges coming into φ's root:*

$$A_\psi(\varphi) = \{\ell \mid \exists \varsigma \in V_\psi.\ \varsigma \xrightarrow{\ell} \rho(\varphi)\}$$

Two operations on proofs are used in this paper: the resolution operation $\odot_\ell$ introduced above and the deletion of a set of subproofs from a proof, denoted $\psi \setminus (\varphi_1 \dots \varphi_n)$ where ψ is the whole proof and φ_i are the deleted subproofs. Algorithm 1 describes the deletion operation, with $\psi \setminus (\varphi_1 \dots \varphi_n)$ being the result of `delete(`ψ,$\{\varphi_1, \dots, \varphi_n\}$`)`. Both the resolution and deletion operations are considered to be left associative.

The deletion algorithm is a minor variant of the Reconstruct-Proof algorithm presented in [3]. The basic idea is to traverse the proof in a top-down manner, replacing each subproof having one of its premises marked for deletion (i.e. in D) by its other direct subproof. The special case when both φ'_L and φ'_R belong to D is treated rather implicitly and deserves an explanation: in such a case, one might intuitively expect the result φ' to be undefined and arbitrary.

Furthermore, to any child of φ, φ' ought to be seen as if it were in D, as if the deletion of φ'_L and φ'_R propagated to φ' as well. Instead of assigning some arbitrary proof to φ' and adding it to D, the algorithm arbitrarily returns (in line 8) φ'_R (which is already in D) as the result φ'. In this way, the propagation of deletion is done automatically and implicitly. For instance, the following hold:

$$\varphi_1 \odot_\ell \varphi_2 \setminus (\varphi_1, \varphi_2) = \varphi_2 \tag{1}$$

$$\varphi_1 \odot_\ell \varphi_2 \odot_{\ell'} \varphi_3 \setminus (\varphi_1, \varphi_2) = \varphi_3 \setminus (\varphi_1, \varphi_2) \tag{2}$$

A side-effect of this clever implicit propagation of deletion is that the actual result of deletion is only meaningful if it is not in D. In the example (1), as $\varphi_1 \odot_\ell \varphi_2 \setminus (\varphi_1, \varphi_2) \in \{\varphi_1, \varphi_2\}$, the actual resulting proof is meaningless. Only the information that it is a deleted subproof is relevant, as it suffices to obtain meaningful results as shown in (2).

Proposition 1. *For any proof ψ and any sets A and B of ψ's subproofs, either $\psi \setminus (A \cup B) \in A \cup B$ and $\psi \setminus (A) \setminus (B) \in A \cup B$, or $\psi \setminus (A \cup B) = \psi \setminus (A) \setminus (B)$.*

Definition 3 (Valent literal). *In a proof ψ, a literal ℓ is* valent *for the subproof φ iff $\overline{\ell}$ belongs to the conclusion of $\psi \setminus (\varphi)$ but not to the conclusion of ψ.*

Proposition 2. *In a proof ψ, every valent literal of a subproof φ is an active literal of φ.*

Proof. Lines 2, 12, 14 and 16 from Algorithm 1 can not introduce a new literal in the conclusion of the subproof being processed. Let ℓ be a valent literal of φ in ψ. Because there is only one subproof to be deleted, $\overline{\ell}$ can only be introduced when processing a subproof φ' such that $\rho(\varphi') \xrightarrow{\ell} \rho(\varphi)$. □

Proposition 3. *Given a proof ψ and a set $D = \{\varphi_1 \dots \varphi_n\}$ of ψ's subproofs, $\forall \ell \in \mathcal{L}$ s.t. ℓ is in the conclusion of $\psi \setminus (D)$ but not in ψ's conclusion, then $\exists i$ s.t. $\overline{\ell}$ is a valent literal of φ_i in ψ.*

3 LowerUnits

When a subproof φ has more than one child in a proof ψ, it may be possible to *factor* all the corresponding resolutions: a new proof is constructed by removing φ from ψ and reintroducing it later. The resulting proof is smaller because φ participates in a single resolution inference in it (i.e. it has a single child), while in the original proof it participates in as many resolution inferences as the number of children it had. Such a factorization is called *lowering* of φ, because its delayed reintroduction makes φ appear at the bottom of the resulting proof.

Formally, a subproof φ in a proof ψ can be lowered if there exists a proof ψ' and a literal ℓ such that $\psi' = \psi \setminus (\varphi) \odot_\ell \varphi$ and $\Gamma_{\psi'} \subseteq \Gamma_\psi$. It has been noted in [8] that φ can always be lowered if it is a *unit*: its conclusion clause has only one literal. This led to the invention of the `LowerUnits` algorithm, which lowers

```
Input: a proof $\psi$
Output: a compressed proof $\psi'$
1 Units $\leftarrow \varnothing$ ;
2 for every subproof $\varphi$ in a bottom-up traversal do
3     if $\varphi$ is a unit and has more than one child then
4         Enqueue $\varphi$ in Units;
5 $\psi' \leftarrow$ delete($\psi$,Units) ;
6 for every unit $\varphi$ in Units do
7     let $\{\ell\} = \Gamma_\varphi$ ;
8     if $\overline{\ell} \in \Gamma_{\psi'}$ then $\psi' \leftarrow \psi' \odot_\ell \varphi$ ;
```

Algorithm 2. `LowerUnits`

every unit with more than one child, taking care to reintroduce units in an order corresponding to the subproof relation: if a unit φ_2 is a subproof of a unit φ_1 then φ_2 has to be reintroduced later than (i.e. below) φ_1.

A possible presentation of `LowerUnits` is shown in Algorithm 2. Units are collected during a first traversal. As this traversal is bottom-up, units are stored in a queue. The traversal could have been top-down and units stored in a stack. Units are effectively deleted during a second, top-down traversal. The last for-loop performs the reintroduction of units.

4 LowerUnivalents

`LowerUnits` does not lower every lowerable subproof. In particular, it does not take into account the already lowered subproofs. For instance, if a unit φ_1 proving $\{a\}$ has already been lowered, a subproof φ_2 with conclusion $\{\neg a, b\}$ may be lowered as well and reintroduced above φ_1. The posterior reintroduction of φ_1 will resolve away $\neg a$ and guarantee that it does not occur in the resulting proof's conclusion. But care must also be taken not to lower φ_2 if $\neg a$ is a valent literal of φ_2, otherwise a will undesirably occur in the resulting proof's conclusion.

Definition 4 (Univalent subproof). *A subproof φ in a proof ψ is* univalent *w.r.t. a set Δ of literals iff φ has exactly one valent literal ℓ in ψ, $\ell \notin \Delta$ and $\Gamma_\varphi \subseteq \Delta \cup \{\ell\}$. ℓ is called the* univalent literal *of φ in ψ w.r.t. Δ.*

The principle of `LowerUnivalents` is to lower all univalent subproofs. Having only one valent literal makes them behave essentially like units w.r.t. the technique of lowering. Δ is initialized to the empty set. Then the complements of the univalent literals are incrementally added to Δ. Proposition 4 ensures that the conclusion of the resulting proof subsumes the conclusion of the original one.

Proposition 4. *Given a proof ψ, if there is a sequence $U = (\varphi_1 \ldots \varphi_n)$ of ψ's subproofs and a sequence $(\ell_1 \ldots \ell_n)$ of literals such that $\forall i \in [1 \ldots n]$, ℓ_i is the univalent literal of φ_i w.r.t. $\Delta_{i-1} = \{\overline{\ell_1} \ldots \overline{\ell_{i-1}}\}$, then the conclusion of*

$$\psi' = \psi \setminus (U) \odot_{\ell_n} \varphi_n \cdots \odot_{\ell_1} \varphi_1$$

subsumes the conclusion of ψ.

Proof. The proposition is proven by induction on n, along with the fact that $\psi \setminus (U) \notin U$. For $n = 0$, $U = \varnothing$ and the properties trivially hold. Suppose a subproof φ_{n+1} of ψ is univalent w.r.t. Δ_n, with univalent literal ℓ_{n+1}. Because $\ell_{n+1} \notin \Delta_n$, there exists a subproof of $\psi \setminus (U)$ with conclusion containing $\overline{\ell_{n+1}}$, and therefore $\psi \setminus (U) \setminus (\varphi_{n+1}) \notin U \cup \{\varphi_{n+1}\}$. Let Γ be the conclusion of $\psi \setminus (U)$. The conclusion of $\psi' = \psi \setminus (U \cup \{\varphi_{n+1}\}) = \psi \setminus (U) \setminus (\varphi_{n+1})$ is included in $\Gamma \cup \{\overline{\ell_{n+1}}\}$. The conclusion of $\psi' \odot_{\ell_{n+1}} \varphi_{n+1}$ is included in $\Gamma \cup \Delta_n$. As $\Gamma \subseteq \Gamma_\psi \cup \Delta_n$, the conclusion of $\psi' \odot_{\ell_{n+1}} \varphi_{n+1} \ldots \odot_{\ell_1} \varphi_1$ is included in Γ_ψ. □

For this principle to lead to proof compression, it is important to take care of the mutual inclusion of univalent subproofs. Suppose, for instance, that $\varphi_i, \varphi_j, \varphi_k \in U$, $i < j < k$, φ_j is a subproof of φ_i but not a subproof of $\psi \setminus (\varphi_i)$, and $\overline{\ell_j} \in \Gamma_{\varphi_k}$. In this case, φ_j will have one more child in

$$\psi \setminus (U) \odot_{\ell_n} \varphi_n \ldots \odot_{\ell_k} \varphi_k \ldots \odot_{\ell_j} \varphi_j \ldots \odot_{\ell_i} \varphi_i \ldots \odot_{\ell_1} \varphi_1$$

than in the original proof ψ. The additional child is created when φ_j is reintroduced. All the other children are reintroduced with the reintroduction of φ_i, because φ_j was not deleted from φ_i.

To solve this issue, `LowerUnivalents` traverses the proof in a top-down manner and simultaneously deletes already collected univalent subproofs, as sketched in Algorithm 3.

Figure 1 shows an example proof and the result of compressing it with `LowerUnivalents`. The top-down traversal starts with the leaves (axioms) and only visits a child when all its parents have already been visited. Assuming the unit with conclusion $\{a\}$ is the first visited leaf, it passes the univalent test in line 5, is marked for lowering (line 8) and the complement of its univalent literal is pushed onto Δ (line 7). When the subproof with conclusion $\{\overline{a}, b\}$ is considered, $\Delta = \{\overline{a}\}$. As this subproof has only one valent literal $b \notin \Delta$ and $\{\overline{a}, b\} \subseteq \Delta \cup \{b\}$, it is marked for lowering as well. At this point, $\Delta = \{\overline{a}, \overline{b}\}$, `Univalents` contains the two subproofs marked for lowering and ψ' is the subproof with conclusion $\{\overline{a}, \overline{b}\}$ shown in Subfig. (b) (i.e. the result of deleting the two marked subproofs from the original proof in Subfig. (a)). No other subproof is univalent; no other subproof is marked for lowering. The final compressed proof (Subfig. (b)) is obtained by reintroducing the two univalent subproofs that had been marked (lines 9 – 12). It has one resolution less than the original. This is so because the subproof with conclusion $\{\overline{a}, b\}$ had been used (resolved) twice in the original proof, but lowering delays its use to a point where a single use is sufficient.

Although the call to `delete` inside the first loop (line 3 to 8) suggests quadratic time complexity, this loop (line 3 to 8) can be (and has been) actually implemented as a recursive function extending a recursive implementation of `delete`. With such an implementation, `LowerUnivalents` has a time complexity linear w.r.t. the size of the proof, assuming the univalent test (at line 5) is performed in constant bounded time.

Determining whether a literal is valent is expensive. But thanks to Proposition 2, subproofs with one active literal which is not in Γ_ψ can be considered instead of subproofs with one valent literal. If the active literal is not valent, the

```
    Input: a proof $\psi$
    Output: a compressed proof $\psi'$
 1  Univalents ← ∅ ;
 2  $\Delta$ ← ∅ ;
 3  for every subproof $\varphi$, in a top-down traversal do
 4      $\psi'$ ← delete($\varphi$,Univalents) ;
 5      if $\psi'$ is univalent w.r.t. $\Delta$ then
 6          let $\ell$ be the univalent literal ;
 7          push $\overline{\ell}$ onto $\Delta$ ;
 8          push $\psi'$ onto Univalents ;
    // At this point, $\psi' = \psi \setminus$ (Univalents)
 9  while Univalents ≠ ∅ do
10      $\varphi$ ← pop from Univalents;
11      $\ell$ ← pop from $\Delta$ ;
12      if $\ell \in \Gamma_{\psi'}$ then $\psi' \leftarrow \varphi \odot_\ell \psi'$ ;
```

Algorithm 3. Simplified `LowerUnivalents`

$$\begin{tikzcd}
\overline{b}, c & & \overline{a}, b & & \overline{a}, \overline{b}, \overline{c} \\
& \overline{a}, c \arrow[ul] \arrow[ur] & & \overline{a}, \overline{c} \arrow[ul] \arrow[ur] & \\
& & \overline{a} \arrow[ul] \arrow[ur] & & a \\
& & & \perp \arrow[ul] \arrow[ur] &
\end{tikzcd}$$

(a) Original proof

$$\begin{tikzcd}
\overline{b}, c & & \overline{a}, \overline{b}, \overline{c} & & \\
& \overline{a}, \overline{b} \arrow[ul] \arrow[ur] & & \overline{a}, b & \\
& & \overline{a} \arrow[ul] \arrow[ur] & & a \\
& & & \perp \arrow[ul] \arrow[ur] &
\end{tikzcd}$$

(b) Compressed proof

Fig. 1. Example of proof crompression by `LowerUnivalents`

corresponding subproof will simply not be reintroduced later (i.e. the condition in line 28 of Algorithm 4 will fail).

While verifying if a subproof could be univalent, some edges might be deleted. If a subproof φ_i has already been collected as univalent subproof with univalent literal ℓ_i and the subproof φ' being considered now has ℓ_i as active literal, the corresponding incoming edges can be removed. Even if ℓ_i is valent for φ', only $\overline{\ell_i}$ would be introduced, and it would be resolved away when reintroducing φ_i. The `delete` operation can be easily modified to remove both nodes and edges.

Algorithm 4 sums up the previous remarks for an efficient implementation of `LowerUnivalents`. As noticed above, sometimes this algorithm may consider a subproof as univalent when it is actually not. But as care is taken when reintroducing subproofs (at line 28), the resulting conclusion still subsumes the original. The test that $\ell \in \Gamma_\varphi$ at line 20 is mandatory since ℓ might have been deleted from Γ_φ by the deletion of previously collected subproofs.

Every node in a proof $\langle V, E, \Gamma \rangle$ has exactly two outgoing edges unless it is the root of an axiom. Hence the number of axioms is $|V| - \frac{1}{2}|E|$ and because there is at least one axiom, the average number of active literals per node is strictly less than two. Therefore, if `LowerUnivalents` is implemented as an improved

```
Data: a proof $\psi$, compressed in place
Input: a set $D_V$ of subproofs to delete
Input: a set $D_E$ of edges to delete
1  Univalents $\leftarrow \varnothing$ ;
2  $\Delta \leftarrow \varnothing$ ;
3  for every subproof $\varphi$, in a top-down traversal of $\psi$ do
       // The deletion part.
4      if $\varphi$ is not an axiom then
5          let $\varphi = \varphi_L \odot_\ell \varphi_R$ ;
6          if $\varphi_L \in D_V$ or $\rho(\varphi) \xrightarrow{\overline{\ell}} \rho(\varphi_L) \in D_E$ then
7              if $\rho(\varphi) \xrightarrow{\ell} \rho(\varphi_R) \in D_E$ then
8                  add $\varphi$ to $D_V$ ;
9              else
10                 replace $\varphi$ by $\varphi_R$ ;
11         else if $\varphi_R \in D_V$ or $\rho(\varphi) \xrightarrow{\overline{\ell}} \rho(\varphi_R) \in D_E$ then
12             if $\rho(\varphi) \xrightarrow{\ell} \rho(\varphi_L) \in D_E$ then
13                 add $\varphi$ to $D_V$ ;
14             else
15                 replace $\varphi$ by $\varphi_L$ ;
       // Test whether $\varphi$ is univalent.
16     ActiveLiterals $\leftarrow \varnothing$ ;
17     for each incoming edge $e = v \xrightarrow{\ell} \rho(\varphi)$, $e \notin D_E$ do
18         if $\overline{\ell} \in \Delta$ then
19             add $e$ to $D_E$ ;
20         else if $\ell \notin \Delta$, $\ell \in \Gamma_\varphi$ and $\ell \notin \Gamma_\psi$ then
21             add $\ell$ to ActiveLiterals;
22     if ActiveLiterals $= \{\ell\}$ and $\Gamma_\varphi \subseteq \Delta \cup \{\ell\}$ then
23         push $\overline{\ell}$ onto $\Delta$ ;
24         push $\varphi$ onto Univalents;
   // Reintroduce lowered subproofs.
25 while Univalents $\neq \varnothing$ do
26     $\varphi \leftarrow$ pop from Univalents;
27     $\ell \leftarrow$ pop from $\Delta$ ;
28     if $\ell \in \Gamma_\psi$ then
29         replace $\psi$ by $\varphi \odot_\ell \psi$ ;
```

Algorithm 4. Optimized `LowerUnivalents` as an enhanced `delete`

recursive `delete`, its time complexity remains linear, assuming membership of literals to the set Δ is computed in constant time.

Proposition 5. *Given a proof ψ, `LowerUnits`(ψ) has at least as many nodes as `LowerUnivalents`(ψ) if there are no two units in ψ with the same conclusion.*

Proof. A unit φ has exactly one active literal ℓ. Therefore φ is collected by `LowerUnivalents` unless $\overline{\ell} \in \Delta$ or $\ell \in \Delta$. If $\overline{\ell} \in \Delta$ all the incoming edges to $\rho(\varphi)$ are deleted. If $\ell \in \Delta$, every edge $v \xrightarrow{\overline{\ell}} v'$ where v is on a path from $\rho(\psi)$ to $\rho(\varphi)$ is deleted. In particular, for every edge $v \xrightarrow{\ell} \rho(\varphi)$ the edge $v \xrightarrow{\overline{\ell}} v'$ is deleted. Moreover, as ℓ is the only literal of φ's conclusion, φ is propagated down the proof until the univalent subproof with valent literal $\overline{\ell}$ is reintroduced. □

In the case where there are at least two units with the same conclusion in ψ, the compressed proof depends on the order in which the units are collected. For both algorithms, only one of these units appears in the compressed proof.

5 Remarks about Combining `LowerUnivalents` with `RPI`

Definition 5 (Regular proof [13]). *A proof ψ is* regular *iff on every path from its root to any of its axioms, each literal labels at most one edge. Otherwise, ψ is* irregular.

Any irregular proof can be converted into a regular proof having the same axioms and the same conclusion. But it has been proved [9] that such a total regularization might result in a proof exponentially bigger than the original.

Nevertheless, *partial* regularization algorithms, such as `RecyclePivots` [2] and `RecyclePivotsWithIntersection` (`RPI`) [8], carefully avoid the worst case of total regularization and do efficiently compress proofs. For any subproof φ of a proof ψ, `RPI` removes the edge $\rho(\varphi) \xrightarrow{\ell} v$ if ℓ is a safe literal for φ.

Definition 6 (Safe literal). *A literal ℓ is* safe *for a subproof φ in a proof ψ iff ℓ labels at least one edge on every path from $\rho(\psi)$ to $\rho(\varphi)$.*

`RPI` performs two traversals. During the first one, safe literals are collected and edges are marked for deletion. The second traversal is the effective deletion similar to the `delete` algorithm.

Both sequential compositions of `LowerUnits` with `RPI` have been shown to achieve good compression ratio [8]. However, the best combination order (`LowerUnits` after `RPI` (`LU.RPI`) or `RPI` after `LowerUnits` (`RPI.LU`)) depends on the input proof. A reasonable solution is to perform both combinations and then to choose the smallest compressed proof, but sequential composition is time consuming. To speed up DAG traversal, it is useful to topologically sort the nodes of the graph first. But in case of sequential composition this costly operation has to be done twice. Moreover, some traversals, like deletion, are identical in both algorithms and might be shared. Whereas implementing a non-sequential combination of `RPI` after `LowerUnits` is not difficult, a non-sequential combination of `LowerUnits` after `RPI` would be complicated. The difficulty is that `RPI` could create some new units which would be visible only after the deletion phase. A solution could be to test for units during deletion. But if units are effectively lowered during this deletion, their deletion would cause some units to become non-units. And postponing deletions of units until a second deletion traversal would prevent the sharing of this traversal and would cause one more topological sorting to be performed, because the deletion phase significantly transforms the structure of the DAG.

Apart from having an improved compression ratio, another advantage of `LowerUnivalents` over `LowerUnits` is that `LowerUnivalents` can be implemented as an enhanced `delete` operation. With such an implementation, a simple non-sequential combination of `LowerUnivalents` after `RPI` can be implemented just by replacing the second traversal of `RPI` by `LowerUnivalents`. After the first traversal of `RPI`, as all edges labeled by a safe literal have been marked for deletion, the remaining active literals are all valent, because for every edge $\rho(\varphi) \xrightarrow{\ell} \rho(\varphi')$, ℓ is either a safe literal of φ or a valent literal of φ'. Therefore, in the second traversal of the non-sequential combination (deletion enhanced by `LowerUnivalents`), all univalent subproofs are lowered.

6 Experiments

`LowerUnivalents` and `LUnivRPI` have been implemented in the functional programming language Scala[1] as part of the Skeptik library[2]. `LowerUnivalents` has been implemented as a recursive `delete` improvement.

The algorithms have been applied to 5 059 proofs produced by the SMT-solver veriT[3] on unsatisfiable benchmarks from the SMT-Lib[4]. The details on the number of proofs per SMT category are shown in Table 1. The proofs were translated into pure resolution proofs by considering every non-resolution inference as an axiom.

The experiment compared the following algorithms:

LU: the `LowerUnits` algorithm from [8];
LUniv: the `LowerUnivalents` algorithm;
RPILU: a non-sequential combination of `RPI` after `LowerUnits`;
RPILUniv: a non-sequential combination of `RPI` after `LowerUnivalents`;
LU.RPI: the sequential composition of `LowerUnits` after `RPI`;
LUnivRPI: the non-sequential combination of `LowerUnivalents` after `RPI` as described in Sect. 5;
RPI: the `RecyclePivotsWithIntersection` from [8];

1 `http://www.scala-lang.org/`
2 `https://github.com/Paradoxika/Skeptik`
3 `http://www.verit-solver.org/`
4 `http://www.smtlib.org/`

Table 1. Number of proofs per benchmark category

Benchmark Category	Number of Proofs
QF_UF	3907
QF_IDL	475
QF_LIA	385
QF_UFIDL	156
QF_UFLIA	106
QF_RDL	30

Split: Cotton's `Split` algorithm ([6]);
RedRec: the `Reduce&Reconstruct` algorithm from [11];
Best RPILU/LU.RPI: which performs both `RPILU` and `LU.RPI` and chooses the smallest resulting compressed proof;
Best RPILU/LUnivRPI: which performs `RPILU` and `LUnivRPI` and chooses the smallest resulting compressed proof.

For each of these algorithms, the time needed to compress the proof along with the number of nodes and the number of axioms (i.e. *unsat core* size) have been measured. Raw data of the experiment can be downloaded from the web[5].

The experiments were executed on the Vienna Scientific Cluster[6] VSC-2. Each algorithm was executed in a single core and had up to 16 GB of memory available. This amount of memory has been useful to compress the biggest proofs (with more than 10^6 nodes).

The overall results of the experiments are shown in Table 2. The compression ratios in the second column are computed according to formula (3), in which ψ ranges over all the proofs in the benchmark and ψ' ranges over the corresponding compressed proofs.

$$1 - \frac{\sum |V_{\psi'}|}{\sum |V_\psi|} \tag{3}$$

The unsat core compression ratios are computed in the same way, but using the number of axioms instead of the number of nodes. The speeds on the fourth column are computed according to formula (4) in which d_ψ is the duration in milliseconds of ψ's compression by a given algorithm.

$$\frac{\sum |V_\psi|}{\sum d_\psi} \tag{4}$$

For the `Split` and `RedRec` algorithms, which must be repeated, a timeout has been fixed so that the speed is about 3 nodes per millisecond.

[5] `http://www.matabio.net/skeptik/LUniv/experiments/`
[6] `http://vsc.ac.at/`

Table 2. Total compression ratios

Algorithm	Compression	Unsat Core Compression	Speed
LU	7.5 %	0.0 %	22.4 n/ms
LUniv	8.0 %	0.8 %	20.4 n/ms
RPILU	22.0 %	3.6 %	7.4 n/ms
RPILUniv	22.1 %	3.6 %	6.5 n/ms
LU.RPI	21.7 %	3.1 %	15.1 n/ms
LUnivRPI	22.0 %	3.6 %	17.8 n/ms
RPI	17.8 %	3.1 %	31.3 n/ms
Split	21.0 %	0.8 %	2.9 n/ms
RedRec	26.4 %	0.4 %	2.9 n/ms
Best RPILU/LU.RPI	22.0 %	3.7 %	5.0 n/ms
Best RPILU/LUnivRPI	22.2 %	3.7 %	5.2 n/ms

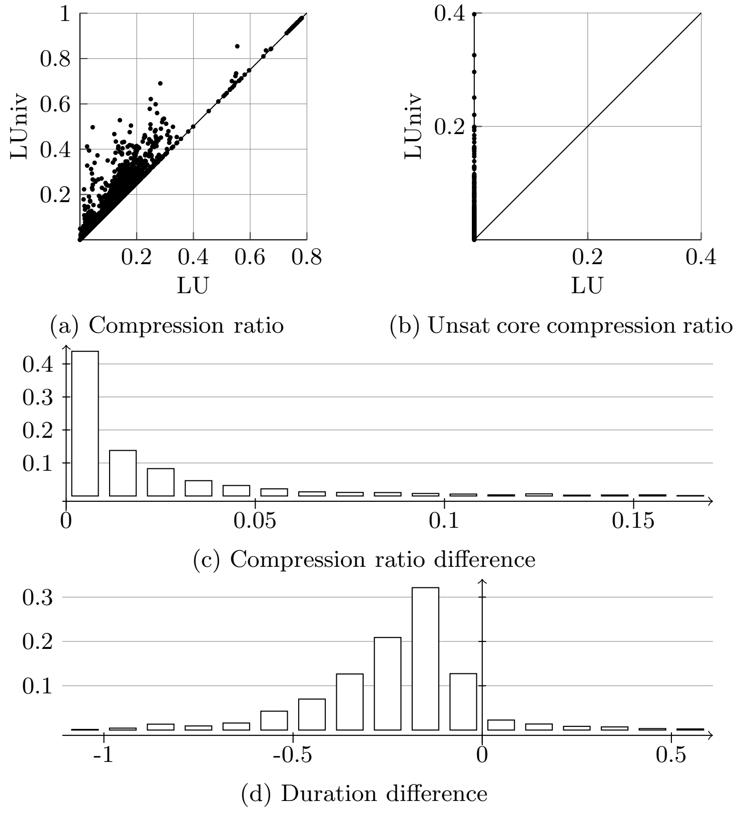
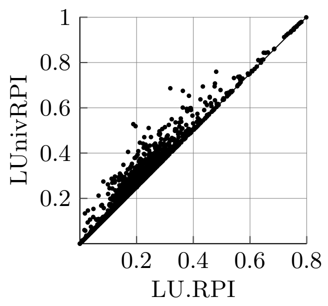
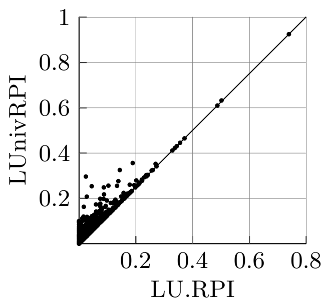


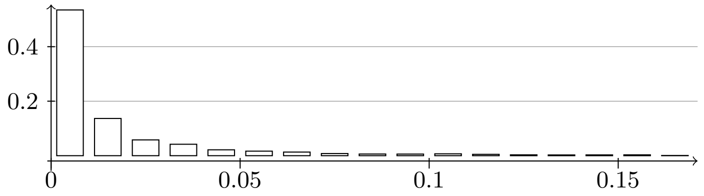
Fig. 2. Comparison between LU and LUniv



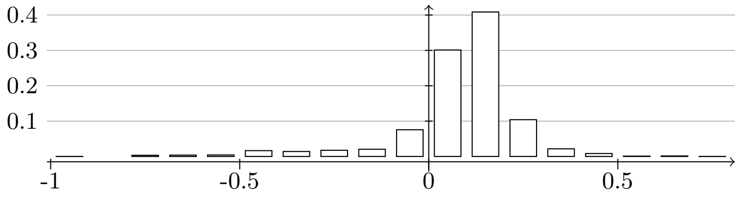
(a) Compression ratio



(b) Unsat core compression ratio



(c) Compression ratio difference



(d) Duration difference

Fig. 3. Comparison between LU.RPI and LUnivRPI

Figure 2 shows the comparison of `LowerUnits` with `LowerUnivalents`. Subfigures (a) and (b) are scatter plots where each dot represents a single benchmark proof. Subfigure (c) is a histogram showing, in the vertical axis, the proportion of proofs having *(normalized) compression ratio difference* within the intervals showed in the horizontal axis. This difference is computed using formula (5) with v_{LU} and v_{LUniv} being the compression ratios obtained respectively by `LowerUnits` and `LowerUnivalents`.

$$\frac{v_{\text{LU}} - v_{\text{LUniv}}}{\frac{v_{\text{LU}}+v_{\text{LUniv}}}{2}} \tag{5}$$

The number of proofs for which $v_{\text{LU}} = v_{\text{LUniv}}$ is not displayed in the histogram. The *(normalized) duration differences* in subfigure (d) are computed using the same formula (5) but with v_{LU} and v_{LUniv} being the time taken to compress the proof by `LowerUnits` and `LowerUnivalents` respectively.

As expected, `LowerUnivalents` always compresses more than `LowerUnits` (subfigure (a)) at the expense of a longer computation (subfigure (d)). And even if the compression gain is low on average (as noticeable in Table 2), subfigure

(a) shows that `LowerUnivalents` compresses some proofs significantly more than `LowerUnits`.

It has to be noticed that veriT already does its best to produce compact proofs. In particular, a forward subsumption algorithm is applied, which results in proofs not having two different subproofs with the same conclusion. This results in `LowerUnits` being unable to reduce unsat core. But as `LowerUnivalents` lowers non-unit subproofs and performs some partial regularization, it achieves some unsat core reduction, as noticeable in subfigure (b).

The comparison of the sequential `LU.RPI` with the non-sequential `LUnivRPI` shown in Fig. 3 outlines the ability of `LowerUnivalents` to be efficiently combined with other algorithms. Not only compression ratios are improved but `LUnivRPI` is faster than the sequential composition for more than 80 % of the proofs.

7 Conclusions and Future Work

`LowerUnivalents`, the algorithm presented here, has been shown in the previous section to compress more than `LowerUnits`. This is so because, as demonstrated in Proposition 5, the set of subproofs it lowers is always a superset of the set of subproofs lowered by `LowerUnits`. It might be possible to lower even more subproofs by finding a characterization of (efficiently) lowerable subproofs broader than that of univalent subproofs considered here. This direction for future work promises to be challenging, though, as evidenced by the non-triviality of the optimizations discussed in Section 4 for obtaining a linear-time implementation of `LowerUnivalents`.

As discussed in Section 5, the proposed algorithm can be embedded in the deletion traversal of other algorithms. As an example, it has been shown that the combination of `LowerUnivalents` with `RPI`, compared to the sequential composition of `LowerUnits` after `RPI`, results in a better compression ratio with only a small processing time overhead (Figure 3). Other compression algorithms that also have a subproof deletion or reconstruction phase (e.g. `Reduce&Reconstruct`) could probably benefit from being combined with `LowerUnivalents` as well.

Acknowledgments. The authors would like to thank Pascal Fontaine for providing veriT's proofs for the experiments, for co-organizing our joint workshops on proof compression, and for several interesting and useful discussions on this topic.

References

1. Amjad, H.: Compressing propositional refutations. Electr. Notes Theor. Comput. Sci. 185, 3–15 (2007)
2. Bar-Ilan, O., Fuhrmann, O., Hoory, S., Shacham, O., Strichman, O.: Linear-time reductions of resolution proofs. In: Chockler, H., Hu, A.J. (eds.) HVC 2008. LNCS, vol. 5394, pp. 114–128. Springer, Heidelberg (2009)

3. Bar-Ilan, O., Fuhrmann, O., Hoory, S., Shacham, O., Strichman, O.: Reducing the size of resolution proofs in linear time. STTT 13(3), 263–272 (2011)
4. Bouton, T., de Oliveira, D.C.B., Déharbe, D., Fontaine, P.: veriT: An open, trustable and efficient SMT-solver. In: Schmidt, R.A. (ed.) CADE 2009. LNCS, vol. 5663, pp. 151–156. Springer, Heidelberg (2009)
5. Cook, S.A.: The complexity of theorem-proving procedures. In: Harrison, M.A., Banerji, R.B., Ullman, J.D. (eds.) STOC, pp. 151–158. ACM (1971)
6. Cotton, S.: Two techniques for minimizing resolution proofs. In: Strichman, O., Szeider, S. (eds.) SAT 2010. LNCS, vol. 6175, pp. 306–312. Springer, Heidelberg (2010)
7. Fontaine, P., Merz, S., Woltzenlogel Paleo, B.: Exploring and exploiting algebraic and graphical properties of resolution. In: 8th International Workshop on Satisfiability Modulo Theories - SMT 2010. Royaume-Uni, Edinburgh (July 2010), http://hal.inria.fr/inria-00544658
8. Fontaine, P., Merz, S., Woltzenlogel Paleo, B.: Compression of propositional resolution proofs via partial regularization. In: Bjørner, N., Sofronie-Stokkermans, V. (eds.) CADE 2011. LNCS, vol. 6803, pp. 237–251. Springer, Heidelberg (2011)
9. Goerdt, A.: Comparing the complexity of regular and unrestricted resolution. In: Marburger, H. (ed.) GWAI. Informatik-Fachberichte, vol. 251, pp. 181–185. Springer (1990)
10. Järvisalo, M., Le Berre, D., Roussel, O., Simon, L.: The international SAT solver competitions. AI Magazine 33(1), 89–92 (2012)
11. Rollini, S.F., Bruttomesso, R., Sharygina, N.: An efficient and flexible approach to resolution proof reduction. In: Barner, S., Harris, I., Kroening, D., Raz, O. (eds.) HVC 2010. LNCS, vol. 6504, pp. 182–196. Springer, Heidelberg (2010)
12. Sinz, C.: Compressing propositional proofs by common subproof extraction. In: Moreno Díaz, R., Pichler, F., Quesada Arencibia, A. (eds.) EUROCAST 2007. LNCS, vol. 4739, pp. 547–555. Springer, Heidelberg (2007)
13. Tseitin, G.S.: On the complexity of derivation in propositional calculus. In: Siekmann, J., Wrightson, G. (eds.) Automation of Reasoning: Classical Papers in Computational Logic 1967–1970, vol. 2. Springer (1983)

A Tableau System for Right Propositional Neighborhood Logic over Finite Linear Orders: An Implementation[*]

Davide Bresolin[1], Dario Della Monica[2], Angelo Montanari[3], and Guido Sciavicco[4]

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

[2] School of Computer Science, University of Reykjavik
Reykjavik, Iceland
dariodm@ru.is

[3] Department of Mathematics and Computer Science,
University of Udine Udine, Italy
angelo.montanari@dimi.uniud.it

[4] Department of Information, Engineering and Communications
University of Murcia, Murcia, Spain
guido@um.es

Abstract. Interval temporal logics are quite expressive temporal logics, which turn out to be difficult to deal with in many respects. Even finite satisfiability of simple interval temporal logics presents non-trivial technical issues when it comes to the implementation of efficient tableau-based decision procedures. We focus our attention on the logic of Allen's relation *meets*, a.k.a. Right Propositional Neighborhood Logic (RPNL), interpreted over finite linear orders. Starting from a high-level description of a tableau system, we developed a first working implementation of a decision procedure for RPNL, and we made it accessible from the web. We report and analyze the outcomes of some initial tests.

1 Introduction

Propositional interval temporal logics play a significant role in computer science, as they provide a natural framework for representing and reasoning about temporal properties in a number of application domains [10]. Interval logic modalities correspond to relations between (pairs of) intervals. In particular, Halpern and Shoham's modal logic of time intervals HS [11] features a set of modalities that

[*] The authors acknowledge the support from the Spanish fellowship program *'Ramon y Cajal' RYC-2011-07821* and the Spanish MEC project *TIN2009-14372-C03-01* (G. Sciavicco), the project *Processes and Modal Logics* (project nr. 100048021) of the Icelandic Research Fund and the project *Decidability and Expressiveness for Interval Temporal Logics* (project nr. 130802-051) of the Icelandic Research Fund in partnership with the European Commission Framework 7 Programme (People) under "Marie Curie Actions" (D. Della Monica), and the Italian GNCS project *Logiche di Gioco estese* (D. Bresolin and A. Montanari).

make it possible to express all Allen's interval relations [1]. HS turns out to be undecidable over all meaningful classes of linear orders, including the class of finite linear orders we are interested in. Temporal reasoning on finite linear orders comes into play in a variety of areas. This is the case, for instance, with planning problems, which consist of finding a finite sequence of actions that, applied to an initial state of the world, leads to a goal state within a bounded amount of time, satisfying suitable conditions about which sequence of states the world must go through. In the last years, a lot of work has been done on (un)decidability and complexity of HS fragments. The complete picture about finite linear orders is given in [6]: there exist 62 non-equivalent decidable fragments of HS, partitioned into four complexities classes, ranging from NP-complete to non-primitive recursive. For each decidable fragment, an optimal decision procedure has been devised. Nevertheless, none of them is available as a working system (they are declarative procedures, which turn out to be unfeasible in practice), with the only exception of the logic of subintervals D over dense linear orders [3]. D has been implemented in LoTrec [8], a generic theorem prover that allows one to specify the rules for his/her own modal/temporal logic. Unfortunately, in general LoTrec is not suitable for interval logics (D over dense linear orders is a very special case), because: (i) it does not support the management of world labels explicitly, and (ii) it does not allow closing conditions based on the number of worlds generated in the construction of a tentative model, but only closing conditions based on patterns and repetitions.

In this paper, we focus our attention on one of the simplest decidable fragment of HS, namely, Right Propositional Neighborhood Logic (RPNL) [4,9], interpreted on finite linear orders, whose satisfiability problem has been proved to be NEXPTIME-complete. RPNL features a single modality corresponding to Allen's relation *meets*. We devised and implemented a working tableau-based decision procedure for RPNL, based on the original (non-terminating) tableau system given in [9], which exploits the small model theorem proved in [7] to guarantee termination.

2 Syntax and Semantics of RPNL

Let $\mathbb{D} = \langle D, < \rangle$ be a finite linear order. An *interval* over $\mathbb{D}$ is an ordered pair $[x, y]$, with $x, y \in D$ and $x < y$ (*strict semantics*). Formulas of RPNL are obtained from a countable set $\mathcal{AP}$ of proposition letters using the standard boolean connectives $\vee$, $\wedge$ and $\neg$, and the temporal modalities $\langle A \rangle$ and $[A]$ (defined as a shorthand for $\neg\langle A \rangle\neg$). Formulas are interpreted on models $M = \langle \mathbb{D}, V \rangle$, where $V : \mathbb{I}(\mathbb{D}) \to 2^{\mathcal{AP}}$ is a *valuation* function that associates every interval of $\mathbb{D}$ with the set of proposition letters that hold true on it. The satisfiability relation $\Vdash$ is defined by the semantic clauses for propositional logic plus the modal clause

$$M, [x, y] \Vdash \langle A \rangle \text{ iff there exists } z > y \text{ such that } M, [y, z] \Vdash \varphi.$$

As shown in [7], satisfiability of RPNL-formulas can be reduced to *initial* satisfiability, that is, satisfiability on the interval $[0, 1]$. Hence, it holds that an RPNL-formula φ is satisfiable if and only if there is a model M such that $M, [0, 1] \Vdash \varphi$.

The decidability proof given in [7] shows that any RPNL-formula φ is satisfiable over finite linear orders if and only if it is satisfiable over a finite linear order whose domain has cardinality strictly less than $2^m \cdot m + 1$, where m is the number of diamonds and boxes in φ. This provides a termination condition that can be used to implement a *fair* procedure that exhaustively searches for a model of size smaller than or equal to the bound. In this paper, we develop and implement a tableau-based decision procedure for RPNL by tailoring the general algorithm described in [9] to it and making use of the bound on the size of the model to guarantee completeness.

3 The Tableau System for RPNL

The abstract structure of a tableau for RPNL is a *rooted tree* where each node is labeled with an *annotated formula* of the form $\psi : [x, y]$, which states that ψ holds over the interval $[x, y]$ on D. Every branch B of the tableau is associated with a finite domain $D_B = \{x_0, x_1, \ldots, x_N\}$ and it represents a partial model for the input formula. At each step of tableau construction, a branch and a node on it are selected and one of the *expansion rules* is applied to expand the branch. Expansion rules follow the semantics of RPNL. They include classical propositional rules plus two additional rules for modalities $[A]$ and $\langle A\rangle$:

$$(box)\ \frac{[A]\psi : [x_i, x_j]}{\psi : [x_j, x_{j+1}], \ldots, \psi : [x_j, x_N]},$$

$$(dia)\ \frac{\langle A\rangle\psi : [x_i, x_j]}{\psi : [x_j, x_{j+1}] \mid \ldots \mid \psi : [x_j, x_N] \mid \psi : [x_j, x'_j] \mid \ldots \mid \psi : [x_j, x'_N]},$$

where, for each $j \le h \le N$, x_h is a point in D_B and x'_h is a new point added to D_B and placed immediately after x_h and immediately before x_{h+1} (when $h < N$). The *(dia)* rule explores all possible ways of satisfying the formula ψ: either it satisfies ψ on an existing interval (nodes labelled with $\psi : [x_j, x_h]$) or it adds a new point x'_h to the domain and it satisfies ψ on the new interval $[x_j, x'_h]$. Similarly, the *(box)* rule asserts that ψ must be true on every existing interval starting at x_j. Thus, the point x_i never appears in the consequent of the rules. A branch in the tableau is declared *closed* if either $p : [x_i, x_j]$ and $\neg p : [x_i, x_j]$ occur on the branch, for some $p \in \mathcal{AP}$ and interval $[x_i, x_j]$ (*contradictory branch*); or the cardinality of the domain D_B is greater than the upper bound on the size of models (*too long branch*). Otherwise, it is considered *open*. Expansion rules are applied only to open branches (closed branches are discarded).

Given a branch B, an annotated formula $\psi : [x_i, x_j]$ is said to be *inactive on B* if and only if ψ is a literal or the rule for ψ has been already applied to it on B, it is *active on B* otherwise. The branch-expansion strategy applied by the system is the simplest possible one: the first (top-down) active formula of the current branch is selected, expanded, and deactivated. Whenever an open branch with no active formulas is found, the procedure terminates with success (the formula is satisfiable). If all branches are closed, the procedure terminates with failure (the formula is unsatisfiable).

4 Implementation of the Tableau System for RPNL

In this section, we illustrate the difficulties we encountered and the implementation choices we made to turn the tableau system described in Section 3 into a computer program. The code of our implementation is written in C++ and it makes no use of external libraries, except for the C++ Standard Library. We exploited suitable data structures to represent formulas, nodes, and branches of the tableau, and we developed a search procedure that keeps track of currently-open branches and expands them by applying expansion rules according to the expansion strategy.

Representation of Formulas, Nodes, and Branches. Since in most applications the input formula φ encodes a set of requirements to be jointly satisfied, e.g., those of a plan, we assume φ to be a logical conjunction, whose conjuncts are entered as distinct lines of a text file. φ is first transformed into an equivalent formula in negated normal form $nnf(\varphi)$. Since such a transformation does not change the number of diamonds and boxes, it does not affect the bound on the maximum cardinality of the domain. Then, φ is stored as a *syntactic tree*, whose leaves are labeled with proposition letters and whose internal nodes are labeled with Boolean connectives and modalities. In such a way, each subformula of φ corresponds to a subtree of the syntactic tree. Nodes of the tableau are represented by a structure with four components: a pointer to the subtree representing the formula labeling the node, two integer variables x and y, that identify the interval annotating the formula, and a Boolean flag, which specifies whether the node is active or not. A branch B is implemented as a list of nodes, enriched with two integer variables N and A representing respectively the cardinality of the domain D_B and the number of active nodes.

The Search Procedure. The search procedure stores the open branches to be expanded into a *priority queue*. At the beginning, the queue contains only the single node initial branch $\{\varphi : [0,1]\}$. Then, the procedure operates as follows: 1. it extracts the branch B with the highest priority from the queue; 2. it checks whether B meets the closure conditions; if so, it deletes the branch and it restarts from 1; 3. it finds the closest-to-the-root active node ν in B; if there are no active nodes in B, it terminates with success and it returns B as a model for φ; 4. it applies the appropriate expansion rule to ν, it deactivates ν, it inserts the branches created by the rule into the queue, and it restarts from 1. The expansion loop is repeated until either a model for φ is found or the queue becomes empty. In the latter case, no model for φ can be found, and the formula is declared unsatisfiable.

Priority Policies. The priority policy of the queue determines the next branch to expand. We implemented five different policies: i) the standard *FIFO* (First In, First Out) policy; ii) expand the branches with the *smallest domain* first (SDF); iii) expand the branches with the *largest domain* first (LDF); iv) expand the branches with the *smallest number of active nodes* first (SAN); v) expand the branches with the *greatest number of active nodes* first (GAN). All the policies are complete: they will eventually check every possible model for the input formula with cardinality less than or equal to the selected bound. By default, the queue follows the FIFO policy, but the user can easily opt for a different one for a particular problem.

Branch Expansion. If the current branch B (extracted from the queue) is declared open at step 2 of the search procedure, nodes in B are scanned to determine the closest-to-the-root active node ν. The expansion of B depends on the shape of the formula labeling ν. Three cases are possible.

Boolean formula. Since formulas are assumed to be in negated normal form, the only possible rules are the $\vee$-rule and the $\wedge$-rule. Let ν be labeled with $\psi \vee \tau : [x_i, x_j]$ (the case $\psi \wedge \tau : [x_i, x_j]$ is similar and thus omitted). We must distinguish four scenarios: (i) both $\psi : [x_i, x_j]$ and $\tau : [x_i, x_j]$ are already on B, (ii) $\psi : [x_i, x_j]$ is on B, while $\tau : [x_i, x_j]$ is not, (iii) $\tau : [x_i, x_j]$ is on B, while $\psi : [x_i, x_j]$ is not, and (iv) neither of the two is on B. In case (i), there is no need to apply the rule: $\psi \vee \tau : [x_i, x_j]$ is deactivated and B is put back in the queue. In case (ii), a copy of the branch is generated and the annotated formula $\tau : [x_i, x_j]$ is added to it; then, $\psi \vee \tau : [x_i, x_j]$ is deactivared and both B and its copy are added to the queue. Case (iii) is completely symmetric. In case (iv), two copies of the branch are generated: one is expanded with the annotated formula $\psi : [x_i, x_j]$, the other one with $\tau : [x_i, x_j]$. Then, $\psi \vee \tau : [x_i, x_j]$ is deactivated, both copies of B are added to the queue, and the original B is discarded.

Box formula $[A]\psi : [x_i, x_j]$. The box rule is applied. First, we deactivate the formula $[A]\psi : [x_i, x_j]$; then, for each $x_j < x_h \leq x_N$, if $\psi : [x_j, x_h]$ does not belong to B, we add it; finally, the expansion of B is inserted into the queue.

Diamond formula $\langle A \rangle\psi : [x_i, x_j]$. The diamond rule is applied. First, we check whether for some $x_h > x_j$ the annotated formula $\psi : [x_j, x_h]$ is on B. If this is the case, we deactivate $\langle A \rangle\psi : [x_i, x_j]$ and we put B back in the queue. Otherwise, we create a distinct copy of B for every possible way of satisfying ψ: $N - j$ copies $B_{j+1}, \ldots, B_N$, with domain cardinality N, that will be expanded with the annotated formulas $\psi : [x_j, x_{j+1}], \ldots, \psi : [x_j, x_N]$, respectively; $N - j + 1$ copies $B'_j, \ldots, B'_N$, with domain cardinality $N + 1$, that will be expanded with the annotated formulas $\psi : [x_j, x'_j], \ldots, \psi : [x_j, x'_N]$, respectively. For each copy B'_h, the expansion of the domain is obtained as follows: (i) every annotated formula $\tau : [x_k, x_l]$ such that $x_k > x_h$ is replaced by the annotated formula $\tau : [x_k + 1, x_l + 1]$. If $x_k \leq x_h < x_l$, the annotated formula is replaced by $\tau : [x_k, x_l + 1]$, while if $x_l \leq x_h$, the annotated formula remains unchanged; (ii) we add a new node labeled with the annotated formula $\psi : [x_j, x_h + 1]$; (iii) we reactivate all annotated formulas $[A]\tau : [x_k, x_l]$ with $x_l \leq x_h$. To conclude the expansion, we deactivate $\langle A \rangle\psi : [x_i, x_j]$, we put all $2 \cdot (N - j) + 1$ copies of B in the queue, and we discard B.

5 Experiments

We have tested our implementation against a benchmark of different problems, divided into two classes. First, we tested the scalability of the program with

Table 1. Experimental results

COMBINATORICS

	Policy (sec)					Outcome
n	**FIFO**	**SDF**	**LDF**	**SAN**	**GAN**	**(size)**
1	0.004	0.004	0.004	0.004	0.004	4
2	0.004	0.008	0.004	0.004	0.008	5
3	0.008	0.15	0.03	0.008	0.03	6
4	0.01	–	30.07	0.01	30.29	7
5	0.012	–	–	0.012	–	8
6	0.02	–	–	0.03	–	9
7	0.07	–	–	0.07	–	10
8	0.15	–	–	0.16	–	11
9	0.3	–	–	0.32	–	12
10	0.56	–	–	0.59	–	13
11	0.99	–	–	1.06	–	14
12	1.67	–	–	1.79	–	15
13	2.73	–	–	2.94	–	16
14	4.25	–	–	4.55	–	17
15	6.56	–	–	7.08	–	18
16	9.77	–	–	10.82	–	19
17	14.42	–	–	15.40	–	20
18	20.79	–	–	22.20	–	21
19	29.28	–	–	32.11	–	22
20	40.91	–	–	44.09	–	23
21	–	–	–	–	–	–
22	–	–	–	–	–	–

RANDOMIZED

	Policy (sec)					Outcome
n	**FIFO**	**SDF**	**LDF**	**SAN**	**GAN**	**(size)**
1	0.004	0.004	0.004	0.004	0.004	4
2	0.004	0.004	0.004	0.004	0.004	4
3	0.004	0.004	0.004	0.004	0.004	4
4	0.004	0.004	0.004	0.004	0.004	4
5	0.004	0.004	0.004	0.004	0.004	4
6	0.004	0.004	0.004	0.004	0.004	4
7	0.07	0.23	0.004	0.18	0.004	3 / 4
8	0.004	0.004	0.004	0.004	0.004	4
9	0.004	0.004	0.004	0.004	0.004	4
10	0.004	0.004	0.004	0.004	0.004	4
11	0.004	0.004	0.004	0.004	0.004	4
12	0.004	0.004	0.004	0.004	0.004	4
13	0.01	0.04	0.004	0.02	0.004	4
14	0.004	0.004	0.004	0.004	0.004	4
15	0.004	0.004	0.004	0.004	0.004	4
16	0.004	1.37	0.004	0.01	0.004	4
17	0.004	0.004	0.004	0.004	0.004	4
18	0.004	0.004	0.004	0.004	0.004	3
19	1.66	45.43	0.68	1.91	0.02	3 / 4
20	0.02	0.004	0.03	0.03	0.004	2 / 4
21	0.004	0.004	0.004	0.004	0.004	4
22	0.74	14.08	0.004	1.04	0.004	4
23	0.004	0.004	0.004	0.004	0.004	4
24	0.004	0.004	0.004	0.004	0.004	4
25	–	–	–	–	–	–
26	0.004	0.004	0.004	0.004	0.004	4
27	0.004	–	0.004	0.01	–	3 / 4
28	0.004	0.004	0.004	0.004	0.004	4
29	0.004	–	0.004	0.004	0.004	4
30	0.14	0.08	0.04	0.19	0.01	2 / 4
31	0.004	0.004	0.004	0.004	0.004	unsat
32	0.25	–	0.02	0.31	0.004	2 / 4
33	0.004	0.004	0.004	0.004	0.004	4
34	–	–	0.02	0.004	0.02	2 / 4
35	0.004	–	0.004	–	0.004	2 / 4
36	–	–	–	–	1.2	3

respect to a set of combinatorial problems of increasing complexity (COMBINATORICS), where the n-th combinatorial problem is defined as the problem of finding a model for the formula that contains n conjuncts, each one of the type $\langle A\rangle p_i$ $(0 \leq i \leq n)$, plus $\frac{n(n+1)}{2}$ formulas of the type $[A]\neg(p_i \wedge p_j)$ $(i \neq j)$. Then, we considered the set of 36 "easy" purely randomized formulas used in [5] to evaluate an Evolutionary Computation algorithm for RPNL finite satisfiability (RANDOMIZED). Table 1 summarizes the outcome of our experiments. For each class of problems, the corresponding table shows, for each instance n, the time necessary to solve the problem for each policy (FIFO, SDF, LDF, SAN, GAN) and the size of the obtained model (or "unsat" if the instance was proved to be unsatisfiable). A time-out of 1 minute was used to stop instances running for too long. All the experiments were executed on a notebook with an Intel Pentium Dual-Core Mobile 1.6 Ghz CPU and 2 Gb of RAM, under Ubuntu Linux 11.04. Despite being a prototypical implementation, our system runs reasonably well on the COMBINATORICS benchmark, being able to produce a result in a short time for formulas up to 20 conjuncts (and up to a model size of 23 points). The results of the RANDOMIZED benchmark allows for a first comparison with the Evolutionary algorithm in [5], and shows that the two algorithms have similar performances on the considered formulas. The tableau system was able to prove that problem 31 is unsatisfiable, while the evolutionary algorithm (being incomplete) can only provide positive answers. It is important to stress that there is no available benchmark neither for RPNL, nor for any other interval temporal logic. To overcome this limitation, we are currently working to adapt some benchmarks for the modal logic K [2] and for the temporal logic LTL [12] to the interval semantics. On the web-page *http://www.di.unisa.it/dottorandi/dario.dellamonica/tableaux/* it is possible to find the system available for testing.

References

1. Allen, J.: Maintaining knowledge about temporal intervals. Communications of the ACM 26(11), 832–843 (1983)
2. Balsiger, P., Heuerding, A., Schwendimann, S.: A benchmark method for the propositional modal logics K, KT, S4. J. of Automated Reasoning 24(3), 297–317 (2000)
3. Bresolin, D., Goranko, V., Montanari, A., Sala, P.: Tableaux for logics of subinterval structures over dense orderings. J. of Logic and Computation 20(1), 133–166 (2010)
4. Bresolin, D., Goranko, V., Montanari, A., Sciavicco, G.: Propositional interval neighborhood logics: Expressiveness, decidability, and undecidable extensions. Annals of Pure and Applied Logic 161(3), 289–304 (2009)
5. Bresolin, D., Jiménez, F., Sánchez, G., Sciavicco, G.: Finite satisfiability of propositional interval logic formulas with multi-objective evolutionary algorithms. In: Proc. of the 12th FOGA (in press, 2013)
6. Bresolin, D., Della Monica, D., Montanari, A., Sala, P., Sciavicco, G.: Interval temporal logics over finite linear orders: the complete picture. In: Proc. of the 20th ECAI, pp. 199–204 (2012)
7. Bresolin, D., Montanari, A., Sciavicco, G.: An optimal decision procedure for Right Propositional Neighborhood Logic. J. of Automated Reasoning 38(1-3), 173–199 (2007)
8. Fariñas del Cerro, L., Fauthoux, D., Gasquet, O., Herzig, A., Longin, D., Massacci, F.: Lotrec: the generic tableau prover for modal and description logics. In: Goré, R.P., Leitsch, A., Nipkow, T. (eds.) IJCAR 2001. LNCS (LNAI), vol. 2083, pp. 453–458. Springer, Heidelberg (2001)
9. Goranko, V., Montanari, A., Sciavicco, G.: Propositional interval neighborhood temporal logics. J. of Universal Computer Science 9(9), 1137–1167 (2003)
10. Goranko, V., Montanari, A., Sciavicco, G.: A road map of interval temporal logics and duration calculi. J. of Applied Non-Classical Logics 14(1-2), 9–54 (2004)
11. Halpern, J., Shoham, Y.: A propositional modal logic of time intervals. J. of the ACM 38(4), 935–962 (1991)
12. Rozier, K.Y., Vardi, M.Y.: LTL satisfiability checking. Int. J. on Software Tools for Technology Transfer 2(12), 123–137 (2010)

Hypersequent and Labelled Calculi for Intermediate Logics*

Agata Ciabattoni[1], Paolo Maffezioli[2], and Lara Spendier[1]

[1] Vienna University of Technology
[2] University of Groningen

Abstract. Hypersequent and labelled calculi are often viewed as antagonist formalisms to define cut-free calculi for non-classical logics. We focus on the class of intermediate logics to investigate the methods of turning Hilbert axioms into hypersequent rules and frame conditions into labelled rules. We show that these methods are closely related and we extend them to capture larger classes of intermediate logics.

1 Introduction

The lack of cut-free sequent calculi for logics having natural semantic characterizations and/or simple axiomatizations has prompted the search for generalizations of the Gentzen sequent framework. Despite the large variety of formalisms introduced in the literature (see e.g., [17]), there are two main approaches. In the *syntactic approach* sequents are generalized by allowing extra structural connectives in addition to sequents' comma; in the *semantic approach* the semantic language is explicit part of the syntax in sequents and rules.

Hypersequent calculus [2] is a prominent example of the syntactic approach, while labelled calculi internalizing Kripke semantics [15,8,16,10] are the most developed systems within the semantic approach. Hypersequent and labelled calculus are general-purpose formalisms powerful enough to capture logics of a different nature ranging from modal to substructural logics [8,16,10,3], and are often viewed as antagonist formalisms to define cut-free calculi.

In this paper we focus on propositional intermediate logics, i.e. logics between intuitionistic and classical logic, in order to analyze and compare the methods in [7,5] for defining cut-free hypersequent and labelled calculi. Intermediate logics are an adequate case study for two reasons: (i) Although most of them have a simple axiomatization obtained by extending intuitionistic logic IL with suitable axioms, and have a natural Kripke semantics defined by imposing conditions on the standard intuitionistic frame, corresponding cut-free sequent calculi cannot be defined in a modular way by simply extending the Gentzen sequent calculus **LJ** for IL with new axioms or rules, see [5]. (ii) Cut-free hypersequent and labelled systems have been provided for a large class of intermediate logics in a modular and algorithmic way in [5,7]. The resulting calculi are indeed defined by adding to the base (hypersequent or labelled) calculus for IL extra *structural* rules

* Work supported by FWF START Y544-N23.

corresponding to the additional conditions characterizing the considered logic. The extra rules are constructed in an algorithmic way by turning Hilbert axioms into hypersequent rules [5] and by turning frame conditions –that are formulas of first-order classical logic– into labelled rules [7]. The main differences between these methods are their starting point (syntactic vs. semantic specifications of the considered logics) and their approach: systematic, i.e. based on a syntactic classification of Hilbert axioms in the case of hypersequents, and presenting a specific class of frame conditions (called *geometric formulas*) for which the method works, in the case of labelled sequents.

In this paper we analyze both methods and refine the approach in [5] (and [6]) to introduce new cut-free calculi for intermediate logics. *For hypersequents*: we define a first cut-free hypersequent calculus for the logic Bd_2 [4], one of the seven interpolable intermediate logics and the only one still lacking a cut-free hypersequent calculus. Our calculus is obtained by adapting the method in [6] to extract a *logical* hypersequent rule out of the peculiar axiom of Bd_2, and then modifying the obtained rule to make the cut-elimination go through. *For labelled sequents*: we classify frame conditions according to their quantifier alternation and apply to them the algorithm in [5]; the rules resulting from geometric formulas coincide with those obtained by the method in [7].

2 Preliminaries

The language of propositional intermediate logics consists of infinitely many propositional variables $p, q \ldots$, the connectives & (conjunction), $\vee$ (disjunction), $\supset$ (implication), and the constant $\bot$ for falsity. $\varphi, \psi, \alpha, \beta \ldots$ are formulas built from atoms by using connectives and $\bot$. As usual, $\sim \varphi$ abbreviates $\varphi \supset \bot$.

An intuitionistic frame is a pair $\mathfrak{F} = \langle W, \leqslant \rangle$ where W is a non-empty set, and $\leqslant$ is a reflexive and transitive (accessibility) relation on W. An intuitionistic model $\mathfrak{M} = \langle \mathfrak{F}, \Vdash \rangle$ is a frame $\mathfrak{F}$ together with a relation $\Vdash$ (called the forcing) between elements of W and atomic formulas. Intuitively, $x \Vdash p$ means that the atom p is true at x. Forcing is assumed to be monotonic w.r.t. the relation $\leqslant$, namely, if $x \leqslant y$ and $x \Vdash p$ then also $y \Vdash p$. It is defined inductively on arbitrary formulas as follows:

$x \Vdash \bot$ for no x	$x \Vdash \varphi \& \psi$ iff $x \Vdash \varphi$ and $x \Vdash \psi$
$x \Vdash \varphi \vee \psi$ iff $x \Vdash \varphi$ or $x \Vdash \psi$	$x \Vdash \varphi \supset \psi$ iff $x \leqslant y$ and $y \Vdash \varphi$ implies $y \Vdash \psi$

Intermediate logics are obtained from intuitionistic logic IL either by (i) adding suitable axioms to the Hilbert system for IL or (ii) imposing on intuitionistic frames additional conditions on the relation $\leqslant$. The latter conditions are usually expressed as formulas of first-order classical logic CL in which variables are interpreted as elements of W, and the binary predicate $\leqslant$ denotes the accessibility relation of $\mathfrak{F}$. Atomic formulas are *relational atoms* of the form $x \leqslant y$. Compound formulas are built from relational atoms using the propositional connectives $\wedge$, $\vee$, $\rightarrow$, $\neg$, and the quantifiers $\forall$ and $\exists$.

Example 1. The intermediate logics below are obtained by extending IL with the given axiom or frame condition for the accessibility relation $\leqslant$.

Logic	**Axioms**	**Frame conditions**
Jankov	*(wc)* $\sim\varphi\vee\sim\sim\varphi$	$\forall x\forall y\forall z((x\leqslant y\wedge x\leqslant z)\rightarrow\exists w(y\leqslant w\wedge z\leqslant w))$
Gödel	*(lin)* $(\varphi\supset\psi)\vee(\psi\supset\varphi)$	$\forall x\forall y\forall z((x\leqslant y\wedge x\leqslant z)\rightarrow(y\leqslant z\vee z\leqslant y))$
Bd_2	*(bd$_2$)* $\xi\vee(\xi\supset(\varphi\vee(\varphi\supset\psi)))$	$\forall x\forall y\forall z((x\leqslant y\wedge y\leqslant z)\rightarrow(y\leqslant x\vee z\leqslant y))$
CL	*(em)* $\varphi\vee\sim\varphi$	$\forall x\forall y(x\leqslant y\rightarrow y\leqslant x)$

Hypersequent and Labelled Calculi. Introduced by Avron in [2], the *hypersequent calculus* is a simple generalization of Gentzen's sequent calculus whose basic objects are finite disjunctions of sequents.

Definition 1. *A hypersequent is a finite multiset $\Gamma_1\Rightarrow\Delta_1|\cdots|\Gamma_n\Rightarrow\Delta_n$ where each $\Gamma_i\Rightarrow\Delta_i, i=1,\ldots,n$ is a sequent, called a* component *of the hypersequent. If all components of a hypersequent contain at most one formula in the succedent, the hypersequent is called* single-conclusion*, and* multiple-conclusion *otherwise.*

A hypersequent calculus is defined by incorporating Gentzen's original calculus (e.g., **LJ**, **LK** or a substructural version of it) as a sub-calculus and adding an additional layer of information by considering a single sequent to live in the context of hypersequents. This opens the possibility to define new rules that "exchange information" between different sequents. This type of rule increases the expressive power of hypersequent calculi compared to ordinary sequent calculi and allows us to capture the characteristic axioms of several intermediate logics. *Labelled systems* are a variant of sequent calculus in which the relational semantics of the formalized logics is made explicit part of the syntax [8,16,10]. In a labelled system, each formula φ receives a label x, indicated by $x:\varphi$. The labels are interpreted as possible worlds, and a labelled formula $x:\varphi$ corresponds to $x\Vdash\varphi$. Moreover, labels may occur also in expressions for accessibility relation (relational atoms) like, e.g., $x\leqslant y$ of intuitionistic and intermediate logics.

Definition 2. *A labelled sequent is a sequent consisting of labelled formulas and relational atoms.*

Table 1 depicts the labelled calculus **G3I** for IL. Note that its logical rules are obtained directly from the inductive definition of forcing. The rule $R\supset$ must satisfy the *eigenvariable* condition (y does not occur in the conclusion). The structural rules Ref and $Trans$ for relational atoms correspond to the assumptions of reflexivity and transitivity of $\leqslant$ in $\mathfrak{F}$.

3 Hypersequent Calculi for Intermediate Logics

It was shown in [5] how to transform a large class of Hilbert axioms into structural hypersequent rules in a systematic way. This allowed for the automated definition of cut-free hypersequent calculi for a large class of (substructural) logics. In the case of intermediate logics, the transformation in [5] works for all axioms within the class $\mathcal{P}_3$ of the classification (*substructural hierarchy*) defined

Table 1. Labelled calculus **G3I** for IL [7]

$$x \leqslant y, x : p, \Gamma \Rightarrow \Delta, y : p \qquad \frac{x : \varphi, x : \psi, \Gamma \Rightarrow \Delta}{x : \varphi \& \psi, \Gamma \Rightarrow \Delta}\, L\& \qquad \frac{\Gamma \Rightarrow \Delta, x : \varphi \quad \Gamma \Rightarrow \Delta, x : \psi}{\Gamma \Rightarrow \Delta, x : \varphi \& \psi}\, R\&$$

$$\frac{}{x : \bot, \Gamma \Rightarrow \Delta}\, L\bot \qquad \frac{\Gamma \Rightarrow \Delta, x : \varphi, x : \psi}{\Gamma \Rightarrow \Delta, x : \varphi \vee \psi}\, R\vee \qquad \frac{x : \varphi, \Gamma \Rightarrow \Delta \quad x : \psi, \Gamma \Rightarrow \Delta}{x : \varphi \vee \psi, \Gamma \Rightarrow \Delta}\, L\vee$$

$$\frac{x \leqslant x, \Gamma \Rightarrow \Delta}{\Gamma \Rightarrow \Delta}\, \mathit{Ref} \qquad \frac{x \leqslant y, y : \varphi, \Gamma \Rightarrow \Delta, y : \psi}{\Gamma \Rightarrow \Delta, x : \varphi \supset \psi}\, R\supset \qquad \frac{x \leqslant z, x \leqslant y, y \leqslant z, \Gamma \Rightarrow \Delta}{x \leqslant y, y \leqslant z, \Gamma \Rightarrow \Delta}\, \mathit{Trans}$$

$$\frac{x \leqslant y, x : \varphi \supset \psi, \Gamma \Rightarrow \Delta, y : \varphi \quad x \leqslant y, x : \varphi \supset \psi, y : \psi, \Gamma \Rightarrow \Delta}{x \leqslant y, x : \varphi \supset \psi, \Gamma \Rightarrow \Delta}\, L\supset$$

by the following grammar[1] based on the (propositional) language of **LJ**: $\mathcal{N}_0, \mathcal{P}_0$ contain the set of atomic formulas.

$$\mathcal{P}_{n+1} ::= \bot \mid \top \mid \mathcal{N}_n \mid \mathcal{P}_{n+1} \& \mathcal{P}_{n+1} \mid \mathcal{P}_{n+1} \vee \mathcal{P}_{n+1}$$
$$\mathcal{N}_{n+1} ::= \bot \mid \top \mid P_n \mid \mathcal{N}_{n+1} \& \mathcal{N}_{n+1} \mid \mathcal{P}_{n+1} \supset \mathcal{N}_{n+1}$$

The classes $\mathcal{P}_n$ and $\mathcal{N}_n$ contain axioms with leading positive and negative connective, respectively. A connective is positive (negative) if its left (right) logical rule is invertible [1]; note that in the sequent calculus **LJ**, $\vee$ is positive, $\supset$ is negative and & is both positive and negative.

Example 2. The axioms (*lin*), (*wc*) and (*em*) in Example 1 are within the class $\mathcal{P}_3$. The corresponding hypersequent rules can be generated using the PROLOG-system *AxiomCalc*, which implements the algorithm in [5] and is available at `http://www.logic.at/people/lara/axiomcalc.html`.

Theorem 1 ([5]). *Given an axiom $\mathcal{A} \in \mathcal{P}_3$, the rules generated by the algorithm in [5] are sound and complete for the intermediate logic* IL$+\mathcal{A}$ *and they preserve cut elimination when added to the hypersequent version of* **LJ**.

3.1 Extending the Method - A Case Study

Not all axioms defining intermediate logics are within the class $\mathcal{P}_3$. For instance, (bd_2) (i.e. $\xi \vee (\xi \supset (\varphi \vee (\varphi \supset \psi)))$) is in $\mathcal{P}_4$ and cannot be transformed into an equivalent structural rule using the procedure in [5]. In this section we show how to combine a heuristic method with the procedure in [5] (in fact, its classical and multiple-conclusion version in [6]) to introduce a *logical* rule for (bd_2). We present ad-hoc proofs of soundness, completeness and cut elimination for the resulting calculus.

[1] The substructural hierarchy, as originally defined in [5], is based on the language of Full Lambek calculus with exchange and on the invertibility of its logical rules.

Table 2. Hypersequent calculus **HLJ'**

$\dfrac{}{G \mid \varphi \Rightarrow \varphi}\ (init)$	$\dfrac{}{G \mid \bot \Rightarrow}\ (\bot, l)$	$\dfrac{G \mid \Sigma \Rightarrow \Pi \mid \Gamma \Rightarrow \Delta}{G \mid \Gamma \Rightarrow \Delta \mid \Sigma \Rightarrow \Pi}\ (ee)$	$\dfrac{G \mid \Gamma \Rightarrow \Delta}{G \mid \Gamma, \varphi \Rightarrow \Delta}\ (w, l)$
$\dfrac{G \mid \Gamma \Rightarrow \varphi, \Delta \quad G \mid \Gamma, \psi \Rightarrow \Delta}{G \mid \Gamma, \varphi \supset \psi \Rightarrow \Delta}\ (\supset, l)$	$\dfrac{G \mid \Gamma, \varphi \Rightarrow \psi}{G \mid \Gamma \Rightarrow \varphi \supset \psi, \Delta}\ (\supset, r)$	$\dfrac{G \mid \Gamma \Rightarrow \varphi, \varphi, \Delta}{G \mid \Gamma \Rightarrow \varphi, \Delta}\ (c, r)$	
$\dfrac{G \mid \Gamma \Rightarrow \varphi, \Delta \quad G \mid \Gamma \Rightarrow \psi, \Delta}{G \mid \Gamma \Rightarrow \varphi \& \psi, \Delta}\ (\&, r)$	$\dfrac{G \mid \varphi, \psi, \Gamma \Rightarrow \Delta}{G \mid \varphi \& \psi, \Gamma \Rightarrow \Delta}\ (\&, l)$	$\dfrac{G \mid \Gamma, \varphi, \varphi \Rightarrow \Delta}{G \mid \Gamma, \varphi \Rightarrow \Delta}\ (c, l)$	
$\dfrac{G \mid \varphi, \Gamma \Rightarrow \Delta \quad G \mid \psi, \Gamma \Rightarrow \Delta}{G \mid \varphi \vee \psi, \Gamma \Rightarrow \Delta}\ (\vee, l)$	$\dfrac{G \mid \Gamma \Rightarrow \varphi, \psi, \Delta}{G \mid \Gamma \Rightarrow \varphi \vee \psi, \Delta}\ (\vee, r)$	$\dfrac{G \mid \Gamma \Rightarrow \Delta}{G \mid \Gamma \Rightarrow \varphi, \Delta}\ (w, r)$	
$\dfrac{G \mid \Gamma \Rightarrow \varphi, \Delta \quad H \mid \varphi, \Sigma \Rightarrow \Pi}{G \mid H \mid \Gamma, \Sigma \Rightarrow \Pi, \Delta}\ (cut)$	$\dfrac{G \mid \Gamma \Rightarrow \Delta \mid \Gamma \Rightarrow \Delta}{G \mid \Gamma \Rightarrow \Delta}\ (ec)$	$\dfrac{G}{G \mid \Gamma \Rightarrow \Delta}\ (ew)$	

Inspired by [14] we use as base calculus (the hypersequent version of) Maehara's calculus **LJ**' for intuitionistic logic, see [13]. This is a multiple-conclusion version of **LJ** where the intuitionistic restriction, i.e., the consequent of a sequent contains at most one formula, applies only to the right rule of $\supset$ (and $\forall$, in the first order case). The rule schemas for the hypersequent version of **LJ**' (we call this calculus **HLJ'**) are depicted in Table 2. Note that $\Gamma, \Sigma, \Pi, \Delta$ stand for multisets of formulas while G and H denote hypersequents.

The calculus $\mathbf{HBd_2}$ is obtained by extending **HLJ'** with the following rule:

$$\frac{G \ \mid \ \Gamma', \Gamma \Rightarrow \Delta' \qquad G \ \mid \ \Gamma, \varphi \Rightarrow \psi, \Delta}{G \ \mid \ \Gamma' \Rightarrow \Delta' \ \mid \ \Gamma \Rightarrow \varphi \supset \psi, \Delta}\ (bd_2)^*$$

Remark 1. A careful application of the transformation steps of the procedure in [6] to the axiom $\xi \vee (\xi \supset (\varphi \vee (\varphi \supset \psi)))$ yields a similar rule (we call it *(bd_2)'*) with ψ not occurring in the premise. Indeed by using the invertible rules of **HLJ'** ($(\supset, r)$ is when $\Delta = \Gamma = \emptyset$) from $G \ \mid \Rightarrow \xi \ \mid \Rightarrow \xi \supset \varphi \vee (\varphi \supset \psi)$ we get

$$G \ \mid \Rightarrow \xi \ \mid \ \xi \Rightarrow \varphi, \varphi \supset \psi$$

which is easily seen to be inter-derivable in **HLJ'** with the following rule:

$$\frac{G \ \mid \ \Gamma', \xi \Rightarrow \Delta' \qquad G \ \mid \ \Gamma \Rightarrow \xi \qquad G \ \mid \ \Gamma, \varphi \Rightarrow \Delta}{G \ \mid \ \Gamma' \Rightarrow \Delta' \ \mid \ \Gamma \Rightarrow \varphi \supset \psi, \Delta}$$

The rule *(bd_2)'* is then obtained by applying cut to the premises $G \ \mid \ \Gamma', \xi \Rightarrow \Delta'$ and $G \ \mid \ \Gamma \Rightarrow \xi$. However *$(bd_2)$' does not* preserve cut elimination when added to **HLJ'**: e.g., $\Rightarrow \alpha \mid \alpha \Rightarrow \beta, \alpha \supset ((\alpha \supset \beta) \supset \delta)$ can be proved with a cut on $\Rightarrow \alpha \mid \alpha \Rightarrow \beta, \sim \beta$ and $\sim \beta \Rightarrow \alpha \supset ((\alpha \supset \beta) \supset \delta)$ but it has no cut-free proof. The rule $(bd_2)^*$ was obtained by a last heuristic step: by inspecting the counterexample for cut admissibility and changing the rule *(bd_2)'* accordingly.

We show that $\mathbf{HBd}_2$ is sound and complete for the logic Bd_2.

Definition 3. *A hypersequent $G := \Gamma_1 \Rightarrow \Delta_1 \mid \cdots \mid \Gamma_n \Rightarrow \Delta_n$ is interpreted as: $G^I := (\bigwedge \Gamma_1 \supset \bigvee \Delta_1) \vee \cdots \vee (\bigwedge \Gamma_n \supset \bigvee \Delta_n)$ where $\bigwedge \Gamma_i$ is the conjunction & of the formulas in Γ_i ($\top$ when Γ_i is empty), and $\bigvee \Delta_i$ is the disjunction of the formulas in Δ_i ($\bot$ when Δ_i is empty).*

The *height* $|d|$ of a derivation d is the maximal number of inference rules + 1 occurring on any branch of d. The *principal formula* of a logical rule is the compound formula introduced in the conclusion. Formulas, which remain unchanged by a rule application, are referred to as *contexts*. Henceforth we use $\vdash_S \varphi$ (or $\vdash_S \Gamma \Rightarrow \Delta$, or $\vdash_S G$) to denote that a formula φ (a sequent $\Gamma \Rightarrow \Delta$, or a hypersequent G) is derivable in the calculus S.

Theorem 2 (Soundness and Completeness). *For any sequent $\Gamma \Rightarrow \Delta$*

$$\vdash_{\mathbf{HBd}_2} \Gamma \Rightarrow \Delta \quad \textit{iff} \quad \vdash_{\mathbf{LJ}+(bd_2)} \Gamma \Rightarrow \Delta$$

Proof. "$\Rightarrow$": We show for any hypersequent G, if $\vdash_{\mathbf{HBd}_2} G$ then $\vdash_{\mathbf{LJ}+(bd_2)} G^I$. By induction on the height of a derivation of G. The base case (G is an initial sequent) is easy. For the inductive case it suffices to see that for each inference rule in $\mathbf{HBd}_2$ with premise(s) G_1 (and G_2), the sequent $G_1^I \Rightarrow G^I$ ($G_1^I, G_2^I \Rightarrow G^I$) is derivable in $\mathbf{LJ}+(bd_2)$. The only non-trivial case to show is $(bd_2)^*$:
$\vdash_{\mathbf{LJ}+(bd_2)} (G \mid \Gamma', \Gamma \Rightarrow \Delta')^I, (G \mid \Gamma, \varphi \Rightarrow \psi, \Delta)^I \Rightarrow (G \mid \Gamma' \Rightarrow \Delta' \mid \Gamma \Rightarrow \varphi \supset \psi, \Delta)^I$
that follows by a (cut) with the axiom (bd_2), i.e., $\Rightarrow \bigwedge \Gamma \vee (\bigwedge \Gamma \supset (\varphi \vee (\varphi \supset \psi)))$.
"$\Leftarrow$": The rules of $\mathbf{LJ}$ are derivable in $\mathbf{HBd}_2$. A proof of the axiom (bd_2) is:

$$
\begin{array}{cl}
\varphi \Rightarrow \varphi \qquad \varphi, \psi \Rightarrow \psi, \xi & \\
\hline
\Rightarrow \varphi \mid \varphi \Rightarrow \psi, \psi \supset \xi & (bd_2)^* \\
\hline
\Rightarrow \varphi \mid \varphi \Rightarrow \psi \vee (\psi \supset \xi) & (\vee, r) \\
\hline
\Rightarrow \varphi \mid \Rightarrow \varphi, \varphi \supset (\psi \vee (\psi \supset \xi)) & (\supset, r) \\
\hline\hline
\Rightarrow \varphi \vee (\varphi \supset (\psi \vee (\psi \supset \xi))) \mid \Rightarrow \varphi \vee (\varphi \supset (\psi \vee (\psi \supset \xi))) & (\vee, r), (w, r) \\
\hline
\Rightarrow \varphi \vee (\varphi \supset (\psi \vee (\psi \supset \xi))) & (ec)
\end{array}
$$

The cut elimination proof for the calculus $\mathbf{HBd}_2$ requires a special strategy. It proceeds by cases according to the cut formula. For non-atomic cut formulas having & and $\vee$ as outermost connective, we use the invertibility of the rules to replace the cut by smaller ones.

Cut formulas having $\supset$ as outermost connective require a different handling. In this case we proceed by shifting the cut upwards in a specific order: First we move the cut upwards in the right derivation d_r which has the cut formula on the right side of the sequent (Lemma 2). If the cut formula is introduced by $(\supset, r)$ or $(bd_2)^*$ we proceed by shifting the cut upwards in the left derivation d_l until the cut formula is introduced and finally cut the premises to replace the cut by smaller ones (Lemma 1). Moving the cut upwards can indeed be problematic in presence of $(\supset, r)$ or $(bd_2)^*$ in d_l. E.g., in the following situation:

$$\dfrac{\begin{array}{c}\vdots d_r\\ G \mid \Gamma' \Rightarrow \Delta' \mid \Gamma \Rightarrow \alpha \supset \beta, \Delta\end{array} \qquad \dfrac{\begin{array}{c}\vdots d_l\\ H \mid \Sigma, \alpha \supset \beta, \varphi \Rightarrow \psi\end{array}}{H \mid \Sigma, \alpha \supset \beta \Rightarrow \varphi \supset \psi, \Pi}\,(\supset, r)}{G \mid H \mid \Gamma' \Rightarrow \Delta' \mid \Gamma, \Sigma \Rightarrow \Delta, \varphi \supset \psi, \Pi}\,(cut)$$

The reason being the presence of the context Δ that does not permit the subsequent application of $(\supset, r)$ to the following derivation

$$\dfrac{\begin{array}{c}\vdots d_r\\ G \mid \Gamma' \Rightarrow \Delta' \mid \Gamma \Rightarrow \alpha \supset \beta, \Delta\end{array} \qquad \begin{array}{c}\vdots d_l\\ H \mid \Sigma, \alpha \supset \beta, \varphi \Rightarrow \psi\end{array}}{G \mid \Gamma' \Rightarrow \Delta' \mid H \mid \Gamma, \Sigma, \varphi \Rightarrow \Delta, \psi}\,(cut)$$

However, it is always possible to shift the cut upward over d_l when the cut formula in the right premise is introduced by a rule $(\supset, r)$ or $(bd_2)^*$. For instance, in the above case assume that $G \mid \Gamma' \Rightarrow \Delta' \mid \Gamma \Rightarrow \alpha \supset \beta, \Delta$ is the conclusion of a $(bd_2)^*$ rule whose premises are

$$d'_r \vdash G \mid \Gamma', \Gamma \Rightarrow \Delta' \quad \text{and} \quad d''_r \vdash G \mid \Gamma, \alpha \Rightarrow \beta, \Delta$$

The original cut above is shifted upwards as follows (we omit the contexts G and H for simplicity):

$$\dfrac{\begin{array}{c}\vdots d'_r\\ \Gamma', \Gamma \Rightarrow \Delta'\\ \hline\hline \Gamma' \Rightarrow \Delta' \mid \Gamma', \Gamma, \Sigma \Rightarrow \Delta'\end{array} \qquad \dfrac{\begin{array}{c}\vdots d_r\\ \Gamma' \Rightarrow \Delta' \mid \Gamma \Rightarrow \alpha \supset \beta, \Delta\end{array} \qquad \begin{array}{c}\vdots d_l\\ \Sigma, \alpha \supset \beta, \varphi \Rightarrow \psi\end{array}}{\Gamma' \Rightarrow \Delta' \mid \Gamma, \Sigma, \varphi \Rightarrow \Delta, \psi}\,(cut)}{\Gamma' \Rightarrow \Delta' \mid \Gamma' \Rightarrow \Delta' \mid \Gamma, \Sigma \Rightarrow \Delta, \varphi \supset \psi}\,(bd_2)^*$$

Definition 4. *The* complexity $|\varphi|$ *of a formula* φ *is defined as usual:* $|\varphi| = 0$ *if* φ *is atomic,* $|\varphi \& \psi| = |\varphi \vee \psi| = |\varphi \supset \psi| = max(|\varphi|, |\psi|) + 1$*. The* cut-rank $\rho(d)$ *of a derivation* d *is the maximal complexity of cut formulas in* $d + 1$*. (*$\rho(d) = 0$ *if* d *is cut-free).*

We use the following notation where φ is a formula and Σ is a multiset of formulas for $n \geq 0$: $\varphi^n = \overbrace{\{\varphi, \ldots, \varphi\}}^{n}$ *and* $\Sigma^n = \overbrace{\Sigma \cup \ldots \cup \Sigma}^{n}$

Lemma 1 (Shift Left and Reduction of $\supset$). *Let* d_l *and* d_r *be derivations in* **HBd**$_2$ *such that:*

- *d_l is a derivation of* $H \mid \Sigma_1, (\alpha \supset \beta)^{n_1} \Rightarrow \Pi_1 \mid \cdots \mid \Sigma_k, (\alpha \supset \beta)^{n_k} \Rightarrow \Pi_k$*,*
- *d_r is a derivation of* $G \mid \Gamma \Rightarrow \alpha \supset \beta, \Delta$*,*
- $\rho(d_l) \leq |\alpha \supset \beta|$ *and* $\rho(d_r) \leq |\alpha \supset \beta|$*,*
- *d_r ends with an application of* $(\supset, r)$ *or* $(bd_2)^*$ *introducing* $\alpha \supset \beta$*.*

Then we can find a derivation d *of* $G \mid H \mid \Gamma^{n_1}, \Sigma_1 \Rightarrow \Delta^{n_1}, \Pi_1 \mid \cdots \mid \Gamma^{n_k}, \Sigma_k \Rightarrow \Delta^{n_k}, \Pi_k$ *in* ***HBd***$_2$ *with* $\rho(d) \leq |\alpha \supset \beta|$*.*

Proof. By induction on $|d_l|$. If $|d_l|$ ends in an axiom, we are done. Otherwise, consider the last inference rule (R) applied in $|d_l|$. Suppose that (R) acts only on H, or (R) is any rule other than $(\supset, l)$ introducing $\alpha \supset \beta$, $(\supset, r)$, or $(bd_2)^*$. Then the claim follows by applications of the inductive hypothesis, (R) and, if needed, weakening and contraction. When $(R) = (\supset, l)$ and $\alpha \supset \beta$ is the principal formula the claim follows by applying the inductive hypothesis and subsequent cuts with cut formulas α and β.

The only interesting cases arise when (R) is $(\supset, r)$ or $(bd_2)^*$.When d_r ends in an application of $(\supset, r)$, the required derivation is simply obtained by applying the inductive hypothesis and (R) (note that in this case Δ is empty and hence no context is added to the premises by the inductive hypothesis).

If d_r ends with $(bd_2)^*$ and $(R) = (\supset, r)$ the case is handled as described on the previous page. Assume that d_r ends with $(bd_2)^*$ and $(R) = (bd_2)^*$ as in the following derivation (we omit the contexts for simplicity):

$$\dfrac{\begin{matrix}\vdots\, d_r \\ \Gamma' \Rightarrow \Delta' \mid \Gamma \Rightarrow \alpha \supset \beta, \Delta\end{matrix} \qquad \dfrac{\begin{matrix}\vdots\, d_l' \\ \Sigma', \Sigma_1, (\alpha \supset \beta)^{n_1} \Rightarrow \Pi'\end{matrix} \quad \begin{matrix}\vdots\, d_l'' \\ \Sigma_1, (\alpha \supset \beta)^{n_1}, \varphi \Rightarrow \psi, \Pi_1\end{matrix}}{\Sigma' \Rightarrow \Pi' \mid \Sigma_1, (\alpha \supset \beta)^{n_1} \Rightarrow \varphi \supset \psi, \Pi_1}\,(bd_2)^*}{\Gamma' \Rightarrow \Delta' \mid \Sigma' \Rightarrow \Pi' \mid \Gamma^{n_1}, \Sigma_1 \Rightarrow \varphi \supset \psi, \Pi_1, \Delta^{n_1}}\,(cut)$$

where $\Gamma' \Rightarrow \Delta' | \Gamma \Rightarrow \alpha \supset \beta, \Delta$ is the conclusion of a $(bd_2)^*$ rule with premises

$$d_r' \vdash \Gamma', \Gamma \Rightarrow \Delta' \quad \text{and} \quad d_r'' \vdash \Gamma, \alpha \Rightarrow \beta, \Delta$$

The cut is moved upwards as follows:

$$\dfrac{\dfrac{\begin{matrix}\vdots d_r' \\ \Gamma', \Gamma \Rightarrow \Delta'\end{matrix}}{\Gamma', \Gamma^{n_1}, \Sigma_1 \Rightarrow \Delta'} \qquad \dfrac{\begin{matrix}\vdots d_r \\ \Gamma' \Rightarrow \Delta' \mid \Gamma \Rightarrow \alpha \supset \beta, \Delta\end{matrix} \quad \begin{matrix}\vdots d_l'' \\ \Sigma_1, (\alpha \supset \beta)^{n_1}, \varphi \Rightarrow \psi, \Pi_1\end{matrix}}{\Gamma' \Rightarrow \Delta' \mid \Gamma^{n_1}, \Sigma_1, \varphi \Rightarrow \psi, \Delta^{n_1}, \Pi_1}\,(cut)}{\Gamma' \Rightarrow \Delta' \mid \Gamma' \Rightarrow \Delta' \mid \Gamma^{n_1}, \Sigma_1 \Rightarrow \varphi \supset \psi, \Pi_1, \Delta^{n_1}}\,(bd_2)^*$$

Lemma 2 (Shift Right). *Let d_l and d_r be derivations in* $\mathbf{HBd}_2$ *such that:*

- *d_l is a derivation of $H \mid \Sigma, \varphi \Rightarrow \Pi$,*
- *φ is either atomic or of the form $\alpha \supset \beta$,*
- *d_r is a derivation of $G \mid \Gamma_1 \Rightarrow \varphi^{n_1}, \Delta_1 \mid \cdots \mid \Gamma_k \Rightarrow \varphi^{n_k}, \Delta_k$,*
- *$\rho(d_l) \leq |\varphi|$ and $\rho(d_r) \leq |\varphi|$.*

Then we can find a derivation d of $G \mid H \mid \Gamma_1, \Sigma^{n_1} \Rightarrow \Delta_1, \Pi^{n_1} \mid \cdots \mid \Gamma_k, \Sigma^{n_k} \Rightarrow \Delta_k, \Pi^{n_k}$ in $\boldsymbol{HBd}_2$ *with $\rho(d) \leq |\varphi|$.*

Proof. By induction on $|d_r|$. If $|d_r|$ ends in an axiom, we are done. Otherwise, consider the last inference rule (R) in $|d_r|$. If (R) acts only on G or (R) is any rule other than a logical rule introducing φ then the claim follows by applications of the inductive hypothesis, (R) and, if needed, weakening or contraction. If (R) is $(\supset, r)$ or $(bd_2)^*$ and φ is the principal formula. The claim follows by applications of the inductive hypothesis, the corresponding rule (R) and Lemma 1.

Theorem 3 (Cut elimination). *Cut elimination holds for* $\mathbf{HBd}_2$.

Proof. Let d be a derivation in $\mathbf{HBd}_2$ with $\rho(d) > 0$. The proof proceeds by a double induction on $\langle \rho(d), \#\rho(d) \rangle$, where $\#\rho(d)$ is the number of applications of (cut) in d with cut rank $\rho(d)$. Consider an uppermost application of (cut) in d with cut rank $\rho(d)$. Let d_l and d_r be its premises, where d_l is a derivation of $H \mid \Sigma, \varphi \Rightarrow \Pi$, and d_r is a derivation of $G \mid \Gamma \Rightarrow \varphi, \Delta$. We can find a proof of $G \mid H \mid \Gamma, \Sigma \Rightarrow \Delta, \Pi$ in which either $\rho(d)$ or $\#\rho(d)$ decreases. Indeed we distinguish the following cases according to φ:

- φ is an atomic formula or $\varphi = \alpha \supset \beta$. The claim follows by Lemma 2.
- Suppose $\varphi = \alpha \vee \beta$. Being $\vee$ an invertible connective in $\mathbf{HBd}_2$ on the left and on the right (standard proof), we can find the derivations $d'_r \vdash G \mid \Gamma \Rightarrow \alpha, \beta, \Delta$, as well as $d'_l \vdash H \mid \alpha, \Sigma \Rightarrow \Pi$ and $d''_l \vdash H \mid \beta, \Sigma \Rightarrow \Pi$. The claim follows by replacing the cut with cut formula $\alpha \vee \beta$ with cuts on α and β.
- The case $\varphi = \alpha \& \beta$ is similar since & is also invertible on both sides.

4 Labelled Calculi for Intermediate Logics

A methodology to define cut-free labelled calculi for a large class of intermediate logics is contained in [7,10]. The resulting calculi are obtained by adding to the labelled intuitionistic system **G3I** (see Table 1) new structural rules, corresponding to the peculiar frame conditions of the considered logics.

The (formulas defining) frame conditions, to which the method in [7] applies, are called *geometric formulas.* These consist of conjunctions of formulas of the form $\forall \overline{x}(P_1 \wedge \cdots \wedge P_m \rightarrow \exists \overline{y}(M_1 \vee \cdots \vee M_n))$, where $\overline{x}, \overline{y}$ are sequences of bound variables, each P_i is a relational atom, each M_j is a conjunction of relational atoms $Q_{j_1}, \ldots, Q_{j_k}$ and $\overline{y}$ does not appear in $P_1, \ldots, P_m$. If $\overline{y}$ does not appear in M_i (for all $i = 1, \ldots, n$) the resulting formula is called a *universal axiom.* As shown in [7], the rule scheme corresponding to geometric formulas has the form

$$\frac{\overline{Q}_1[z_1/y_1], P_1, \ldots, P_m, \Gamma \Rightarrow \Delta \quad \cdots \quad \overline{Q}_n[z_n/y_n], P_1, \ldots, P_m, \Gamma \Rightarrow \Delta}{P_1, \ldots, P_m, \Gamma \Rightarrow \Delta}\ (geom)$$

where each $\overline{Q_j}$ is the multiset of $Q_{j_1}, \ldots, Q_{j_k}$ and $z_1, \ldots, z_n$ are *eigenvariables* (i.e. variables not occurring in the conclusion). The accessibility relation $\leqslant$ in all intermediate logics of Example 1 is characterized by universal or geometric axioms.

Theorem 4 ([7]). *Cut is admissible in any extension of* **G3I** *by rules of the form* $(geom)$. *Weakening and contraction are height-preserving (hp-) admissible, i.e. whenever their premises are derivable, so is their conclusion with at most the same derivation height. All rules are hp-invertible.*

Henceforth we will use $P, Q, \ldots$ (possibly indexed) to indicate relational atoms and $A, B, C, \ldots$ (possibly indexed) for compound formulas.

4.1 Towards a Systematic Approach

Inspired by the algorithms in [5,6] for hypersequent calculi, we provide a systematic method to transform a large class of frame conditions for intermediate logics into labelled rules. Soundness, completeness and cut-elimination are proved for the generated calculi, that in the case of geometric formulas coincide with those introduced in [7].

We classify the frame conditions characterizing intermediate logics into a hierarchy which intuitively accounts for the difficulty to deal proof theoretically with the corresponding formulas of first-order classical logic. As for the substructural hierarchy in [5] (see Section 3) the classification is based on the invertibility of the logical/quantifier rules of the base calculus, which in our case is **LK'**, i.e., a variant of Gentzen **LK** calculus for first-order classical logic in which all logical rules are invertible, while the universal (existential) quantifier is invertible on the right (respectively on the left). W.l.o.g. we will consider formulas in prenex form. The class to which a formula belongs is determined by the alternation of universal and existential quantifiers in the prefix. The resulting classification is essentially the arithmetical hierarchy.

Definition 5. *The classes Π_k and Σ_k are defined as follows: $A \in \Sigma_0$ and $A \in \Pi_0$, if A is quantifier-free. Otherwise:*

– *if A is classically equivalent to $\exists\overline{x}B$ where $B \in \Pi_n$ then $A \in \Sigma_{n+1}$*
– *if A is classically equivalent to $\forall\overline{x}B$ where $B \in \Sigma_n$ then $A \in \Pi_{n+1}$*

Example 3. Universal axioms are in Π_1, while geometric formulas are in Π_2.

We show below how to transform all formulas within the class Π_2 into structural labelled rules that preserve cut-elimination once added to (a slightly modified version of) **G3I**. The resulting rules are *equivalent* to the corresponding axioms, that is, **LK'** extended with the defined rules or **LK'** extended with the original formula proves the same sequents.

As for the algorithm in [5,6] (see Remark 1), the key ingredients for our transformation are: (1) the invertibility in **LK'** of the rules $R\forall$ (i.e. introduction of $\forall$ on the right) and $L\exists$ (i.e. introduction of $\exists$ on the left) and of all logical rules; (2) the following lemma that allows formulas to change the side of the (labelled) sequent going from the conclusion to the premises.

Lemma 3 ([5]). *The sequent $A_1, \ldots, A_n \Rightarrow B_1, \ldots, B_m$ is equivalent to the rule*

$$\frac{B_1, \Gamma \Rightarrow \Delta \quad \cdots \quad B_m, \Gamma \Rightarrow \Delta}{A_1, \ldots, A_n, \Gamma \Rightarrow \Delta}$$

where Γ, Δ are fresh metavariables standing for multisets of formulas.

Proof. "$\Rightarrow$": Follows by m applications of CUT (and weakening). "$\Leftarrow$": Follows by instantiating $\Gamma = \emptyset$ and $\Delta = B_1, \ldots, B_m$.

Theorem 5. *Every frame condition $\mathcal{F}$ within the class Π_2 can be transformed into a set of equivalent structural rules in labelled calculi.*

Proof. Let $\mathcal{F} = \forall\overline{x}\exists\overline{y}A$, where A is a quantifier-free formula, $\overline{x} = x_1, \dots, x_h$ and $\overline{y} = y_1, \dots, y_l$. W.l.o.g. we assume that A is in disjunctive normal form and has the shape $B_1 \vee \dots \vee B_k$ where every B_i has the form $Q_{i_1} \wedge \dots \wedge Q_{i_n} \wedge \neg P_{i_1} \wedge \dots \wedge \neg P_{i_m}$. By the invertibility of the rule $R\forall$, $\Rightarrow \mathcal{F}$ is equivalent to $\Rightarrow \exists\overline{y}A'$, where A' is obtained by replacing in A all $x_1, \dots, x_h$ with fresh variables $x'_1, \dots, x'_h$ (*eigenvariable* condition). We distinguish two cases according to whether $\mathcal{F}$ contains at least one existential quantifier ($\mathcal{F} \in \Pi_2$) or it does not ($\mathcal{F} \in \Pi_1$).

Assume that $l = 0$ ($\mathcal{F} \in \Pi_1$). By the invertibility of $R\vee$, $R\wedge$ and $R\neg$, $\Rightarrow A'$ is equivalent to a set of *atomic* sequents $\overline{P} \Rightarrow \overline{Q}$ with $\overline{P}, \overline{Q}$ multisets of relational atoms P_{i_r}, Q_{i_s}. By Lemma 3, these sequents are equivalent to rules of the form

$$\frac{\overline{Q}, \Gamma \Rightarrow \Delta}{\overline{P}, \Gamma \Rightarrow \Delta}\ (\Pi'_1)$$

Assume that $l > 0$ ($\mathcal{F} \in \Pi_2$). By Lemma 3, $\Rightarrow \exists\overline{y}A'$ is equivalent to $\dfrac{\exists\overline{y}A', \Gamma \Rightarrow \Delta}{\Gamma \Rightarrow \Delta}$ which is in turn equivalent to $\dfrac{A'', \Gamma \Rightarrow \Delta}{\Gamma \Rightarrow \Delta}$ where A'' is obtained by replacing in A all $y_1, \dots, y_l$ with fresh variables $y'_1, \dots, y'_l$ (*eigenvariable* condition). By the invertibility of $L\vee, L\wedge$ and $L\neg$ we get

$$\frac{\{Q_{i_1}, \dots, Q_{i_n}, \Gamma \Rightarrow \Delta, P_{i_1}, \dots, P_{i_m}\}_{i=1\dots k}}{\Gamma \Rightarrow \Delta}\ (\Pi_2)$$

The resulting rules are equivalent to $\mathcal{F}$.

Remark 2. The (Π_2) rule (which is, in fact, a rule schema) is invertible. To make (Π'_1) invertible we simply repeat $\overline{P}$ in its premises, thus obtaining

$$\frac{\overline{P}, \overline{Q}, \Gamma \Rightarrow \Delta}{\overline{P}, \Gamma \Rightarrow \Delta}\ (\Pi_1)$$

which is interderivable with the rule (Π'_1) in **LK'**.

Observe that while (Π_1) coincides with the rule defined in [7] for universal axioms, this is not the case for geometric formulas. Indeed the above procedure applied to a geometric formula generates a rule of the form (Π_2) which might contain relational atoms ($P_{i_1}, \dots, P_{i_m}$) on the right hand side of premises and is therefore not of the form (*geom*) [7]. Being geometric formulas Π_2 formulas of a *particular* shape, we show below that the (Π_2) rules for them (generated by Th. 5) can be easily transformed into rules with no relational atom on the right hand side; the resulting rules are nothing but the (*geom*) rules in [7].

Corollary 1. *Geometric axioms are equivalent to rules of the form (geom).*

Proof. Geometric axioms are formulas in Π_2 of the form $\forall\overline{x}\exists\overline{y}A_G$, where A_G is $B_1 \vee \dots \vee B_n \vee C_1 \vee \dots \vee C_m$ where each B_i is $Q_{i_1} \wedge \dots \wedge Q_{i_k}$ and each C_j is $\neg P_j$. Theorem 5 transforms such an axiom into the equivalent rule

$$\frac{\{Q_{i_1}, \dots, Q_{i_k}, \Gamma \Rightarrow \Delta\}_{i=1\dots n} \quad \{\Gamma \Rightarrow \Delta, P_j\}_{j=1\dots m}}{\Gamma \Rightarrow \Delta}\ (\Pi'_2)$$

The claim follows by showing that (Π_2') can be transformed into a rule

$$\frac{\overline{Q}_1, P_1, \ldots, P_m, \Gamma \Rightarrow \Delta \quad \cdots \quad \overline{Q}_n, P_1, \ldots, P_m, \Gamma \Rightarrow \Delta}{P_1, \ldots, P_m, \Gamma \Rightarrow \Delta} \; (\Pi_2^G)$$

where each $\overline{Q}_i$ is a multiset of $Q_{i_1}, \ldots, Q_{i_k}$. Observe that (Π_2^G) is nothing but a (*geom*) rule [7]. To derive (Π_2^G) we use (Π_2') and m initial sequents:

$$\frac{\{\overline{Q}_i, P_1, \ldots, P_m, \Gamma \Rightarrow \Delta\}_{i=1\ldots n} \quad \{P_1, \ldots, P_m, \Gamma \Rightarrow \Delta, P_j\}_{j=1\ldots m}}{P_1, \ldots, P_m, \Gamma \Rightarrow \Delta} \; (\Pi_2')$$

To derive (Π_2') we first apply (Π_2^G) followed by m applications of CUT.

Rules for non-geometric Π_2 formulas manipulate relational atoms in both sides of the sequent. We show below that this is not an obstacle for obtaining admissibility results analogous to those in Theorem 4. The base calculus we will work with is a slightly modified version of **G3I** which is obtained by adding initial sequents of the form $x \leqslant y, \Gamma \Rightarrow \Delta, x \leqslant y$ to **G3I**. Note that these sequents, which are needed for our completeness proof (Theorem 6), were first introduced for **G3I** and later removed as they were not needed in the labelled systems for intermediate logics presented in [7]; the reason being that in these systems no rule contains atoms $x \leqslant y$ in the succedent.

Henceforth we denote by **G3SI*** (super-intuitionistic) the system obtained by adding to our base calculus rules of the form (Π_1) and (Π_2) defined by applying Theorem 5 to the set $*$ of formulas within the class Π_2.

Consider the following version of the structural rules for contraction and weakening (Z is either a labelled formula $u : \varphi$ or a relational atom $x \leqslant y$):

Table 3. Structural rules

$$\frac{\Gamma \Rightarrow \Delta}{Z, \Gamma \Rightarrow \Delta}\,\text{L-W} \qquad \frac{\Gamma \Rightarrow \Delta}{\Gamma \Rightarrow \Delta, Z}\,\text{R-W} \qquad \frac{Z, Z, \Gamma \Rightarrow \Delta}{Z, \Gamma \Rightarrow \Delta}\,\text{L-C} \qquad \frac{\Gamma \Rightarrow \Delta, Z, Z}{\Gamma \Rightarrow \Delta, Z}\,\text{R-C}$$

Lemma 4. *In* **G3SI*** *we have:*

1. Substitution of variables is hp-admissible; *2. Weakening is hp-admissible;*
3. All the rules are hp-invertible; *4. Contraction is hp-admissible.*

Proof. 1. We need to show that if y is free for x in every formula in $\Gamma \Rightarrow \Delta$ and $\Gamma \Rightarrow \Delta$ is derivable in **G3SI*** then so is $\Gamma[y/x] \Rightarrow \Delta[y/x]$ with the same derivation height. The proof is by induction on the height of the derivation of $\Gamma \Rightarrow \Delta$ and follows mostly the proof of the same theorem in [7].

2. The proof follows the pattern of [7] with the new case of R-W for $x \leqslant y$. Also in this case the proof is by induction on the premise of weakening. When $\Gamma \Rightarrow \Delta$ is concluded by a rule R with variable condition, i.e. by $R \supset$ or (Π_2), we first might need to replace the *eigenvariable* of the rule with a new one. For instance, if R is $R \supset$ and the premise of weakening is a sequent of the form

$x \leqslant y, x : \varphi, \Gamma \Rightarrow \Delta', y : \psi$, we first replace y with a new z (Lemma 4.1) and obtain $x \leqslant z, x : \varphi, \Gamma \Rightarrow \Delta', z : \psi$; now by the inductive hypothesis $x \leqslant z, x : \varphi, \Gamma \Rightarrow \Delta', z : \psi, x \leqslant y$; the conclusion follows by $R \supset$.

3. Observe that $L \supset$ and the rules (Π_1) and (Π_2) are hp-invertible since their premises are obtained from the conclusion by applying weakening which is hp-admissible. For the other rules the proof is as in [7].

4. Similar to the case of weakening.

The proof of soundness and completeness of our calculi follows the same pattern of the analogous proof in [11] and it is sketched below. Let $\mathfrak{F}_{SI^*} = \langle W, \leqslant \rangle$ be a frame with the properties of the accessibility relation expressed as (Π_2 and Π_1) formulas in $*$. Let $L = \{x, y, z \dots\}$ be the labels occurring in a **G3SI***-derivation. An *interpretation* I of L in $\mathfrak{F}_{SI^*}$ is a function $I : L \to W$.

Definition 6. *Let $\mathfrak{M}_{SI^*} = \langle \mathfrak{F}_{SI^*}, \Vdash \rangle$ be a model and I an interpretation. A labelled sequent $\Gamma \Rightarrow \Delta$ is* valid *in $\mathfrak{M}_{SI^*}$ if for every interpretation I we have: if for all labelled formulas $x : \varphi$ and relational atoms $y \leqslant z$ in Γ, $x^I \Vdash \varphi$ and $y^I \leqslant z^I$ hold, then for some $w : \psi$, $u \leqslant v$ in Δ we have $w^I \Vdash \psi$ or $u^I \leqslant v^I$. A sequent $\Gamma \Rightarrow \Delta$ is* valid *in a frame $\mathfrak{F}_{SI^*}$ when it is valid in every model $\mathfrak{M}_{SI^*}$.*

Theorem 6 (Soundness and Completeness). *For any sequent $\Gamma \Rightarrow \Delta$*

$$\vdash_{\mathbf{G3SI}^*} \Gamma \Rightarrow \Delta \textit{ iff } \Gamma \Rightarrow \Delta \textit{ is valid in every frame } \mathfrak{F}_{SI^*}.$$

Proof. "$\Rightarrow$": By induction on the height of a derivation of $\Gamma \Rightarrow \Delta$ in **G3SI***. The claim is straightforward if $\Gamma \Rightarrow \Delta$ is initial (notice the new case of sequents of the form $x \leqslant y, \Gamma' \Rightarrow \Delta', x \leqslant y$ that are clearly valid). The cases of the rules for **G3I** are as in [11] with $R \supset$ similar to the case $R\Box$, while (Π_2) is handled as the mathematical rules there with *eigenvariable.*

"$\Leftarrow$": We show that each sequent $\Gamma \Rightarrow \Delta$ is either derivable in **G3SI*** or it has a countermodel in a frame with properties expressed by formulas in $*$. We first construct in the usual manner a derivation tree for $\Gamma \Rightarrow \Delta$ by applying the rules of **G3SI*** root first. If the reduction tree is finite, i.e., all leaves are initial or conclusions of $L\bot$, we have a proof in **G3SI***. Assume that the derivation tree is infinite. By König's lemma, it has an infinite branch that is used to build the needed counterexample. Let $\Gamma \Rightarrow \Delta = \Gamma_0 \Rightarrow \Delta_0, \Gamma_1 \Rightarrow \Delta_1, \dots, \Gamma_i \Rightarrow \Delta_i, \dots$ be one such branch. Consider the sets $\mathbf{\Gamma} \equiv \bigcup \Gamma_i$ and $\mathbf{\Delta} \equiv \bigcup \Delta_i$ for $i \geqslant 0$. We now construct a countermodel, i.e. a model that makes all labelled formulas and relational atoms in $\mathbf{\Gamma}$ true and all labelled formulas and relational atoms in $\mathbf{\Delta}$ false. Let $\mathfrak{F}_{SI^*}$ be a frame, whose elements are all the labels occurring in $\mathbf{\Gamma}, \mathbf{\Delta}$. $\mathfrak{F}_{SI^*}$ is defined as follows: (i) for all $x : p$ in $\mathbf{\Gamma}$ it holds that $x^I \Vdash p$ in $\mathfrak{F}_{SI^*}$; (ii) for all $x \leqslant y$ in $\mathbf{\Gamma}$ we have $x^I \leqslant y^I$ in $\mathfrak{F}_{SI^*}$; (iii) for all $x' : p'$ in $\mathbf{\Delta}$ we have $x'^I \nVdash p'$ in $\mathfrak{F}_{SI^*}$; finally (iv) for all $x' \leqslant y'$ in $\mathbf{\Delta}$ it holds $x'^I \nleqslant y'^I$ in $\mathfrak{F}_{SI^*}$. $\mathfrak{F}_{SI^*}$ is well defined as it is not the case that either $x \leqslant y$ is in Γ_i and $x \leqslant y$ is in Δ_j or $x \leqslant y, x : p$ is in Γ_i and $y : p$ is in Δ_j (for any i and j), as otherwise we would have an initial sequent and therefore the branch would be finite, against the hypothesis. We then show that for any formula φ, φ is forced at x^I if $x : \varphi$ is in $\mathbf{\Gamma}$ and φ is not forced at x^I if $x : \varphi$ is in $\mathbf{\Delta}$. As all relational atoms in $\mathbf{\Gamma}$

are true and those in $\mathbf{\Delta}$ are false by definition of $\mathfrak{F}_{SI^*}$ we have a countermodel to $\Gamma \Rightarrow \Delta$. By induction on the formula φ.

If φ is $\bot$, it cannot be in $\mathbf{\Gamma}$ because no sequent in the branch contains $x : \bot$ in the antecedent, so it is not forced at any node of the model. If φ is an atom p in $\mathbf{\Gamma}$ then $x^I \Vdash p$ by definition; and $x^I \nVdash p$ if it is in Δ.

If $x : \varphi \& \psi$ is in $\mathbf{\Gamma}$, there exists i such that $x : \varphi \& \psi$ appears first in Γ_i, and therefore, for some $j \geqslant 0$, $x : \varphi$ and $x : \psi$ are in Γ_{i+j} . By inductive hypothesis, $x \Vdash \varphi$ and $x \Vdash \psi$ and therefore $x \Vdash \varphi \& \psi$ (analogous for $x : \varphi \vee \psi$ in $\mathbf{\Delta}$).

If $x : \varphi \& \psi$ is in $\mathbf{\Delta}$ then either $x : \varphi$ or $x : \psi$ is in $\mathbf{\Delta}$. By inductive hypothesis, $x \nVdash \varphi$ or $x \nVdash \psi$ and therefore $x \nVdash \varphi \& \psi$ (analogous for $x : \varphi \vee \psi$ in $\mathbf{\Gamma}$).

If $x : \varphi \supset \psi$ is in $\mathbf{\Gamma}$, we consider all the relational atoms $x \leqslant y$ that occur in $\mathbf{\Gamma}$. If there is no such atom then $x \Vdash \varphi \supset \psi$ is in the model. Else, for any occurrence of $x \leqslant y$ in $\mathbf{\Gamma}$, by construction of the tree either $y : \varphi$ is in $\mathbf{\Delta}$ or $y : \psi$ is in $\mathbf{\Gamma}$. By inductive hypothesis $y \nVdash \varphi$ or $y \Vdash \psi$, and since $x \leqslant y$ we have $x \Vdash \varphi \supset \psi$ in the model.

If $x : \varphi \supset \psi$ is in $\mathbf{\Delta}$, at next step of the reduction tree we have that $x \leqslant y$ and $y : \varphi$ are in $\mathbf{\Gamma}$, whereas $y : \psi$ is in $\mathbf{\Delta}$. By inductive hypothesis this gives $x \leqslant y$ and $y \Vdash \varphi$ but $y \nVdash \psi$, i.e. $x \nVdash \varphi \supset \psi$.

Theorem 7 (Cut elimination). *The cut rule (Z is either $u : \varphi$ or $x \leqslant y$)*

$$\frac{\Gamma \Rightarrow \Delta, Z \quad Z, \Gamma' \Rightarrow \Delta}{\Gamma, \Gamma' \Rightarrow \Delta', \Delta}\ \text{CUT}$$

can be eliminated from **G3SI****-derivations.*

Proof. We distinguish two cases according to the cut formula Z. When Z is a labelled formula $u : \varphi$, the proof has the same structure of the cut elimination proof in [7] for **G3I** extended with rules of the form (*geom*). It proceeds by a double induction on the complexity of the cut formula and on the sum of the derivation heights of the premises of cut. We observe that the additional initial sequents, i.e. $x \leqslant y, \Sigma \Rightarrow \Pi, x \leqslant y$, make no trouble as Z belongs to $\Sigma \Rightarrow \Pi$. Moreover, cuts can be permuted upward over any structural rule (Π_1) and (Π_2). To avoid clashes with the variable conditions when permuting a cut with (Π_2) (or with $R \supset$) an appropriate substitution (Lemma 4.1) is used.

When Z is a relational atom $x \leqslant y$ the proof proceeds by induction on the derivation height of the right premises of cut, i.e. $\Gamma \Rightarrow \Delta, x \leqslant y$. The base case is when this is initial; then it is either *(i)* $u \leqslant v, u : p, \Gamma'' \Rightarrow, \Delta'', v : p, x \leqslant y$; or *(ii)* $u \leqslant v, \Gamma'' \Rightarrow \Delta'', u \leqslant v, x \leqslant y$; or else *(iii)* $x \leqslant y, \Gamma'' \Rightarrow \Delta, x \leqslant y$. If *(i)* or *(ii)*, the conclusion of cut is initial. Otherwise, if *(iii)*, the conclusion of cut is obtained by weakening (Lemma 4.2). Assume that $\Gamma \Rightarrow \Delta, x \leqslant y$ is not initial and that R is the last rule applied to derive it. We reason by cases according to R and show that the cut can be shifted upwards over the premise(s) of R. The key observation is that $x \leqslant y$ is left unchanged by the application of R as no rule of **G3SI*** changes the relational atoms appearing on the right hand side of its conclusion. If R is a logical rule other than $R \supset$ or a rule following the (Π_2) scheme then cut is simply permuted upwards with R. For instance let R be (Π_1); then the derivation

$$\dfrac{\dfrac{Q_1, \ldots, Q_m, P_1, \ldots, P_n, \Gamma'' \Rightarrow \Delta, x \leqslant y}{P_1, \ldots, P_n, \Gamma'' \Rightarrow \Delta, x \leqslant y}\ \Pi_1 \qquad x \leqslant y, \Gamma' \Rightarrow \Delta'}{P_1, \ldots, P_n, \Gamma'', \Gamma' \Rightarrow \Delta', \Delta}\ \textsc{cut}$$

is transformed into

$$\dfrac{\dfrac{Q_1, \ldots, Q_m, P_1, \ldots, P_n, \Gamma'' \Rightarrow \Delta, x \leqslant y \quad x \leqslant y, \Gamma' \Rightarrow \Delta'}{Q_1, \ldots, Q_m, P_1, \ldots, P_n, \Gamma'', \Gamma' \Rightarrow \Delta', \Delta}\ \textsc{cut}}{P_1, \ldots, P_n, \Gamma'', \Gamma' \Rightarrow \Delta', \Delta}\ \Pi_1$$

If R is a rule with variable condition as $R \supset$ or a rule following the (Π_2) scheme then we need first to replace the *eigenvariable* in the premise(s) of R and then permute cut and R. Note that the permutation with a $R \supset$ rule is not problematic as the cut formula $x \leqslant y$ on the right hand side always belongs to the context of the rule (i.e., to the Δ in the rule schemas in Table 1).

Open Problems: (1) Characterize the class of axioms that can be transformed into equivalent hypersequent *logical* rules (Section 3.1 shows a particular axiom for which this is the case) and define an algorithm for the transformation. Note that when defining logical rules the cut-admissibility of the resulting calculus needs either an ad-hoc syntactic proof or suitable semantic methods as in [12].

(2) Are there intermediate logics characterized by frame conditions that are Π_2 formulas not equivalent to any geometric formula?

(3) Not all frame conditions are formulas within the class Π_2. As shown in [4], all axiomatizable intermediate logics are definable by *canonical formulas* that are in the class $\mathcal{N}_3$ of the substructural hierarchy (cf. Sec. 3). In light of this result, which is the maximum nesting of quantifiers occurring in formulas defining frame conditions for intermediate logics? How to capture all[2] these formulas?

Acknowledgment. We are grateful to Sara Negri for her suggestions and for pointing out [11] to us.

References

1. Andreoli, J.-M.: Logic programming with focusing proofs in linear logic. Journal of Logic and Computation 2(3), 297–347 (1992)
2. Avron, A.: A Constructive Analysis of RM. J. of Symb. Logic 52(4), 939–951 (1987)
3. Avron, A.: The method of hypersequents in the proof theory of propositional non-classical logic. In: Hodges, W., Hyland, M., Steinhorn, C., Truss, J. (eds.) Logic: From Foundations to Applications, pp. 1–32. Oxford University Press (1996)
4. Chagrov, A., Zakharyaschev, M.: Modal Logic. Oxford University Press (1997)
5. Ciabattoni, A., Galatos, N., Terui, K.: From axioms to analytic rules in nonclassical logics. In: Proceedings of LICS 2008, pp. 229–240. IEEE (2008)
6. Ciabattoni, A., Straßburger, L., Terui, K.: Expanding the realm of systematic proof theory. In: Grädel, E., Kahle, R. (eds.) CSL 2009. LNCS, vol. 5771, pp. 163–178. Springer, Heidelberg (2009)

[2] See [9] for a recent work in this direction.

7. Dyckhoff, R., Negri, S.: Proof analysis in intermediate logics. Archive for Mathematical Logic 51(1-2), 71–92 (2012)
8. Gabbay, D.: Labelled Deductive Systems: Foundations. Oxford Univ. Press (1996)
9. Negri, S.: Proof analysis beyond geometric theories: from rule systems to systems of rules (submitted)
10. Negri, S.: Proof analysis in non-classical logics. In: Logic Coll. 2005, pp. 107–128 (2007)
11. Negri, S.: Kripke completeness revisited. In: Primiero, G., Rahman, S. (eds.) Acts of Knowledge - History, Philosophy and Logic. College Publications (2009)
12. Lahav, O.: From Frame Properties to Hypersequent Rules in Modal Logics. In: Proceedings of LICS 2013 (2013)
13. Takeuti, G.: Proof Theory, 2nd edn. North-Holland (1987)
14. Rothenberg, R.: On the relationship between hypersequent calculi and labelled sequent calculi for intermediate logics with geometric Kripke semantics. PhD thesis (2010)
15. Simpson, A.: The Proof Theory and Semantics of Intuitionistic Modal Logic. PhD thesis (1994)
16. Viganò, L.: Labelled Non-Classical Logics. Kluwer (2000)
17. Wansing, H.: Sequent Systems for Modal Logic. In: Gabbay, D., Guenther, F. (eds.) Handbook of Philosophical Logic, pp. 61–145. Kluwer, Dordrecht (2002)

TATL: Implementation of ATL Tableau-Based Decision Procedure

Amélie David

Laboratoire IBISC - Université d'Évry Val d'Essonne - EA 4526
23 bd de France - 91037 Évry Cedex - France
adavid@ibisc.univ-evry.fr

Abstract. This paper describes the implementation of a tableau-based decision procedure for the Alternating-time Temporal Logic proposed by Goranko and Shkatov in 2009, as well as a set of representative formulas used for testing.

Keywords: Alternating-time Temporal Logic, tableaux, theorem prover.

1 Introduction

The Alternating-time Temporal Logic (ATL) was introduced by Alur, Henzinger and Kupferman in 2002 [1] in order to formally specify and verify reactive multi-agent systems. Such systems are represented by a concurrent game structure $\mathcal{C}$, in short CGS, which are state-transition graphs. Transitions enable to go from one state of the system to another depending on the choices made by every agent of the system at a given state. At a given state, the different possible transitions (which may lead to the same successor state) are represented by move vectors.

In ATL, properties of such systems are expressed with formulas following the grammar:
$F := p \mid \neg F \mid (F_1 \wedge F_2) \mid (F_1 \vee F_2) \mid (F_1 \rightarrow F_2) \mid \langle\langle A\rangle\rangle \bigcirc F \mid \langle\langle A\rangle\rangle \Box F \mid \langle\langle A\rangle\rangle \Diamond F \mid \langle\langle A\rangle\rangle F_1 \mathcal{U} F_2$,
where p is a proposition and A is a coalition, that is, a set of agents. ATL-formulas represent objectives for the agents of the system $\mathcal{C}$ or the possibility of achieving objectives. The connectors $\neg$, $\wedge$, $\vee$ and $\rightarrow$ have the same meanings as in classical propositional logic. The operators $\bigcirc$, $\Box$, $\Diamond$ and $\mathcal{U}$ are those of temporal logics and mean *next time*, *always*, *eventually* and *until*, respectively. The novelty of ATL in comparison to other temporal logics is the use of agents' coalitions and strategies for these coalitions by using the path quantifier $\langle\langle A\rangle\rangle$. $\langle\langle A\rangle\rangle F$ means: *the coalition A has a strategy to achieve F*. So, for example, the formula $\langle\langle 1,2\rangle\rangle \Box(p \wedge q) \wedge \neg\langle\langle 2\rangle\rangle \Diamond p$ means that the coalition of agents 1 and 2 has a strategy to always achieve p and q, and the "coalition" with only agent 2 does not have a strategy to eventually achieve p.

Checking whether a formula is satisfiable, that is, checking whether a model exists for the formula or not, is a common question in logic. Methods to respond to this question were introduced in 2006 by Goranko and van Drimmelen using automata [2], and then in 2009 by Goranko and Shkatov using tableaux [3]. To our knowledge, none of them has been implemented. Therefore, in this paper, we present TATL, a prototype constituting the first implementation of Goranko and Shkatov's decision procedure. The core of TATL is implemented in Ocaml. TATL is available at `http://atila.ibisc.univ-evry.fr/tableau_ATL/`.

At first, we recall the tableau-based decision procedure of Goranko and Shkatov, then we present, in Section 3, the general principles of TATL's algorithm. In Section 4 we explain how to get and use TATL, and in Section 5 how TATL had been tested.

2 Tableau-Based Decision Procedure

Goranko and Shkatov's tableau method decides the satisfiability of an ATL formula θ by constructing a representation of models for θ, if a model exists. Construction of a tableau for an ATL formula consists of two phases: first, construct a pretableau and then obtain the tableau itself. The pretableau and the tableau are state-transition graphs. The pretableau contains two kinds of vertexes (states and prestates) and two kinds of edges (unmarked and marked transitions). Prestates and states are sets of ATL-formulas and marked transitions are labeled by sets of move vectors. Prestates and unmarked transitions are technical items ensuring the termination of the procedure and will not remain in the tableau. Intuitively, a prestate is an embryo of states that, when properly saturated, generates one or more states.

The first phase of the tableau construction uses two rules, **SR** and **Next** recursively applied on new sets of prestates or states. The construction phase starts with a set containing only the input formula as a prestate. The rule **SR** allows one to obtain states from prestates and the rule **Next** to obtain prestates from states. The rule **SR** decomposes each formula of a prestate Γ into primitive literal formulas, that is, formulas of the form $\top$,p,$\neg p$,$\langle\langle A\rangle\rangle \bigcirc F$ or $\neg\langle\langle B\rangle\rangle \bigcirc F$ ($B \neq \Sigma$, where Σ is the coalition of all agents). With the primitive formulas, we get all the information needed to verify the properties of the current state of a CGS and with the primitive next-time formulas $\langle\langle A\rangle\rangle \bigcirc F$ and $\neg\langle\langle B\rangle\rangle \bigcirc F$, all the information to verify the properties of next states in the CGS. From this decomposition, we obtain the states generated by Γ. Then, from each of these states, say Δ, the rule **Next** treats all Δ's next-time formulas in order to obtain a new set of prestates, the successors of Δ, which respects all the possible strategies of agents and their objectives. So the rule **SR** creates states and unmarked transitions, whereas the rule **Next** creates prestates and marked transitions. When a state or a prestate already exists, the rules **SR** and **Next** do not generate copies and only create a transition.

When the rules **SR** and **Next** cannot be applied any further, the obtained structure is the complete pretableau of the input formula and the elimination phase can start. The first elimination rule, **PR**, eliminates all prestates after having properly interconnected states. Then we apply the elimination rules **E1**, **E2**, and **E3**. The rule **E1** eliminates all states that contain an explicit inconsistency, that is, any state containing formulas F and also $\neg F$, where F is a primitive formula. Then we apply rules **E3** and **E2** until no more states can be eliminated. The rule **E2** eliminates all states which have lost all their successors linked to the same move vector. Eventualities are formulas of the form a) $\langle\langle A\rangle\rangle\Diamond F_2$, b) $\langle\langle A\rangle\rangle F_1 \mathcal{U} F_2$ or c) $\neg\langle\langle A\rangle\rangle\Box F_3$. The rule **E3** eliminates all states not satisfying their eventualities, that is always postponing the realization of F_2 for a) and b), and the realization of F_3 for c).

At the end of the elimination phase, we obtain the tableau of the input formula. The tableau is open and the input formula is satisfiable if there remains at least one state containing the input formula; otherwise, the tableau is closed and the formula is unsatisfiable.

3 General Principles of TATL

The algorithm of TATL follows the steps described in the previous section. To represent states and prestates of the tableau and pretableau, that is, vertexes, we use a structure which allows us to stock information about the name and type of the vertex, the associated set of formulas, all the possible move vectors linked to that vertex, its successors, as well as an indicator about its consistency.

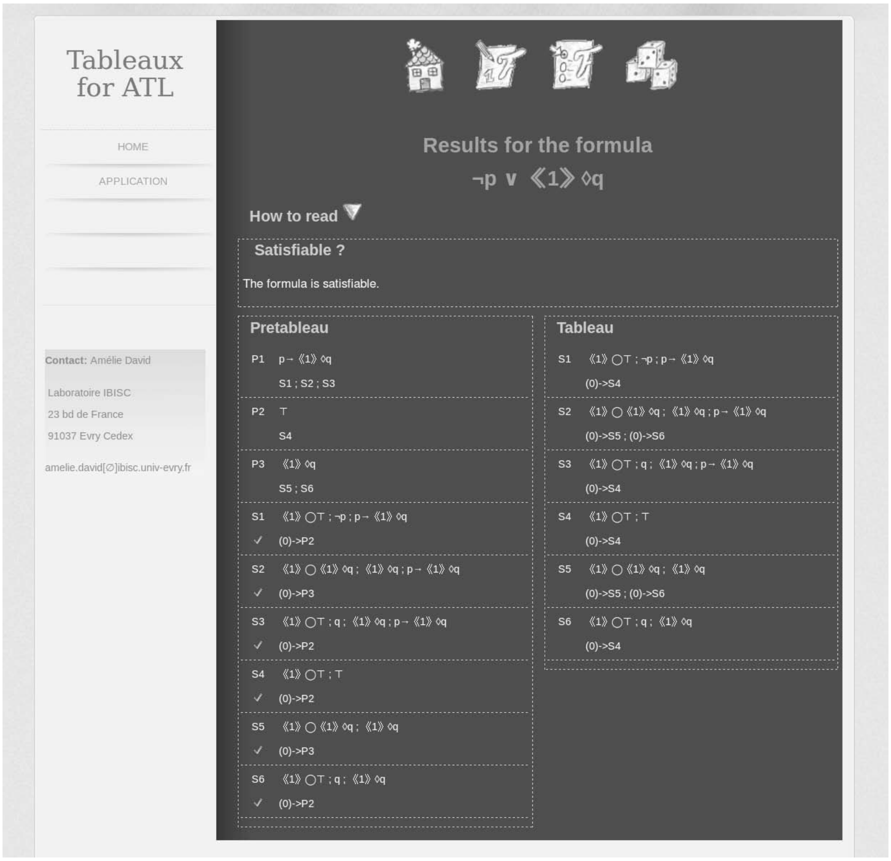
During the construction phase, we use four sets of vertexes: partial states, complete states, partial prestates and complete prestates. Partial prestates and partial states are waiting for application of the rule **SR** and the rule **Next**, respectively, to become complete prestates and complete states. The rule **SR** generates partial states and the rule **Next** generates partial prestates.

The most difficult rules to implement were the rules **SR** and **E3**, so we describe their implementation in more detail. Indeed, for the rule **SR**, we need to deal with decompositions where the operator *or* occurs. This decomposition generates several choices and therefore several successors for the same prestate. So we use a decomposition tree where all interior nodes still contain non-decomposed formulas and each leaf node contains a set of fully decomposed formulas. Each node of the tree is composed of two sets: one with decomposed formulas and primitives, and one with formulas that still need to be decomposed. For each interior node, we process one of the non-decomposed formulas to obtain successors. If a formula resulting from the decomposition is not primitive, it joins the set of non-decomposed formulas.

The rule **E3** needs, for each eventuality occurring in the tableau, to find all states containing that eventuality. This results in finding a path from a given state to a state satisfying the eventuality, avoiding looping indefinitely in a cycle. Let us call ξ the set of all states containing the eventuality. Then we separate the states of ξ which realize the eventuality from the others, using three sets: *to_be_treated*, *satisfied* and *current*. The procedure is iterative with the following halt conditions: the set *current* is empty or stable at the end of the iteration. At the beginning, all states ξ are placed in the set *to_be_treated*. For each state s in the set *to_be_treated*, we first check if the eventuality is immediately satisfied, that is, if s contains F_2 for an eventuality of the form $\langle\langle A\rangle\rangle\Diamond F_2$ or $\langle\langle A\rangle\rangle F_1 \mathcal{U} F_2$, and $\neg F_3$ for an eventuality of the form $\neg\langle\langle A\rangle\rangle\Box F_3$. In that case, we move s to the set *satisfied*, otherwise, we check for each move vector leading to successors of s in ξ whether one of these successors is also in the set *satisfied*. If the eventuality is of the form $\langle\langle A\rangle\rangle F_1 \mathcal{U} F_2$, we also check that s contains F_1. If these conditions are satisfied then the state is moved to the set *satisfied*, otherwise it is moved to the set *current*. When the set *to_be_treated* is empty, if the set *current* is also empty or contains all the states to be treated at the beginning of the iteration, we return the set *current*, which contains all the states not satisfying the eventuality, otherwise we move the states of the set *current* to the set *to_be_treated* and the procedure is repeated.

4 Description of TATL

TATL was conceived as a web application in order to be multi-platform and easy to use, but binaries are also available. TATL is an Ocaml program and



Contact: Amélie David

Laboratoire IBISC
23 bd de France
91037 Evry Cedex

amelie.david[∅]ibisc.univ-evry.fr

Results for the formula
¬p ∨ 《1》◊q

Satisfiable ?

The formula is satisfiable.

Pretableau

P1	p→ 《1》◊q	S1 ; S2 ; S3
P2	⊤	S4
P3	《1》◊q	S5 ; S6
S1	《1》○⊤ ; ¬p ; p→ 《1》◊q	✓ (0)->P2
S2	《1》○《1》◊q ; 《1》◊q ; p→ 《1》◊q	✓ (0)->P3
S3	《1》○⊤ ; q ; 《1》◊q ; p→ 《1》◊q	✓ (0)->P2
S4	《1》○⊤ ; ⊤	✓ (0)->P2
S5	《1》○《1》◊q ; 《1》◊q	✓ (0)->P3
S6	《1》○⊤ ; q ; 《1》◊q	✓ (0)->P2

Tableau

S1	《1》○⊤ ; ¬p ; p→ 《1》◊q	(0)->S4
S2	《1》○《1》◊q ; 《1》◊q ; p→ 《1》◊q	(0)->S5 ; (0)->S6
S3	《1》○⊤ ; q ; 《1》◊q ; p→ 《1》◊q	(0)->S4
S4	《1》○⊤ ; ⊤	(0)->S4
S5	《1》○《1》◊q ; 《1》◊q	(0)->S5 ; (0)->S6
S6	《1》○⊤ ; q ; 《1》◊q	(0)->S4

Fig. 1. Screenshot of TATL's result page for options 1 and 2

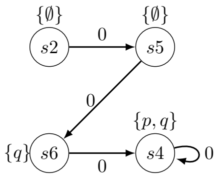
we use PHP for the graphical user interface of the web application. The web application and binaries are both available at `http://atila.ibisc.univ-evry.fr/tableau_ATL/`. In this paper, we will focus on the web application as the functionalities of binaries are similar. However, you can find details about the use of the binaries on the web page.

At the top of the page, a menu gives access to three options:

1. enter a formula as specified above;
2. choose among a set of preselected formulas corresponding to test cases (see Section 5);
3. use the random generator of formulas.

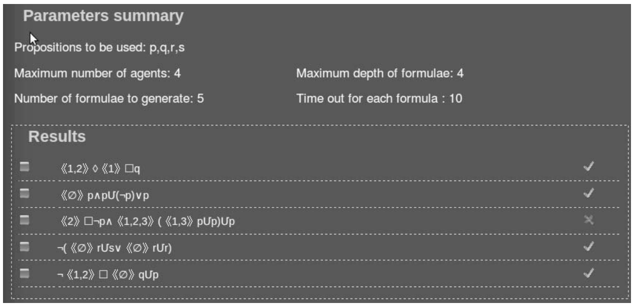
Option 1 (*One formula*) allows one to enter an ATL-formula and get in return the information about the satisfiability of the formula as well as the pretableau and the tableau, as shown in the screenshot in Fig. 1.

On the result area, each state (or prestate) is displayed with 4 elements: a name, a set of formulas, a mark and a set of successors. Prestates are referred to as Px and states as Sx where x is a number and the pretableau always begins with prestate P1. The check mark indicates that the state is consistent whereas the cross indicates that it is inconsistent. Successors of states are also given with their associated move vectors. An example of a move vector could be, for instance, $(0, 1, 0)$ for a formula with 3 agents. As agents are automatically sorted by their number, the first element of the move vector corresponds to the choice of the player with the smallest number. When a formula is satisfiable, this means that there exists at least one model for that formula. It is possible to manually construct a model from a tableau via the explanations in the completeness proof of [3]. For the tableau of Fig. 1, a model can be:



Option 2 (*Preselected formula*) allows one to select a formula among a set of 42 formulas and get in return the pretableau, the tableau, and the satisfiability of the formula, in the same way as in option 1. This set of formulas has been used to test the application (see Section 5).

Options 3 (*Random formulas*) allows one to randomly generate a set of ATL-formulas and get the answer on their satisfiability. This option needs some additional information to run: a set of propositions, a maximum number of agents from which TATL creates effective agents, a number of formulas to generate, a maximal depth of formulas and a time-out in seconds to stop the computation when it takes too long. The screenshot in Fig. 2 shows the output for this option. A check mark indicates that the formula is satisfiable, a cross that the formula is unsatisfiable and a question mark that the computation has timed-out. Clicking on the box in front of each generated formula transforms the syntax of the


Parameters summary

Propositions to be used: p,q,r,s

Maximum number of agents: 4 Maximum depth of formulae: 4

Number of formulae to generate: 5 Time out for each formula : 10

Results

⟪1,2⟫ ◊ ⟪1⟫ □q	✓
⟪∅⟫ p∧pU(¬p)∨p	✓
⟪2⟫ □¬p∧ ⟪1,2,3⟫ (⟪1,3⟫ pUp)Up	✗
¬(⟪∅⟫ rUs∨ ⟪∅⟫ rUr)	✓
¬ ⟪1,2⟫ □ ⟪∅⟫ qUp	✓

Fig. 2. Example of results obtained with the option "random formulas"

formula to make it compatible with option "One formula", thereby getting the pretableau and tableau of the formula.

5 Tests for TATL

A common way to test an implementation is to compare the outputs against another implementation. But, to our knowledge, there are no available tools to decide the satisfiability of an ATL-formula, either by using tableaux or by using automata. So we decided to check that TATL works correctly by creating a set of ATL-formulas that enables us to test each part of the algorithm. It should be noted that, to our knowledge, a benchmark set does not exist for ATL-satisfiability, thus our set of formulas might be used as a starting set for more refined future benchmarks. Our set consists of 42 formulas and allows us to test 50 points distributed in 15 categories, which are:

Coalition screen output	Next-time formulas
recognition of formulas	Cartesian product for move vectors
Treatment of agents	formula decomposition
Primitives	Rule E2
Inconsistency(Rule E1)	Rule E3
Eventualities	Creation of state/prestates sets
Move vectors	Several eventualities
sorting of Next-time formulas	

The set of formulas is provided in the appendix. Details of the 50 points and of the 15 categories can be found at `http://atila.ibisc.univ-evry.fr/tableau_ATL/test_cases.ods`. For instance, the formulas 23, 24 and 25 have been conceived to test eventualities, whereas formulas 16 and 17 allowed us to test the rule **E2**.

In order to test our implementation, we manually calculated a tableau for each of the 42 formulas and compared our results with TATL results to ensure that both satisfiability outputs and tableau's descriptions comply with the specification for these test cases.

6 Conclusion and Perspectives

TATL is, to our knowledge, the first implementation for testing the satisfiability of an ATL-formula. TATL is multi-platform and easy to use thanks to its web interface. TATL is also available as a command line application. As no reference tools were available for testing, we also provide a set of ATL-formulas. Such a set can be reused to develop more sophisticated benchmarks for ATL. TATL is a prototype so we need to improve it, for example, using better data structures to save computation time. We also worked on automata's construction based on tableaux, so we plan to add this functionality to TATL, by adding automata construction.

Acknowledgment. The author would like to thank for their help and support Serenella Cerrito, Marta Cialdea Mayer, Valentin Goranko and Laurent Poligny.

References

1. Alur, R., Henzinger, T.A., Kupferman, O.: Alternating-time temporal logic. J. ACM 49(5), 672–713 (2002)
2. Goranko, V., van Drimmelen, G.: Complete axiomatization and decidability of Alternating-time temporal logic. Theor. Comput. Sci. 353(1-3), 93–117 (2006)
3. Goranko, V., Shkatov, D.: Tableau-based decision procedures for logics of strategic ability in multi-agent systems. ACM Trans. Comput. Log. 11(1) (2009)

List of the 42 Formulas

1 p	5 $\langle\langle 1\rangle\rangle\bigcirc p$	9 $\neg\langle\langle 1\rangle\rangle pUq$	11 $\langle\langle 1,2\rangle\rangle p\mathcal{U}q \wedge \langle\langle 1,2\rangle\rangle\bigcirc r$
2 $p \wedge q$	6 $\langle\langle 1\rangle\rangle\Diamond p$	10 $\neg\langle\langle 1\rangle\rangle\Diamond p$	12 $\langle\langle 1,2\rangle\rangle p\mathcal{U}q \wedge \langle\langle 3,4\rangle\rangle\bigcirc r$
3 $p \vee q$	7 $\langle\langle 1\rangle\rangle\Box p$	18 $\neg p \vee \langle\langle 1\rangle\rangle\Diamond p$	13 $\langle\langle 1,2\rangle\rangle p\mathcal{U}q \wedge \langle\langle 2,3\rangle\rangle\bigcirc r$
4 $p \to q$	8 $\langle\langle 1\rangle\rangle p\mathcal{U}q$	19 $p \wedge \neg p$	14 $\langle\langle 2,1\rangle\rangle p\mathcal{U}q \wedge \langle\langle 3,2\rangle\rangle\bigcirc r$

15 $\langle\langle\rangle\rangle p\mathcal{U}q \wedge \langle\langle 1,2\rangle\rangle\bigcirc r$	23 $\langle\langle 1\rangle\rangle p\mathcal{U}q \vee \neg\langle\langle 1\rangle\rangle\Box q$
16 $\neg\langle\langle 1,2\rangle\rangle\bigcirc p \wedge \langle\langle 1\rangle\rangle\Box p$	24 $\langle\langle 1,2\rangle\rangle p\mathcal{U}(\neg\langle\langle 1\rangle\rangle\Box p)$
17 $\neg\langle\langle 1,2\rangle\rangle\bigcirc p \wedge \langle\langle 1,2,3\rangle\rangle\Box p$	25 $\langle\langle 1\rangle\rangle(\neg\langle\langle 1,2\rangle\rangle\Box p)Uq$
20 $(p \wedge q) \wedge \langle\langle 1\rangle\rangle\Box\neg(p \wedge q)$	26 $\langle\langle\rangle\rangle\Box\langle\langle\rangle\rangle p\mathcal{U}q$
21 $\langle\langle 1\rangle\rangle\Box p \wedge \neg\langle\langle 2\rangle\rangle\Diamond\langle\langle 1\rangle\rangle\Box p$	27 $\neg\langle\langle 1\rangle\rangle\Box p \wedge \langle\langle 1,2\rangle\rangle\bigcirc p \wedge \neg\langle\langle 2\rangle\rangle\bigcirc\neg p$
22 $\langle\langle 1\rangle\rangle\bigcirc p \wedge \neg\langle\langle 1\rangle\rangle\bigcirc p$	31 $\langle\langle 1,2,3\rangle\rangle\Box\langle\langle 2,3,4\rangle\rangle\Box(p \wedge q)$

33 $\neg\neg\langle\langle 1\rangle\rangle pUq$	38 $\langle\langle 1\rangle\rangle\Box p \wedge \neg\langle\langle 1,2\rangle\rangle\Box p$
34 $\neg(\langle\langle 1\rangle\rangle\Box p \vee \langle\langle 1\rangle\rangle\Box\neg p)$	39 $\neg\langle\langle 1\rangle\rangle\bigcirc p \wedge \langle\langle 2\rangle\rangle\bigcirc\neg p$
35 $\neg(\langle\langle 1\rangle\rangle\Box p \wedge \langle\langle 1\rangle\rangle\Box\neg p)$	40 $\langle\langle 1\rangle\rangle\bigcirc p \wedge \langle\langle 2\rangle\rangle\bigcirc\neg p$
36 $\neg\langle\langle 1\rangle\rangle p\mathcal{U}\neg\langle\langle 2\rangle\rangle q\mathcal{U}r$	41 $\langle\langle 1\rangle\rangle pUq \wedge \langle\langle 2\rangle\rangle q\mathcal{U}r \wedge \langle\langle 2\rangle\rangle\Box\neg r$
37 $\langle\langle 1\rangle\rangle\Box\neg q \wedge \langle\langle 2\rangle\rangle pUq$	42 $\langle\langle 1\rangle\rangle pUq \wedge \langle\langle 2\rangle\rangle q\mathcal{U}r \wedge \langle\langle 1\rangle\rangle\Box\neg r$

28 $\langle\langle 1\rangle\rangle\bigcirc p \wedge \langle\langle 2\rangle\rangle\bigcirc q \wedge \langle\langle 1,2\rangle\rangle\bigcirc r \wedge \neg\langle\langle 1\rangle\rangle\bigcirc r \wedge \neg\langle\langle 3\rangle\rangle\bigcirc q$
29 $\neg\langle\langle 1\rangle\rangle\bigcirc r \wedge \neg\langle\langle 3\rangle\rangle\bigcirc q \wedge \langle\langle 1\rangle\rangle\bigcirc p \wedge \langle\langle 2\rangle\rangle\bigcirc q \wedge \langle\langle 1,2\rangle\rangle\bigcirc r$
30 $\neg\langle\langle 1\rangle\rangle\bigcirc r \wedge \langle\langle 1\rangle\rangle\bigcirc p \wedge \langle\langle 2\rangle\rangle\bigcirc q \wedge \neg\langle\langle 3\rangle\rangle\bigcirc q \wedge \langle\langle 1,2\rangle\rangle\bigcirc r$
32 $\langle\langle 1,2,3\rangle\rangle\Box\langle\langle 2,3\rangle\rangle\Box(p \wedge q) \wedge \langle\langle 4\rangle\rangle\bigcirc\neg p$

A Terminating Evaluation-Driven Variant of G3i

Mauro Ferrari[1], Camillo Fiorentini[2], and Guido Fiorino[3]

[1] DiSTA, Univ. degli Studi dell'Insubria, Via Mazzini, 5, 21100, Varese, Italy
[2] DI, Univ. degli Studi di Milano, Via Comelico, 39, 20135 Milano, Italy
[3] DISCO, Univ. degli Studi di Milano-Bicocca, Viale Sarca, 336, 20126, Milano, Italy

Abstract. We present **Gbu**, a terminating variant of the sequent calculus **G3i** for intuitionistic propositional logic. **Gbu** modifies **G3i** by annotating the sequents so to distinguish rule applications into two phases: an unblocked phase where any rule can be backward applied, and a blocked phase where only right rules can be used. Derivations of **Gbu** have a trivial translation into **G3i**. Rules for right implication exploit an *evaluation* relation, defined on sequents; this is the key tool to avoid the generation of branches of infinite length in proof-search. To prove the completeness of **Gbu**, we introduce a refutation calculus **Rbu** for unprovability dual to **Gbu**. We provide a proof-search procedure that, given a sequent as input, returns either a **Rbu**-derivation or a **Gbu**-derivation of it.

1 Introduction

It is well-known that **G3i** [10], the sequent calculus for intuitionistic propositional logic with weakening and contraction "absorbed" in the rules, is not suited for proof-search. Indeed, the naïve proof-search strategy, consisting in applying the rules of the calculus bottom-up until possible, is not terminating. This is because the rule for left implication retains the main formula $A \to B$ in the left-hand side premise, hence such a formula might be selected for application more and more times. A possible solution to this problem is to support the proof-search procedure with a *loop-checking* mechanism [5,6,7]: whenever the "same" sequent occurs twice along a branch of the proof under construction, the search is cut. An efficient implementation of loop-checking exploits *histories* [6,7]. In the construction of a branch, the formulas decomposed by right rules are stored in the history; loops are avoided by preventing the application of some right rules to formulas in the history.

In this paper we propose a different and original approach: we show that terminating proof-search for **G3i** can be accomplished only exploiting the information contained in the sequent to be proved by means of a suitable *evaluation relation*. Our proof-search strategy alternates two phases: an unblocked phase (u-phase), where all the rules of **G3i** can be backward applied, and a blocked phase (b-phase), where only right-rules can be used. To improve the presentation, we embed the strategy inside the calculus by annotating sequents with the label u (*unblocked*) or b (*blocked*); we call **Gbu** the resulting calculus (see Fig. 1). A **Gbu**-derivation can be straightforwardly mapped to a **G3i**-derivation by erasing the labels and, possibly, by padding the left contexts; from this, the soundness of **Gbu** immediately follows. Unblocked sequents, characterizing an u-phase, behave as the ordinary sequents of **G3i**: any rule of **Gbu** can be (backward) applied to them. Instead, b-sequents resemble focused-right sequents (see, e.g., [2]): they only allow backward right-rule applications (thus, the left context is "blocked"). Proof-search starts from an u-sequent (u-phase); the transition to a b-phase is determined by the application of one of the rules for left implication or right disjunction. For instance, let $[A \to B, \Gamma \overset{\mathrm{u}}{\Rightarrow} H]$ be the u-sequent to be proved and suppose we apply the rule $\to L$ with main formula $A \to B$. The next goals are the b-sequent $[A \to B, \Gamma \overset{\mathrm{b}}{\Rightarrow} A]$ and the u-sequent $[B, \Gamma \overset{\mathrm{u}}{\Rightarrow} H]$, corresponding to the two premises of $\to L$. While the latter goal continues the u-phase, the former one starts a new b-phase, which focuses on A. Similarly, if we apply the rule $\vee R_k$ (with $k \in \{0,1\}$) to $[\Gamma \overset{\mathrm{u}}{\Rightarrow} H_0 \vee H_1]$, the phase changes to b and the next goal is $[\Gamma \overset{\mathrm{b}}{\Rightarrow} H_k]$, the only premise of $\vee R_k$.

Rules for right implication have two possible outcomes determined by the evaluation relation. Indeed, let $[\Gamma \overset{l}{\Rightarrow} A \to B]$ be the current goal ($l \in \{\mathrm{u}, \mathrm{b}\}$) and let $A \to B$ be the selected main formula: if A is evaluated in Γ, then we continue the search with $[\Gamma \overset{l}{\Rightarrow} B]$ and the phase does not change (see rule $\to R_1$); note that the formula A is dropped out. If A is not evaluated in Γ the next goal is $[A, \Gamma \overset{\mathrm{u}}{\Rightarrow} B]$. Moreover, if $l = \mathrm{b}$, we switch from a b-phase to an u-phase and this is the only case where a b-sequent is "unblocked". The crucial point is that, due to the side conditions on the application of rules $\to R_1$ and $\to R_2$ (which rely on the evaluation relation), every branch of a **Gbu**-tree has finite length (Section 3); this implies that our proof-search strategy always terminates. We point out that we do not bound ourselves to a specific evaluation relation, but we admit any evaluation relation satisfying properties ($\mathcal{E}1$)–($\mathcal{E}6$) defined in Section 2.

The proof of completeness ($[\Gamma \Rightarrow H]$ provable in **G3i** implies $[\Gamma \overset{\mathrm{u}}{\Rightarrow} H]$ provable in **Gbu**) involves non-trivial aspects. Following [3,9], we introduce a refutation calculus **Rbu** for asserting intuitionistic unprovability (Section 4). From an **Rbu**-derivation of an u-sequent $\sigma^{\mathrm{u}} = [\Gamma \overset{\mathrm{u}}{\Rightarrow} H]$ we can extract a Kripke countermodel for σ^{u}, namely a Kripke model such that, at its root, all formulas in Γ are forced and H is not forced; from this, it follows that σ^{u} is not intuitionistically valid. In Section 5 we introduce the function `F` which implements the proof-search strategy outlined above; if the search for a **Gbu**-derivation of σ^{u} fails, an **Rbu**-derivation of σ^{u} is built. To sum up, `F`(σ^{u}) returns either a **Gbu**-derivation or an **Rbu**-derivation of σ^{u}; in the former case we get a **G3i**-derivation of the sequent $\sigma = [\Gamma \Rightarrow H]$, in the latter case we can build a countermodel for σ.

2 Preliminaries and Evaluations

We consider the propositional language $\mathcal{L}$ based on a denumerable set of propositional variables $\mathcal{V}$, the connectives $\wedge$, $\vee$, $\to$ and the logical constant $\bot$. We denote with $\mathcal{V}(A)$ the set of propositional variables occurring in A, with $|A|$ *the size of* A, that is the number of symbols occurring in A, and with $\mathrm{Sf}(A)$ the set of subformulas of A (including A itself).

A *(finite) Kripke model* for $\mathcal{L}$ is a structure $\mathcal{K} = \langle P, \leq, \rho, V \rangle$, where $\langle P, \leq, \rho \rangle$ is a finite partially ordered set with minimum ρ and $V : P \to 2^{\mathcal{V}}$ is a function such that $\alpha \leq \beta$ implies $V(\alpha) \subseteq V(\beta)$. The *forcing relation* $\Vdash \subseteq P \times \mathcal{L}$ is defined as follows:

- $\mathcal{K}, \alpha \nVdash \bot$ and, for every $p \in \mathcal{V}$, $\mathcal{K}, \alpha \Vdash p$ iff $p \in V(\alpha)$;
- $\mathcal{K}, \alpha \Vdash A \wedge B$ iff $\mathcal{K}, \alpha \Vdash A$ and $\mathcal{K}, \alpha \Vdash B$;
- $\mathcal{K}, \alpha \Vdash A \vee B$ iff $\mathcal{K}, \alpha \Vdash A$ or $\mathcal{K}, \alpha \Vdash B$;
- $\mathcal{K}, \alpha \Vdash A \to B$ iff, for every $\beta \in P$ such that $\alpha \leq \beta$, $\mathcal{K}, \beta \nVdash A$ or $\mathcal{K}, \beta \Vdash B$.

Given a set Γ of formulas, $\mathcal{K}, \alpha \Vdash \Gamma$ iff $\mathcal{K}, \alpha \Vdash A$ for every $A \in \Gamma$. *Monotonicity property* holds for arbitrary formulas, i.e.: $\mathcal{K}, \alpha \Vdash A$ and $\alpha \leq \beta$ imply $\mathcal{K}, \beta \Vdash A$. A formula A is *valid* in $\mathcal{K}$ iff $\mathcal{K}, \rho \Vdash A$. Intuitionistic propositional logic coincides with the set of the formulas valid in all (finite) Kripke models [1].

As motivated in the Introduction, we use (labelled) sequents of the form $\sigma = [\Gamma \overset{l}{\Rightarrow} H]$ where $l \in \{\mathrm{b}, \mathrm{u}\}$, Γ is a finite set of formulas and H is a formula. We adopt the usual notational conventions; e.g., $[A, \Gamma \overset{l}{\Rightarrow} H]$ stands for $[\{A\} \cup \Gamma \overset{l}{\Rightarrow} H]$. The *size* of σ is $|\sigma| = \sum_{A \in \Gamma} |A| + |H|$; the set of subformulas of σ is $\mathrm{Sf}(\sigma) = \bigcup_{A \in \Gamma \cup \{H\}} \mathrm{Sf}(A)$.

The semantics of formulas extends to sequents as follows. Given a Kripke model $\mathcal{K}$ and a world α of $\mathcal{K}$, α *refutes* $\sigma = [\Gamma \overset{l}{\Rightarrow} H]$ *in* $\mathcal{K}$, written $\mathcal{K}, \alpha \triangleright \sigma$, iff $\mathcal{K}, \alpha \Vdash \Gamma$ and $\mathcal{K}, \alpha \nVdash H$; σ is *refutable* if there exists a Kripke model $\mathcal{K}$ with root ρ such that $\mathcal{K}, \rho \triangleright \sigma$; in this case $\mathcal{K}$ is a *countermodel* for σ. It is easy to check that σ is refutable iff the formula $\wedge\Gamma \to H$ is not intuitionistically valid iff, by soundness and completeness of **G3i** [10], $[\Gamma \Rightarrow H]$ is not provable in **G3i**.

Evaluations. An *evaluation relation* $\vdash_{\mathcal{E}}$ is a relation between a set Γ of formulas and a formula A satisfying the following properties:

($\mathcal{E}$1) $\Gamma \vdash_{\mathcal{E}} A$ iff $\Gamma \cap \mathrm{Sf}(A) \vdash_{\mathcal{E}} A$.
($\mathcal{E}$2) $A, \Gamma \vdash_{\mathcal{E}} A$.
($\mathcal{E}$3) $\Gamma \vdash_{\mathcal{E}} A$ and $\Gamma \vdash_{\mathcal{E}} B$ implies $\Gamma \vdash_{\mathcal{E}} A \wedge B$.
($\mathcal{E}$4) $\Gamma \vdash_{\mathcal{E}} A_k$, with $k \in \{0, 1\}$, implies $\Gamma \vdash_{\mathcal{E}} A_0 \vee A_1$.
($\mathcal{E}$5) $\Gamma \vdash_{\mathcal{E}} B$ implies $\Gamma \vdash_{\mathcal{E}} A \to B$.
($\mathcal{E}$6) Let $\mathcal{K} = \langle P, \leq, \rho, V \rangle$ and $\alpha \in P$; if $\mathcal{K}, \alpha \Vdash \Gamma$ and $\Gamma \vdash_{\mathcal{E}} A$, then $\mathcal{K}, \alpha \Vdash A$.

Conditions ($\mathcal{E}$1)–($\mathcal{E}$5) concern syntactical properties; note that, by ($\mathcal{E}$1), the evaluation of A w.r.t. Γ only depends on the subformulas in Γ which are subformulas of A. Intuitively, the role of an evaluation relation is to check if the "information contained" in A is semantically implied by Γ (see ($\mathcal{E}$6)). In the sequel, we also write $[\Gamma \overset{l}{\Rightarrow} H] \vdash_{\mathcal{E}} A$ to mean $\Gamma \vdash_{\mathcal{E}} A$.

In the examples we use the evaluation relation $\vdash_{\tilde{\mathcal{E}}}$ defined below. Let $\mathcal{L}_\top$ be the language extending $\mathcal{L}$ with the constant $\top$ ($\mathcal{K}, \alpha \Vdash \top$, for every $\mathcal{K}$ and every α in $\mathcal{K}$). To define $\vdash_{\tilde{\mathcal{E}}}$, we introduce the function $\mathcal{R}$ which simplifies a formula $A \in \mathcal{L}_\top$ w.r.t. a set Γ of formulas of $\mathcal{L}$ (see [4]):

$$\mathcal{R}(A,\Gamma) = \begin{cases} \top & A \in \Gamma \\ A & \text{if } A \notin \Gamma \text{ and } A \in \mathcal{V} \cup \{\bot, \top\} \\ \mathcal{B}\left(\mathcal{R}(A_0,\Gamma) \cdot \mathcal{R}(A_1,\Gamma)\right) & \text{if } A \notin \Gamma \text{ and } A = A_0 \cdot A_1 \text{ with } \cdot \in \{\wedge, \vee, \rightarrow\} \end{cases}$$

$\mathcal{B}(A)$ performs the *boolean simplification* of A [4,8], consisting in applying the following reductions inside A:

$$\begin{array}{llllll} K \wedge \top \rightsquigarrow K & K \wedge \bot \rightsquigarrow \bot & K \vee \top \rightsquigarrow \top & K \vee \bot \rightsquigarrow K & K \rightarrow \top \rightsquigarrow \top & K \rightarrow K \rightsquigarrow \top \\ \top \wedge K \rightsquigarrow K & \bot \wedge K \rightsquigarrow \bot & \top \vee K \rightsquigarrow \top & \bot \vee K \rightsquigarrow K & \top \rightarrow K \rightsquigarrow K & \bot \rightarrow K \rightsquigarrow \top \end{array}$$

We set $\Gamma \vdash_{\tilde{\mathcal{E}}} A$ iff $\mathcal{R}(A,\Gamma) = \top$.

Theorem 1. $\vdash_{\tilde{\mathcal{E}}}$ *is an evaluation relation.*

Proof. We have to prove that $\vdash_{\tilde{\mathcal{E}}}$ satisfies properties ($\mathcal{E}1$)–($\mathcal{E}6$) of Section 2.

- ($\mathcal{E}1$) It is easy to prove, by induction on the structure of A, that $\mathcal{R}(A,\Gamma) = \mathcal{R}(A,\Gamma \cap \mathrm{Sf}(A))$, thus $\Gamma \vdash_{\tilde{\mathcal{E}}} A$ iff $\Gamma \cap \mathrm{Sf}(A) \vdash_{\tilde{\mathcal{E}}} A$.
- ($\mathcal{E}2$) It immediately follows by the definition of $\vdash_{\tilde{\mathcal{E}}}$ and $\mathcal{R}$.
- ($\mathcal{E}3$) Let $\Gamma \vdash_{\tilde{\mathcal{E}}} A$ and $\Gamma \vdash_{\tilde{\mathcal{E}}} B$. By definition of $\vdash_{\tilde{\mathcal{E}}}$, $\mathcal{R}(A,\Gamma) = \mathcal{R}(B,\Gamma) = \top$. To prove $\Gamma \vdash_{\tilde{\mathcal{E}}} A \wedge B$, we must show that $\mathcal{R}(A \wedge B,\Gamma) = \top$. If $A \wedge B \in \Gamma$, this immediately follows. Otherwise: $\mathcal{R}(A \wedge B,\Gamma) = \mathcal{B}(\mathcal{R}(A,\Gamma) \wedge \mathcal{R}(B,\Gamma)) = \mathcal{B}(\top \wedge \top) = \top$. The proof of properties ($\mathcal{E}4$) and ($\mathcal{E}5$) is similar.
- ($\mathcal{E}6$) Let $\mathcal{K} = \langle P, \leq, \rho, V \rangle$ and $\alpha \in P$ such that $\mathcal{K}, \alpha \Vdash \Gamma$. It is easy to prove, by induction on A, that $\mathcal{K}, \alpha \Vdash A \leftrightarrow \mathcal{R}(A,\Gamma)$. Now, if $\Gamma \vdash_{\tilde{\mathcal{E}}} A$ then $\mathcal{R}(A,\Gamma) = \top$; hence by the above property $\mathcal{K}, \alpha \Vdash A \leftrightarrow \top$ and this implies $\mathcal{K}, \alpha \Vdash A$. □

3 The Sequent Calculus Gbu

We present the **G3**-style [10] calculus **Gbu** for intuitionistic propositional logic. The calculus consists of the *axiom rules* (rules with zero premises) $\bot L$ and Id, and the left and right introduction rules in Fig. 1. The *main formula* of a rule is the one put in evidence in the conclusion of the rule. In the conclusion of a rule, when we write C, Γ we assume that $C \notin \Gamma$; e.g., in the rule $\wedge L$ it is assumed that $A \wedge B \notin \Gamma$, hence the formula $A \wedge B$ is not retained in the premise. The choice between $\rightarrow R_1$ and $\rightarrow R_2$ depends on the relation $\vdash_{\mathcal{E}}$. In the application of $\rightarrow L$ to $\sigma = [A \rightarrow B, \Gamma \overset{\mathrm{u}}{\Rightarrow} H]$, contraction of $A \rightarrow B$ is explicitly introduced in the leftmost premise σ_A; as a consequence we might have $|\sigma_A| \geq |\sigma|$. In all the other cases, passing from the conclusion to a premise of a rule, the size of the sequents strictly decreases. The rule $\rightarrow R_2$ is the only rule that, when applied backward, can turn a b-sequent into an u-sequent.

A **Gbu**-*tree* π is a tree of sequents such that: if σ is a node of π with $\sigma_1, \ldots, \sigma_n$ as children, then there exists a rule of **Gbu** having premises $\sigma_1, \ldots, \sigma_n$ and conclusion σ. The *root rule of* π is the one having as conclusion the root sequent of π. A **Gbu**-*derivation* of σ is a **Gbu**-tree π with root σ and having conclusions

$$\frac{}{[\bot, \Gamma \overset{l}{\Rightarrow} H]}\ \bot L \qquad \frac{}{[H, \Gamma \overset{l}{\Rightarrow} H]}\ \mathrm{Id}$$

$$\frac{[A, B, \Gamma \overset{\mathrm{u}}{\Rightarrow} H]}{[A \wedge B, \Gamma \overset{\mathrm{u}}{\Rightarrow} H]}\ \wedge L \qquad \frac{[\Gamma \overset{l}{\Rightarrow} A] \quad [\Gamma \overset{l}{\Rightarrow} B]}{[\Gamma \overset{l}{\Rightarrow} A \wedge B]}\ \wedge R$$

$$\frac{[A, \Gamma \overset{\mathrm{u}}{\Rightarrow} H] \quad [B, \Gamma \overset{\mathrm{u}}{\Rightarrow} H]}{[A \vee B, \Gamma \overset{\mathrm{u}}{\Rightarrow} H]}\ \vee L \qquad \frac{[\Gamma \overset{\mathrm{b}}{\Rightarrow} H_k]}{[\Gamma \overset{l}{\Rightarrow} H_0 \vee H_1]}\ \vee R_k \quad k \in \{0,1\}$$

$$\frac{[A \to B, \Gamma \overset{\mathrm{b}}{\Rightarrow} A]\,[B, \Gamma \overset{\mathrm{u}}{\Rightarrow} H]}{[A \to B, \Gamma \overset{\mathrm{u}}{\Rightarrow} H]}\ \to L \qquad \frac{[\Gamma \overset{l}{\Rightarrow} B]}{[\Gamma \overset{l}{\Rightarrow} A \to B]}\ \to R_1 \qquad \frac{[A, \Gamma \overset{\mathrm{u}}{\Rightarrow} B]}{[\Gamma \overset{l}{\Rightarrow} A \to B]}\ \to R_2$$

$$\text{if } \Gamma \vdash_{\mathcal{E}} A \qquad\qquad \text{if } \Gamma \nvdash_{\mathcal{E}} A$$

Fig. 1. The calculus **Gbu**

of an axiom rule as leaves. A sequent σ is provable in **Gbu** iff there exists a **Gbu**-derivation of σ; H is provable in **Gbu** iff $[\overset{\mathrm{u}}{\Rightarrow} H]$ is provable in **Gbu**. Note that **Gbu** has the *subformula property*: given a **Gbu**-tree π with root σ, for every sequent σ' occurring in π it holds that $\mathrm{Sf}(\sigma') \subseteq \mathrm{Sf}(\sigma)$.

A **Gbu**-derivation π can be translated into a **G3i**-derivation $\tilde{\pi}$ applying the following steps: erase the labels from the sequents in π; when rule $\to R_1$ is applied, add the formula A to the left context; rename all occurrences of $\to R_1$ and $\to R_2$ to $\to R$. From this translation and the soundness of **G3i** [10] we get the soundness of **Gbu**. Semantically, this means that, if σ is provable in **Gbu**, then σ is not refutable.

Here we provide an example of a **Gbu**-derivation, then we prove that **Gbu** is terminating. The completeness of **Gbu** (Theorem 4) is proved in Section 5 as a consequence of the correctness of the proof-search procedure.

Example 1. Let $W = ((((p \to q) \to p) \to p) \to q) \to q$ be an instance of the *Weak Pierce Law* [1]. In Fig. 2 we give a **Gbu**-derivation[1] π_1 of $\sigma_1 = [\overset{\mathrm{u}}{\Rightarrow} W]$, using the evaluation $\vdash_{\tilde{\mathcal{E}}}$ of Section 2. Sequents are indexed by integers; by π_i we denote the subderivation of π_1 with root σ_i. When ambiguities can arise, we underline the main formula of a rule application. Building the derivation bottom-up, the only choice points are in the (backward) application of rule $\to L$ to σ_4 and σ_7, since we can select both A and B as main formula. If at sequent σ_6 we choose B instead of A, we get the **Gbu**-tree with root σ_6 sketched on the right. We have $\sigma_{7'} \vdash_{\tilde{\mathcal{E}}} p$ (indeed, p occurs on the left in $\sigma_{7'}$), hence the rule $\to R_1$ must be applied to $\sigma_{7'}$, which

$$\frac{\dfrac{[p, B, A \overset{\mathrm{b}}{\Rightarrow} q]_{8'}}{[p, B, A \overset{\mathrm{b}}{\Rightarrow} p \to q]_{7'}}\ \to R_1 \qquad \dfrac{\vdots}{[p, A \overset{\mathrm{u}}{\Rightarrow} q]_{9'}}}{[p, \underbrace{\underline{(p \to q) \to p}}_{B}, A \overset{\mathrm{u}}{\Rightarrow} q]_6}\ \to L$$

[1] The derivations and their LaTeX rendering are generated with `g3ibu`, an implementation of **Gbu** and **Rbu** available at `http://www.dista.uninsubria.it/~ferram/`.

$$W = A \to q \qquad A = (B \to p) \to q \qquad B = (p \to q) \to p$$

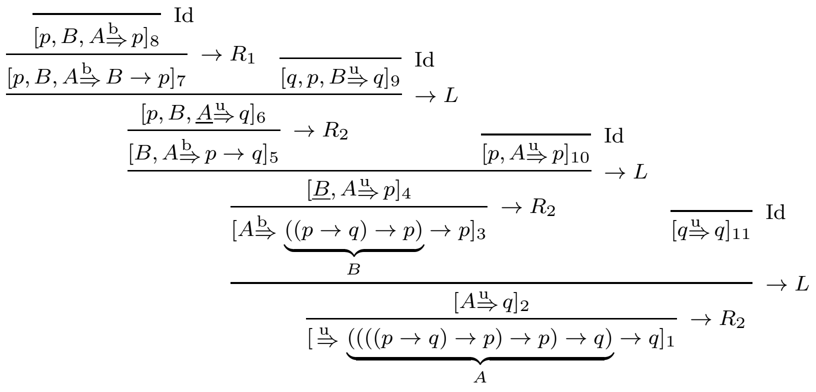


Fig. 2. **Gbu**-derivation of Weak Pierce Law

yields the b-sequent $\sigma_{8'}$. Since $\sigma_{8'}$ is blocked, we cannot decompose again left implications; thus the proof-search fails without entering an infinite loop. $\Diamond$

Termination of Gbu. We show that every **Gbu**-tree has finite depth. A **Gbu**-*branch* is a sequence of sequents $\mathcal{B} = (\sigma_1, \sigma_2, \dots)$ such that, for every $i \geq 1$, there exists a rule $\mathcal{R}$ of **Gbu** having σ_i as conclusion and σ_{i+1} among its premises. The *length* of $\mathcal{B}$ is the number of sequents in it. Let $\gamma = (\sigma_i, \sigma_{i+1})$ be a pair of successive sequents in $\mathcal{B}$ with labels l_i and l_{i+1} respectively; γ is a bu-pair if $l_i = \text{b}$ and $l_{i+1} = \text{u}$; γ is an ub-pair if $l_i = \text{u}$ and $l_{i+1} = \text{b}$. By $\text{BU}(\mathcal{B})$ and $\text{UB}(\mathcal{B})$ we denote the number of bu-pairs and ub-pairs occurring in $\mathcal{B}$ respectively. Note that the only rule generating bu-pairs is $\to R_2$. Moreover, $|\sigma_{i+1}| \geq |\sigma_i|$ can happen only if (σ_i, σ_{i+1}) is an ub-pair generated by $\to L$: σ_{i+1} is the leftmost premise of an application of $\to L$ with conclusion σ_i. As a consequence, every subbranch of $\mathcal{B}$ not containing ub-pairs is finite. Hence, if we show that $\text{UB}(\mathcal{B})$ is finite, we get that $\mathcal{B}$ has finite length.

We prove a kind of persistence of $\vdash_{\mathcal{E}}$, namely: if A occurs in the left-hand side of a sequent σ occurring in $\mathcal{B}$, then $\sigma' \vdash_{\mathcal{E}} A$ for every σ' following σ in $\mathcal{B}$.

Lemma 1. *Let $\mathcal{B} = (\sigma_1, \sigma_2, \dots)$ be a* **Gbu**-*branch where, for every $i \geq 1$, $\sigma_i = [\Gamma_i \overset{l_i}{\Rightarrow} H_i]$. Let $n \geq 1$ and $A \in \bigcup_{1 \leq i \leq n} \Gamma_i$. Then, $\Gamma_n \vdash_{\mathcal{E}} A$.*

Proof. By induction on $|A|$. If $A \in \Gamma_n$, by $(\mathcal{E}2)$ we immediately get $\Gamma_n \vdash_{\mathcal{E}} A$. If $A \notin \Gamma_n$, there exists $i : 1 \leq i < n$ such that $A \in \Gamma_i$ and $A \notin \Gamma_{i+1}$. This implies $A = B \cdot C$ with $\cdot \in \{\wedge, \vee, \to\}$. Let $\cdot = \wedge$; then σ_{i+1} is obtained from σ_i by an application of $\wedge L$ with main formula $B \wedge C$, hence $B \in \Gamma_{i+1}$ and $C \in \Gamma_{i+1}$. By induction hypothesis, $\Gamma_n \vdash_{\mathcal{E}} B$ and $\Gamma_n \vdash_{\mathcal{E}} C$; by $(\mathcal{E}3)$, $\Gamma_n \vdash_{\mathcal{E}} B \wedge C$. The cases $\cdot \in \{\vee, \to\}$ are similar and require properties $(\mathcal{E}4)$ and $(\mathcal{E}5)$. $\square$

Now, we provide a bound on $\text{BU}(\mathcal{B})$.

$$\frac{}{[\Gamma^{\rightarrow}, \Gamma^{\mathrm{At}} \overset{l}{\Rightarrow} H]}\ \mathrm{Irr} \qquad \text{if } [\Gamma^{\rightarrow}, \Gamma^{\mathrm{At}} \overset{l}{\Rightarrow} H] \text{ is irreducible } \begin{cases} H = \bot \text{ or } H \in \mathcal{V} \setminus \Gamma^{\mathrm{At}} \\ l = \mathrm{b} \text{ or } \Gamma^{\rightarrow} = \emptyset \end{cases}$$

$$\frac{[A, B, \Gamma \overset{\mathrm{u}}{\Rightarrow} H]}{[A \wedge B, \Gamma \overset{\mathrm{u}}{\Rightarrow} H]}\ \wedge L \qquad \frac{[\Gamma \overset{l}{\Rightarrow} H_k]}{[\Gamma \overset{l}{\Rightarrow} H_0 \wedge H_1]}\ \wedge R_k \quad k \in \{0,1\}$$

$$\frac{[A_k, \Gamma \overset{\mathrm{u}}{\Rightarrow} H]}{[A_0 \vee A_1, \Gamma \overset{\mathrm{u}}{\Rightarrow} H]}\ \vee L_k \quad k \in \{0,1\} \qquad \frac{[\Gamma \overset{\mathrm{b}}{\Rightarrow} H_0] \quad [\Gamma \overset{\mathrm{b}}{\Rightarrow} H_1]}{[\Gamma \overset{\mathrm{b}}{\Rightarrow} H_0 \vee H_1]}\ \vee R$$

$$\frac{[B, \Gamma \overset{\mathrm{u}}{\Rightarrow} H]}{[A \rightarrow B, \Gamma \overset{\mathrm{u}}{\Rightarrow} H]}\ \rightarrow L \qquad \frac{[\Gamma \overset{l}{\Rightarrow} B]}{[\Gamma \overset{l}{\Rightarrow} A \rightarrow B]}\ \rightarrow R_1 \ \text{if } \Gamma \vdash_{\mathcal{E}} A \qquad \frac{[A, \Gamma \overset{\mathrm{u}}{\Rightarrow} B]}{[\Gamma \overset{l}{\Rightarrow} A \rightarrow B]}\ \rightarrow R_2 \ \text{if } \Gamma \nvdash_{\mathcal{E}} A$$

$$\frac{\{\, [\Gamma^{\rightarrow}, \Gamma^{\mathrm{At}} \overset{\mathrm{b}}{\Rightarrow} A] \,\}_{A \rightarrow B \in \Gamma^{\rightarrow}}}{[\Gamma^{\rightarrow}, \Gamma^{\mathrm{At}} \overset{\mathrm{u}}{\Rightarrow} H]}\ \mathrm{S}_u^{\mathrm{At}} \qquad \text{where } \Gamma^{\rightarrow} \neq \emptyset \text{ and } (H = \bot \text{ or } H \in \mathcal{V} \setminus \Gamma^{\mathrm{At}})$$

$$\frac{\{\, [\Gamma^{\rightarrow}, \Gamma^{\mathrm{At}} \overset{\mathrm{b}}{\Rightarrow} A] \,\}_{A \rightarrow B \in \Gamma^{\rightarrow}} \quad [\Gamma^{\rightarrow}, \Gamma^{\mathrm{At}} \overset{\mathrm{b}}{\Rightarrow} H_0] \quad [\Gamma^{\rightarrow}, \Gamma^{\mathrm{At}} \overset{\mathrm{b}}{\Rightarrow} H_1]}{[\Gamma^{\rightarrow}, \Gamma^{\mathrm{At}} \overset{\mathrm{u}}{\Rightarrow} H_0 \vee H_1]}\ \mathrm{S}_u^{\vee}$$

Fig. 3. The refutation calculus **Rbu**

Lemma 2. *Let* $\mathcal{B} = (\sigma_1, \sigma_2, \dots)$ *be a* **Gbu**-*branch. Then,* $\mathrm{BU}(\mathcal{B}) \leq |\sigma_1|$.

Proof. Let $(\sigma_i^{\mathrm{b}}, \sigma_{i+1}^{\mathrm{u}})$ be a bu-pair in $\mathcal{B}$. Since bu-pairs are generated by applications of $\rightarrow R_2$, we have: $\sigma_i^{\mathrm{b}} = [\Gamma \overset{\mathrm{b}}{\Rightarrow} A \rightarrow B]$, $\sigma_{i+1}^{\mathrm{u}} = [A, \Gamma \overset{\mathrm{u}}{\Rightarrow} B]$ and $\Gamma \nvdash_{\mathcal{E}} A$. By Lemma 1, for every $j \geq i+1$ it holds that $\Gamma_j \vdash_{\mathcal{E}} A$. Thus, any bu-pair following $(\sigma_i^{\mathrm{b}}, \sigma_{i+1}^{\mathrm{u}})$ must treat an implication $C \rightarrow D$ with $C \neq A$. Since **Gbu** has the subformula property, the main formulas of $\rightarrow R_2$ applications belong to $\mathrm{Sf}(\sigma_1)$. Thus, $\mathrm{BU}(\mathcal{B})$ is bounded by the number $\#\mathrm{Sf}(\sigma_1)$ of subformulas of σ_1. Since $\#\mathrm{Sf}(\sigma_1) \leq |\sigma_1|$, we get $\mathrm{BU}(\mathcal{B}) \leq |\sigma_1|$. □

Since between two ub-pairs of $\mathcal{B}$ a bu-pair must occur, $\mathrm{UB}(\mathcal{B}) \leq \mathrm{BU}(\mathcal{B}) + 1$; by Lemma 2, $\mathrm{UB}(\mathcal{B})$ is finite. We can conclude:

Proposition 1. *Every* **Gbu**-*branch has finite length.* □

As a consequence, every **Gbu**-tree has finite depth and **Gbu** is terminating.

4 The Refutation Calculus Rbu

In this section, following the ideas of [3,9], we introduce the refutation calculus **Rbu** for deriving intuitionistic unprovability. Intuitively, an **Rbu**-derivation π of a sequent σ^{u} is a sort of "constructive proof" of refutability of σ^{u} in the sense that from π we can extract a countermodel $\mathrm{Mod}(\pi)$ for σ^{u}.

We denote with Γ^{At} a finite set of propositional variables and with $\Gamma^{\rightarrow}$ a finite set of implicative formulas. A sequent σ is *irreducible* iff $\sigma = [\Gamma^{\rightarrow}, \Gamma^{\mathrm{At}} \overset{l}{\Rightarrow} H]$ with

$H \in \{\bot\} \cup (\mathcal{V} \setminus \Gamma^{\mathrm{At}})$ and ($l = \mathrm{b}$ or $\Gamma^{\to} = \emptyset$). The rules of **Rbu** are given in Fig. 3. As in **Gbu**, writing C, Γ in the conclusion of a rule, we assume that $C \notin \Gamma$. The notions of **Rbu**-tree, **Rbu**-derivation and **Rbu**-branch are defined analogously to those for **Gbu**.

The rule $\mathrm{S}_u^{\mathrm{At}}$ has a premise $[\Gamma^{\to}, \Gamma^{\mathrm{At}} \overset{\mathrm{b}}{\Rightarrow} A]$ for every A such that $A \to B \in \Gamma^{\to}$; since $\Gamma^{\to} \neq \emptyset$, there exists at least one premise. The rule $\mathrm{S}_u^{\vee}$ is similar and has at least two premises. All the premises of $\mathrm{S}_u^{\mathrm{At}}$ and $\mathrm{S}_u^{\vee}$ are b-sequents.

It is easy to check that an **Rbu**-branch is also a **Gbu**-branch[2]. Accordingly, Proposition 1 implies that the calculus **Rbu** is terminating. In the following we prove that **Rbu** is sound in the following sense:

Theorem 2 (Soundness of Rbu). *If an* u*-sequent* σ^{u} *is provable in* **Rbu**, *then* σ^{u} *is refutable.* □

Example 2. Let $S = ((\neg\neg p \to p) \to (\neg p \vee p)) \to (\neg\neg p \vee \neg p)$ be an instance of the *Scott principle* [1], where $\neg Z = Z \to \bot$. We show the **Rbu**-derivation π_1 of $[\overset{\mathrm{u}}{\Rightarrow} S]$.

$$S = A \to (\neg\neg p \vee \neg p) \qquad A = (\neg\neg p \to p) \to (\neg p \vee p)$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{[p, \neg\neg p \overset{\mathrm{b}}{\Rightarrow} \bot]_{10}}\,\mathrm{Irr}}{[p, \neg\neg p \overset{\mathrm{b}}{\Rightarrow} \neg p]_{9}}\,{\to}R_1}{[p, \neg\neg p \overset{\mathrm{u}}{\Rightarrow} \bot]_{8}}\,\mathrm{S}_u^{At}}{[\neg p \vee p, p, \neg\neg p \overset{\mathrm{u}}{\Rightarrow} \bot]_{7}}\,\vee L_1}{[p, \neg\neg p, \underline{A} \overset{\mathrm{u}}{\Rightarrow} \bot]_{6}}\,{\to}L}{[\neg\neg p, A \overset{\mathrm{b}}{\Rightarrow} \neg p]_{5}}\,{\to}R_2 \quad \dfrac{\dfrac{}{[\neg\neg p, A \overset{\mathrm{b}}{\Rightarrow} p]_{12}}\,\mathrm{Irr}}{[\neg\neg p, A \overset{\mathrm{b}}{\Rightarrow} \neg\neg p \to p]_{11}}\,{\to}R_1}{[\neg\neg p, A \overset{\mathrm{u}}{\Rightarrow} p]_{4}}\,\mathrm{S}_u^{At}}{[A \overset{\mathrm{b}}{\Rightarrow} \neg\neg p \to p]_{3}}\,{\to}R_2 \quad \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{[\neg p \overset{\mathrm{b}}{\Rightarrow} p]_{17}}\,\mathrm{Irr}}{[\neg p \overset{\mathrm{u}}{\Rightarrow} \bot]_{16}}\,\mathrm{S}_u^{At}}{[\neg p \vee p, \neg p \overset{\mathrm{u}}{\Rightarrow} \bot]_{15}}\,\vee L_0}{[\neg p, \underline{A} \overset{\mathrm{u}}{\Rightarrow} \bot]_{14}}\,{\to}L}{[A \overset{\mathrm{b}}{\Rightarrow} \neg\neg p]_{13}}\,{\to}R_2 \quad \dfrac{\dfrac{\dfrac{\dfrac{}{[p \overset{\mathrm{u}}{\Rightarrow} \bot]_{21}}\,\mathrm{Irr}}{[\neg p \vee p, p \overset{\mathrm{u}}{\Rightarrow} \bot]_{20}}\,\vee L_1}{[p, \underline{A} \overset{\mathrm{u}}{\Rightarrow} \bot]_{19}}\,{\to}L}{[A \overset{\mathrm{b}}{\Rightarrow} \neg p]_{18}}\,{\to}R_2}{[A \overset{\mathrm{u}}{\Rightarrow} \neg\neg p \vee \neg p]_{2}}\,\mathrm{S}_u^{\vee}}{[\overset{\mathrm{u}}{\Rightarrow} \underbrace{((\neg\neg p \to p) \to (\neg p \vee p))}_{A} \to (\neg\neg p \vee \neg p)]_{1}}\,{\to}R_2$$

◊

Soundness of Rbu. Let π be an **Rbu**-derivation with root $\sigma^{\mathrm{b}} = [\Gamma^{\to}, \Gamma^{\mathrm{At}} \overset{\mathrm{b}}{\Rightarrow} H]$. By $\Pi(\pi, \sigma^{\mathrm{b}})$ we denote the maximal subtree of π having root σ^{b} and only containing b-sequents (that is, any subtree of π with root σ^{b} extending $\Pi(\pi, \sigma^{\mathrm{b}})$ contains at least one u-sequent). Since only the rules $\wedge R_k$, $\vee R$ and $\to R_1$ can be applied in $\Pi(\pi, \sigma^{\mathrm{b}})$, every leaf σ' of $\Pi(\pi, \sigma^{\mathrm{b}})$ has the form $[\Gamma^{\to}, \Gamma^{\mathrm{At}} \overset{\mathrm{b}}{\Rightarrow} H']$, where $H' \in \mathrm{Sf}(H)$; moreover, σ' is either an irreducible sequent (hence a leaf of π) or the conclusion of an application of $\to R_2$ (the only rule of **Rbu** which, read bottom-up, "unblocks" a b-sequent). Thus, π can be displayed as in Fig. 4. The sequents $\sigma_1^{\mathrm{u}}, \ldots, \sigma_n^{\mathrm{u}}$ ($n \geq 0$) are called the u-*successors* of σ^{b} in π, while the sequents $\tau_1^{\mathrm{b}}, \ldots, \tau_m^{\mathrm{b}}$ ($m \geq 0$) are the *i-successors* (irreducible successors) of σ^{b} in π. Let $\mathrm{d}(\pi)$ be the depth of π; if $\mathrm{d}(\pi) = 0$, then σ^{b} coincides with τ_1^{b}, hence σ^{b} has no u-successors and has itself as only i-successor.

Now, let us consider an **Rbu**-derivation π of an u-sequent σ^{u} having root rule $\mathcal{R} = \mathrm{S}_u^{\mathrm{At}}$ or $\mathcal{R} = \mathrm{S}_u^{\vee}$. Every premise σ' of $\mathcal{R}$ is a b-sequent and the subderivation

[2] The converse in general does not hold since the rule $\vee R$ of **Rbu** requires a b-sequent as conclusion.

$$\begin{array}{c}
\cdots \quad \dfrac{\begin{array}{c}\vdots\ \pi_i \\ \sigma_i^{\mathrm{u}} = [\Gamma^{\to}, \Gamma^{\mathrm{At}}, A_i \overset{\mathrm{u}}{\Rightarrow} B_i]\end{array}}{\sigma_i^{\mathrm{b}} = [\Gamma^{\to}, \Gamma^{\mathrm{At}} \overset{\mathrm{b}}{\Rightarrow} A_i \to B_i]} \to R_2 \quad \cdots \quad \dfrac{}{\tau_j^{\mathrm{b}} = [\Gamma^{\to}, \Gamma^{\mathrm{At}} \overset{\mathrm{b}}{\Rightarrow} H_j]} \mathrm{Irr} \quad \cdots \\
\vdots\ \Pi(\pi, \sigma^{\mathrm{b}}) \\
\sigma^{\mathrm{b}} = [\Gamma^{\to}, \Gamma^{\mathrm{At}} \overset{\mathrm{b}}{\Rightarrow} H]
\end{array}$$

where $i \in \{1, \ldots, n\}$, $j \in \{1, \ldots, m\}$, $n \geq 0$, $m \geq 0$, $n + m \geq 1$ and:

- the **Rbu**-tree $\Pi(\pi, \sigma^{\mathrm{b}})$ only contains b-sequents;
- π_i is an **Rbu**-derivation of σ_i^{u}.

Fig. 4. Structure of an **Rbu**-derivation π of $\sigma^{\mathrm{b}} = [\Gamma^{\to}, \Gamma^{\mathrm{At}} \overset{\mathrm{b}}{\Rightarrow} H]$

of π with root σ' has the structure shown in Fig. 4. The set of the u-*successors* of σ^{u} in π is the union of the sets of u-successors in π of the premises of $\mathcal{R}$; the set of the i-*successors* of σ^{u} in π is defined analogously. To display a proof π of this kind we use the concise notation of Fig. 5.

Example 3. Let us consider the **Rbu**-derivation π_1 in Ex. 2. The u-successors and i-successors are defined as follows:

u-sequent	**u-successors**	**i-successors**
σ_2	σ_4 , σ_{14} , σ_{19}	
σ_4	σ_6	σ_{12}
σ_8		σ_{10}
σ_{16}		σ_{17}

◊

Now we describe how to extract from an **Rbu**-derivation of an u-sequent σ^{u} a Kripke countermodel $\mathrm{Mod}(\pi)$ for σ^{u}. $\mathrm{Mod}(\pi)$ is defined by induction on $\mathrm{d}(\pi)$. By $\mathcal{K}^1(\rho, \Gamma^{\mathrm{At}})$ we denote the Kripke model $\mathcal{K} = \langle \{\rho\}, \{(\rho, \rho)\}, \rho, V \rangle$ consisting of only one world ρ such that $V(\rho) = \Gamma^{\mathrm{At}}$. Let $\mathcal{R}$ be the root rule of π.

(K1) If $\mathcal{R} = \mathrm{Irr}$, then $\mathrm{d}(\pi) = 0$ and $\sigma^{\mathrm{u}} = [\Gamma^{\mathrm{At}} \overset{\mathrm{u}}{\Rightarrow} H]$ (being σ^{u} irreducible, $\Gamma^{\to} = \emptyset$). We set $\mathrm{Mod}(\pi) = \mathcal{K}^1(\rho, \Gamma^{\mathrm{At}})$, with ρ any element.

(K2) Let $\mathcal{R}$ be different from Irr, $\mathrm{S}_u^{\mathrm{At}}$, $\mathrm{S}_u^{\vee}$ and let π' be the only immediate subderivation of π. Then, $\mathrm{Mod}(\pi) = \mathrm{Mod}(\pi')$.

(K3) Let $\mathcal{R}$ be $\mathrm{S}_u^{\mathrm{At}}$ or $\mathrm{S}_u^{\vee}$ and let π be displayed as in Fig. 5.
If $n = 0$, then $\mathcal{K}$ is the model $\mathcal{K}^1(\rho, \Gamma^{\mathrm{At}})$, with ρ any element.
Let $n > 0$ and, for every $i \in \{1, \ldots, n\}$, let $\mathrm{Mod}(\pi_i) = \langle P_i, \leq_i, \rho_i, V_i \rangle$. Without loss of generality, we can assume that the P_i's are pairwise disjoint. Let ρ be an element not in $\bigcup_{i \in \{1,\ldots,n\}} P_i$ and let $\mathcal{K} = \langle P, \leq, \rho, V \rangle$ be the model such that:
- $P = \{\rho\} \cup \bigcup_{i \in \{1,\ldots,n\}} P_i$;
- $\leq = \{ (\rho, \alpha) \mid \alpha \in P \} \cup \bigcup_{i \in \{1,\ldots,n\}} \leq_i$;
- $V(\rho) = \Gamma^{\mathrm{At}}$ and, for every $i \in \{1, \ldots, n\}$ and $\alpha \in P_i$, $V(\alpha) = V_i(\alpha)$.

Then $\mathrm{Mod}(\pi) = \mathcal{K}$. The model $\mathrm{Mod}(\pi)$ is represented in Fig. 5.

$$\pi \;=\; \dfrac{\begin{array}{c}\vdots\;\pi_1\\ \sigma^{\mathrm{u}}_1\end{array}\quad\dots\quad\begin{array}{c}\vdots\;\pi_n\\ \sigma^{\mathrm{u}}_n\end{array}\quad \tau^{\mathrm{b}}_1\dots\;\tau^{\mathrm{b}}_m}{\sigma^{\mathrm{u}} = [\Gamma^{\rightarrow}, \Gamma^{\mathrm{At}} \overset{\mathrm{u}}{\Rightarrow} H]}\;\mathcal{R}$$

– $\mathcal{R} \in \{\mathrm{S}^{\mathrm{At}}_u, \mathrm{S}^{\vee}_u\}$, $n \geq 0$, $m \geq 0$, $n + m \geq 1$.
– $\sigma^{\mathrm{u}}_1, \dots, \sigma^{\mathrm{u}}_n$ are all the u-successors of σ^{u} in π.
– π_i is an **Rbu**-derivation of σ^{u}_i $(1 \leq i \leq n)$.
– $\tau^{\mathrm{b}}_1, \dots, \tau^{\mathrm{b}}_m$ are all the i-successors of σ^{u} in π.

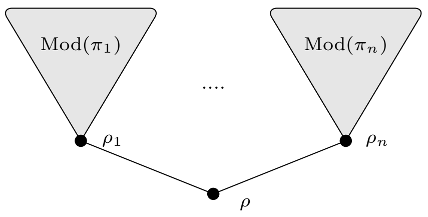
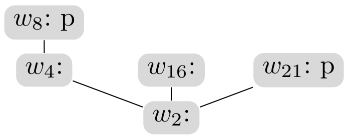


Fig. 5. An **Rbu**-derivation π with root rule $\mathrm{S}^{\mathrm{At}}_u$ or $\mathrm{S}^{\vee}_u$ and the model Mod(π)

Example 4. We show the Kripke model Mod(π_1) extracted from the **Rbu**-derivation π_1 of Ex. 2. The model is displayed as a tree with the convention that $w < w'$ if the world w is drawn below w'. For each w_i, we list the propositional variables in $V(w_i)$. We inductively define the models Mod(π_i) for every i such that $\sigma_i = [\Gamma_i \overset{\mathrm{u}}{\Rightarrow} H_i]$ is an u-sequent. At each step one can check that Mod(π_i), $\rho_i \triangleright \sigma_i$, where ρ_i is the root of Mod(π_i). Hence, Mod(π_1), $w_2 \nVdash S$ (Mod(π_1) is a countermodel for S).



– By Point (K3), since σ_8 has no u-successors (see Ex. 3), Mod(π_8) $= \mathcal{K}^1(w_8, \{p\})$. Similarly, Mod($\pi_{16}$) $= \mathcal{K}^1(w_{16}, \emptyset)$.
– Since σ_{21} is irreducible, by Point (K1) Mod(π_{21}) $= \mathcal{K}^1(w_{21}, \{p\})$.
– By Point (K2), Mod(π_6) = Mod(π_7) = Mod(π_8). Similarly, Mod(π_{14}) = Mod(π_{15}) = Mod(π_{16}) and Mod(π_{19}) = Mod(π_{20}) = Mod(π_{21}).
– By Point (K3), Mod(π_4) is obtained by extending with w_4 the model Mod(π_6) (indeed, σ_6 is the only u-successor of σ_4) and $V(w_4) = \Gamma_4 \cap \mathcal{V} = \emptyset$. Similarly, Mod($\pi_2$) is obtained by gluing on w_2 the models generated by the u-successors σ_4, σ_{14} and σ_{19} of σ_2 and $V(w_2) = \Gamma_2 \cap \mathcal{V} = \emptyset$.
– Finally, Mod(π_1) = Mod(π_2) by Point (K2). $\Diamond$

We prove the soundness of **Rbu**. Given an **Rbu**-tree π with root $[\Gamma^{\rightarrow}, \Gamma^{\mathrm{At}} \overset{\mathrm{b}}{\Rightarrow} H]$ and only containing b-sequents, every leaf of π has the form $[\Gamma^{\rightarrow}, \Gamma^{\mathrm{At}} \overset{\mathrm{b}}{\Rightarrow} H']$.

Lemma 3. *Let π be an* **Rbu**-*tree with root $\sigma^{\mathrm{b}} = [\Gamma^{\rightarrow}, \Gamma^{\mathrm{At}} \overset{\mathrm{b}}{\Rightarrow} H]$ and only containing* b-*sequents, let $\sigma^{\mathrm{b}}_1 = [\Gamma^{\rightarrow}, \Gamma^{\mathrm{At}} \overset{\mathrm{b}}{\Rightarrow} H_1]$,..., $\sigma^{\mathrm{b}}_n = [\Gamma^{\rightarrow}, \Gamma^{\mathrm{At}} \overset{\mathrm{b}}{\Rightarrow} H_n]$ be the leaves of π. Let $\mathcal{K} = \langle P, \leq, \rho, V \rangle$ be a Kripke model and $\alpha \in P$ such that:*

(H1) $\mathcal{K}, \alpha \nVdash H_i$, for every $i \in \{1, \dots, n\}$;
(H2) $\mathcal{K}, \alpha \Vdash Z$, for every $Z \in \Gamma^{\rightarrow} \cap \mathrm{Sf}(H)$;
(H3) $V(\alpha) = \Gamma^{\mathrm{At}}$.

Then, $\mathcal{K}, \alpha \nVdash H$.

Proof. By induction on d(π). If d(π) = 0, then $\sigma^{\mathrm{b}} = \sigma^{\mathrm{b}}_1$ and the assertion immediately follows by (H1). Let us assume that d(π) > 0 and let $\mathcal{R}$ be the root rule of π. Since both the conclusion and the premises of $\mathcal{R}$ are b-sequents, $\mathcal{R}$ is one of the rules $\wedge R_k$, $\vee R$ and $\to R_1$. The proof proceeds by cases on $\mathcal{R}$. The cases $\mathcal{R} \in \{\wedge R_k, \vee R\}$ immediately follow by the induction hypothesis.

If $\mathcal{R}$ is $\to R_1$, then $\sigma^{\mathrm{b}} = [\Gamma^{\to}, \Gamma^{\mathrm{At}} \overset{\mathrm{b}}{\Rightarrow} A \to B]$, the premise of $\mathcal{R}$ is $\sigma' = [\Gamma^{\to}, \Gamma^{\mathrm{At}} \overset{\mathrm{b}}{\Rightarrow} B]$ and, by the side condition, $\Gamma^{\to}, \Gamma^{\mathrm{At}} \vdash_{\mathcal{E}} A$. By induction hypothesis on the subderivation of π having root σ', we get $\mathcal{K}, \alpha \nVdash B$. We show that $\mathcal{K}, \alpha \Vdash A$. Let $\Gamma_A = (\Gamma^{\to} \cap \mathrm{Sf}(A)) \cup \Gamma^{\mathrm{At}}$. Since $\Gamma_A \cap \mathrm{Sf}(A) = (\Gamma^{\to} \cup \Gamma^{\mathrm{At}}) \cap \mathrm{Sf}(A)$ and $\Gamma^{\to}, \Gamma^{\mathrm{At}} \vdash_{\mathcal{E}} A$, by ($\mathcal{E}1$) we get $\Gamma_A \vdash_{\mathcal{E}} A$. By the hypothesis (H2) and (H3) of the lemma, it holds that $\mathcal{K}, \alpha \Vdash \Gamma_A$; by ($\mathcal{E}6$), we deduce $\mathcal{K}, \alpha \Vdash A$. Thus $\mathcal{K}, \alpha \Vdash A$ and $\mathcal{K}, \alpha \nVdash B$, which implies $\mathcal{K}, \alpha \nVdash A \to B$. □

Now, we show that the model $\mathrm{Mod}(\pi)$ is a countermodel for σ^{u}.

Theorem 3. *Let π be an* **Rbu**-*derivation of an* u-*sequent σ^{u} and let ρ be the root of* $\mathrm{Mod}(\pi)$. *Then* $\mathrm{Mod}(\pi), \rho \rhd \sigma^{\mathrm{u}}$.

Proof. By induction on $\mathrm{d}(\pi)$. If $\mathrm{d}(\pi) = 0$, then $\mathrm{Mod}(\pi)$ is defined as in (K1) and the assertion immediately follows.

Let $\mathrm{d}(\pi) > 0$ and let $\mathcal{R}$ be the root rule of π. If $\mathcal{R} \notin \{\mathrm{S}_u^{\mathrm{At}}, \mathrm{S}_u^{\vee}\}$, the assertion immediately follows by induction hypothesis (the case $\mathcal{R} = \to R_1$ requires ($\mathcal{E}6$)).

Let $\mathcal{R} = \mathrm{S}_u^{\vee}$ (the case $\mathcal{R} = \mathrm{S}_u^{\mathrm{At}}$ is similar). Let $\sigma^{\mathrm{u}} = [\Gamma^{\to}, \Gamma^{\mathrm{At}} \overset{\mathrm{u}}{\Rightarrow} H_0 \vee H_1]$ and let $\mathcal{K} = \langle P, \leq, \rho, V \rangle$ be the model $\mathrm{Mod}(\pi)$. By a secondary induction hypothesis on the structure of formulas, we prove that:

(B1) $\mathcal{K}, \rho \nVdash A$, for every $A \to B \in \Gamma^{\to}$;
(B2) $\mathcal{K}, \rho \Vdash A \to B$, for every $A \to B \in \Gamma^{\to}$;
(B3) $\mathcal{K}, \rho \nVdash H_0$ and $\mathcal{K}, \rho \nVdash H_1$.

To prove Point (B1), let $A \to B \in \Gamma^{\to}$. By definition of $\mathrm{S}_u^{\vee}$, π has an immediate subderivation π_A of $\sigma_A^{\mathrm{b}} = [\Gamma^{\to}, \Gamma^{\mathrm{At}} \overset{\mathrm{b}}{\Rightarrow} A]$ of the form (see Fig. 4):

$$
\begin{array}{c}
\begin{array}{ccccc}
\ldots &
\dfrac{\begin{array}{c}\vdots\ \pi_i \\ \sigma_i^{\mathrm{u}} = [\Gamma^{\to}, \Gamma^{\mathrm{At}}, A_i \overset{\mathrm{u}}{\Rightarrow} B_i]\end{array}}{\sigma_i^{\mathrm{b}} = [\Gamma^{\to}, \Gamma^{\mathrm{At}} \overset{\mathrm{b}}{\Rightarrow} A_i \to B_i]}\ {\to} R_2
& \ldots &
\dfrac{}{\tau_j^{\mathrm{b}} = [\Gamma^{\to}, \Gamma^{\mathrm{At}} \overset{\mathrm{b}}{\Rightarrow} H_j]}\ \mathrm{Irr}
& \ldots
\end{array} \\
\vdots\ \Pi(\pi_A, \sigma_A^{\mathrm{b}}) \\
\sigma_A^{\mathrm{b}} = [\Gamma^{\to}, \Gamma^{\mathrm{At}} \overset{\mathrm{b}}{\Rightarrow} A]
\end{array}
$$

We show that $\Pi(\pi_A, \sigma_A^{\mathrm{b}})$ meets the hypothesis (H1)–(H3) of Lemma 3 w.r.t. the root ρ of $\mathcal{K}$, so that we can apply the lemma to infer $\mathcal{K}, \rho \nVdash A$. We prove (H1). Let us assume $n \geq 1$ and let $i \in \{1, \ldots, n\}$; we must show that $\mathcal{K}, \rho \nVdash A_i \to B_i$. Since σ_i^{u} is an u-successor of σ^{u}, the root ρ_i of $\mathrm{Mod}(\pi_i)$ is an immediate successor of ρ in $\mathcal{K}$. By the main induction hypothesis $\mathrm{Mod}(\pi_i), \rho_i \rhd \sigma_i^{\mathrm{u}}$; this implies that $\mathrm{Mod}(\pi_i), \rho_i \Vdash A_i$ and $\mathrm{Mod}(\pi_i), \rho_i \nVdash B_i$. Since $\mathrm{Mod}(\pi_i)$ is a submodel of $\mathcal{K}$, we get $\mathcal{K}, \rho_i \Vdash A_i$ and $\mathcal{K}, \rho_i \nVdash B_i$, which implies $\mathcal{K}, \rho \nVdash A_i \to B_i$. Let $m \geq 1$ and $j \in \{1, \ldots, m\}$. By definition of τ_j^{b}, either $H_j = \bot$ or $H_j \in \mathcal{V} \setminus \Gamma^{\mathrm{At}}$; in both cases $\mathcal{K}, \rho \nVdash H_j$. This proves that hypothesis (H1) of Lemma 3 holds. To prove hypothesis (H2), let $Z \in \Gamma^{\to} \cap \mathrm{Sf}(A)$. Since $|Z| < |A \to B|$, by the secondary induction hypothesis on Point (B2) we get $\mathcal{K}, \rho \Vdash Z$. The hypothesis (H3) follows by the definition of V in $\mathcal{K}$. We can apply Lemma 3 to deduce $\mathcal{K}, \rho \nVdash A$, and this proves Point (B1).

We prove Point (B2). Let π and $\mathrm{Mod}(\pi)$ be as in Fig. 5 (with $H = H_0 \vee H_1$). Let $A \to B \in \Gamma^{\to}$ and let α be a world of $\mathcal{K}$ such that $\mathcal{K}, \alpha \Vdash A$; we show that $\mathcal{K}, \alpha \Vdash B$. By Point (B1), α is different from ρ. Thus, $n \geq 1$ and, for some $i \in \{1, \ldots, n\}$, α belongs to $\mathrm{Mod}(\pi_i)$. Let ρ_i be the root of $\mathrm{Mod}(\pi_i)$. By the main induction hypothesis, $\mathrm{Mod}(\pi_i), \rho_i \rhd \sigma_i^{\mathrm{u}}$; since $A \to B$ belongs to the left-hand side of σ_i^{u}, we get $\mathrm{Mod}(\pi_i), \rho_i \Vdash A \to B$, which implies $\mathcal{K}, \rho_i \Vdash A \to B$. Since $\rho_i \leq \alpha$ and $\mathcal{K}, \alpha \Vdash A$, we get $\mathcal{K}, \alpha \Vdash B$; thus $\mathcal{K}, \rho \Vdash A \to B$ and Point (B2) holds.

The proof of Point (B3) is similar to the proof of Point (B1), considering the immediate subderivations of π with root sequents $[\Gamma^{\to}, \Gamma^{\mathrm{At}} \overset{\mathrm{b}}{\Rightarrow} H_0]$ and $[\Gamma^{\to}, \Gamma^{\mathrm{At}} \overset{\mathrm{b}}{\Rightarrow} H_1]$. By Points (B2) and (B3) we conclude $\mathcal{K}, \rho \rhd \sigma^{\mathrm{u}}$. □

By Theorem 3, we get the soundness of **Rbu** stated in Theorem 2.

5 The Proof-Search Procedure

We show that, given an u-sequent σ^{u}, either a **Gbu**-derivation or an **Rbu**-derivation of σ^{u} can be built; from this, the completeness of **Gbu** follows. To this aim, we introduce the function `F` of Fig. 6. A sequent $[\Gamma \overset{l}{\Rightarrow} H]$ is in *normal form* if $l = \mathrm{b}$ implies $\Gamma = \Gamma^{\to}, \Gamma^{\mathrm{At}}$; given a sequent σ in normal form, `F`(σ) returns either a **Gbu**-derivation or an **Rbu**-derivation of σ. To construct a derivation, we use the auxiliary function `B`: given a calculus $\mathcal{C} \in \{\mathbf{Gbu}, \mathbf{Rbu}\}$, a sequent σ, a set $\mathcal{P}$ of $\mathcal{C}$-trees and a rule $\mathcal{R}$ of $\mathcal{C}$, `B`($\mathcal{C}, \sigma, \mathcal{P}, \mathcal{R}$) is the $\mathcal{C}$-tree having root sequent σ, root rule $\mathcal{R}$, and all the $\mathcal{C}$-trees in $\mathcal{P}$ as immediate subtrees.

Proof-search is performed by applying backward the rules of **Gbu**. For instance, the recursive call `F`($[A, B, \Gamma' \overset{\mathrm{u}}{\Rightarrow} H]$) at line 3 corresponds to the backward application of the rule $\wedge L$ to $\sigma = [A \wedge B, \Gamma' \overset{\mathrm{u}}{\Rightarrow} H]$; according to the outcome, at lines 4–5 a **Gbu**-derivation or an **Rbu**-derivation of σ with root rule $\wedge L$ is built. We remark that the input sequent of `F` must be in normal form; to guarantee that the recursive invocations are sound, the rules $\vee R_k$ and $\to L$, generating b-sequents, can be backward applied to $[\Gamma \overset{\mathrm{u}}{\Rightarrow} H]$ only if Γ has the form $\Gamma^{\to}, \Gamma^{\mathrm{At}}$.

To save space, some instructions are written in a high-level compact form (see, e.g., line 8); the rules used in lines 1 and 32 are defined as follows:

$$\mathcal{R}_{\mathrm{ax}}([\Gamma \overset{l}{\Rightarrow} H]) = \begin{cases} \bot L & \text{if } \bot \in \Gamma \\ \mathrm{Id} & \text{otherwise} \end{cases} \qquad \mathcal{R}_{\mathrm{s}}([\Gamma \overset{l}{\Rightarrow} H]) = \begin{cases} \vee R & \text{if } l = \mathrm{b} \\ \mathrm{S}_u^{\mathrm{At}} & \text{if } l = \mathrm{u} \text{ and } H \in \mathcal{V} \\ \mathrm{S}_u^{\vee} & \text{otherwise} \end{cases}$$

By $\|\sigma\|$ we denote the maximal length of a **Gbu**-branch starting from σ (by Prop. 1, $\|\sigma\|$ is finite). Note that, whenever a recursive call `F`(σ') occurs along the computation of `F`(σ), it holds that $\|\sigma'\| < \|\sigma\|$.

In the next lemma we prove the correctness of `F`.

```
   Precondition : $\sigma$ is in normal form ($l = \mathrm{b}$ implies $\Gamma = \Gamma^{\to}, \Gamma^{\mathrm{At}}$)
 1 if $\bot \in \Gamma$ or $H \in \Gamma$ then return B(Gbu, $\sigma$, $\emptyset$, $\mathcal{R}_{\mathrm{ax}}(\sigma)$) // $\mathcal{R}_{\mathrm{ax}}(\sigma)$ is $\bot L$ or Id
 2 else if $\sigma = [A \wedge B, \Gamma' \overset{\mathrm{u}}{\Rightarrow} H]$ where $\Gamma' = \Gamma \setminus \{A \wedge B\}$ then
 3     $\pi' \leftarrow$ F($[A, B, \Gamma' \overset{\mathrm{u}}{\Rightarrow} H]$)
 4     if $\pi'$ is a Gbu-tree then return B(Gbu, $\sigma$, $\{\pi'\}$, $\wedge L$)
 5     else return B(Rbu, $\sigma$, $\{\pi'\}$, $\wedge L$)
 6 else if $\sigma = [A_0 \vee A_1, \Gamma' \overset{\mathrm{u}}{\Rightarrow} H]$ where $\Gamma' = \Gamma \setminus \{A_0 \vee A_1\}$ then
 7     $\pi_0 \leftarrow$ F($[A_0, \Gamma' \overset{\mathrm{u}}{\Rightarrow} H]$) ,  $\pi_1 \leftarrow$ F($[A_1, \Gamma' \overset{\mathrm{u}}{\Rightarrow} H]$)
 8     if $\exists k \in \{0,1\}$ s.t. $\pi_k$ is an Rbu-tree then return B(Rbu, $\sigma$, $\{\pi_k\}$, $\vee L_k$)
 9     else return B(Gbu, $\sigma$, $\{\pi_0, \pi_1\}$, $\vee L$)
10 else if $\sigma = [\Gamma \overset{l}{\Rightarrow} A \to B]$ then
11     if $\Gamma \vdash_{\mathcal{E}} A$ then $\pi' \leftarrow$ F($[\Gamma \overset{l}{\Rightarrow} B]$) ,  $k \leftarrow 1$
12     else $\pi' \leftarrow$ F($[A, \Gamma \overset{\mathrm{u}}{\Rightarrow} B]$) ,  $k \leftarrow 2$
13     if $\pi'$ is a Gbu-tree then return B(Gbu, $\sigma$, $\{\pi'\}$, $\to R_k$)
14     else return B(Rbu, $\sigma$, $\{\pi'\}$, $\to R_k$)
15 else if $\sigma = [\Gamma \overset{l}{\Rightarrow} H_0 \wedge H_1]$ then
16     $\pi_0 \leftarrow$ F($[\Gamma \overset{l}{\Rightarrow} H_0]$) ,  $\pi_1 \leftarrow$ F($[\Gamma \overset{l}{\Rightarrow} H_1]$)
17     if $\exists k \in \{0,1\}$ s.t. $\pi_k$ is an Rbu-tree then return B(Rbu, $\sigma$, $\{\pi_k\}$, $\wedge R_k$)
18     else return B(Gbu, $\sigma$,$\{\pi_0, \pi_1\}$, $\wedge R$)
19 // Here $\sigma = [\Gamma^{\to}, \Gamma^{\mathrm{At}} \overset{l}{\Rightarrow} H]$, where $H = \bot$ or $H \in \mathcal{V} \setminus \Gamma^{\mathrm{At}}$ or $H = H_0 \vee H_1$
20 else if ($l = \mathrm{u}$ and $\Gamma^{\to} \neq \emptyset$) or $H = H_0 \vee H_1$ then
21     Refs $\leftarrow \emptyset$ // set of Rbu-trees
22     if $H = H_0 \vee H_1$ then
23         $\pi_0 \leftarrow$ F($[\Gamma \overset{\mathrm{b}}{\Rightarrow} H_0]$) ,  $\pi_1 \leftarrow$ F($[\Gamma \overset{\mathrm{b}}{\Rightarrow} H_1]$)
24         if $\exists k \in \{0,1\}$ s.t. $\pi_k$ is a Gbu-tree then return B(Gbu, $\sigma$, $\{\pi_k\}$, $\vee R_k$)
25         else Refs $\leftarrow$ Refs $\cup$ { $\pi_0$, $\pi_1$ }
26     if $l = \mathrm{u}$ then
27         foreach $A \to B \in \Gamma^{\to}$ do
28             $\pi_A \leftarrow$ F($[\Gamma^{\to}, \Gamma^{\mathrm{At}} \overset{\mathrm{b}}{\Rightarrow} A]$), $\pi_B \leftarrow$ F($[B, \Gamma^{\to} \setminus \{A \to B\}, \Gamma^{\mathrm{At}} \overset{\mathrm{u}}{\Rightarrow} H]$)
29             if $\pi_B$ is an Rbu-tree then return B(Rbu, $\sigma$, $\{\pi_B\}$, $\to L$)
30             else if $\pi_A$ is a Gbu-tree then return B(Gbu, $\sigma$, $\{\pi_A, \pi_B\}$, $\to L$)
31             else Refs $\leftarrow$ Refs $\cup$ { $\pi_A$ }
32     return B(Rbu, $\sigma$, Refs, $\mathcal{R}_{\mathrm{s}}(\sigma)$) // $\mathcal{R}_{\mathrm{s}}(\sigma)$ is $\vee R$ or $\mathrm{S}_u^{\mathrm{At}}$ or $\mathrm{S}_u^{\vee}$
33 // Here ($H = \bot$ or $H \in \mathcal{V} \setminus \Gamma^{\mathrm{At}}$) and ($l = \mathrm{b}$ or $\Gamma^{\to} = \emptyset$)
34 else return B(Rbu,$\sigma$, $\emptyset$, Irr)
```

Fig. 6. $\mathrm{F}(\sigma = [\Gamma \overset{l}{\Rightarrow} H])$

Lemma 4. *Let σ be a sequent in normal form. Then,* F(σ) *returns either a* **Gbu**-*derivation or an* **Rbu**-*derivation of σ.*

Proof. By induction on $\|\sigma\|$. If $\|\sigma\| = 1$, F(σ) does not execute any recursive invocation and the computation ends at line 1 or at line 34. In the former case, a **Gbu**-derivation of σ is returned. In the latter case, since σ is in normal form and none of the conditions at lines 1, 2, 6, 10 15, 20 holds, the sequent σ is irreducible and the tree built at line 34 is an **Rbu**-derivation of σ.

Let $\|\sigma\| > 1$. Whenever a recursive call F(σ') occurs, we have that $\|\sigma'\| < \|\sigma\|$ and σ' is in normal form, hence the induction hypothesis applies to F(σ'). Using this, one can easily show that the arguments of function B are correctly instantiated. We only analyse some cases.

Let us assume that one of the return instructions at lines 8–9 is executed. By induction hypothesis, for every $k \in \{0,1\}$, π_k is either a **Gbu**-proof or an **Rbu**-derivation of $\sigma_k = [A_k, \Gamma' \overset{\mathrm{u}}{\Rightarrow} H]$. If, for some k, π_k is an **Rbu**-derivation of σ_k, then the **Rbu**-tree returned at line 8 is an **Rbu**-derivation of σ. Otherwise, both π_0 and π_1 are **Gbu**-derivations, hence the value returned at line 9 is a **Gbu**-derivation of σ.

Let us assume that `F(`σ`)` ends at line 32; in this case σ satisfies the conditions at lines 19 and 20. If $l = \mathrm{b}$, then $H = H_0 \vee H_1$. Since the condition at line 24 is false, we have $\mathsf{Refs} = \{\pi_0, \pi_1\}$ and, by induction hypothesis, both π_0 and π_1 are **Rbu**-derivations. Accordingly, the value returned at line 32 is an **Rbu**-derivation of σ with root rule $\mathcal{R}_{\mathrm{s}}(\sigma) = \vee R$. Let $l = \mathrm{u}$ and let us assume that $H = \bot$ or $H \in \mathcal{V} \setminus \Gamma$. In this case $\sigma = [\Gamma^{\rightarrow}, \Gamma^{\mathrm{At}} \overset{\mathrm{u}}{\Rightarrow} H]$ and the set Refs contains an **Rbu**-tree π_A of $\sigma_A = [\Gamma^{\rightarrow}, \Gamma^{\mathrm{At}} \overset{\mathrm{b}}{\Rightarrow} A]$ for every $A \rightarrow B \in \Gamma^{\rightarrow}$. By induction hypothesis, π_A is an **Rbu**-derivation of σ_A, hence line 32 returns an **Rbu**-derivation of σ with root rule $\mathcal{R}_{\mathrm{s}}(\sigma) = \mathrm{S}_u^{\mathrm{At}}$. The subcase ($l = \mathrm{u}$ and $H = H_0 \vee H_1$) is similar. □

Finally, we get the completeness of **Gbu**:

Theorem 4. *An* u*-sequent* σ^{u} *is provable in* **Gbu** *iff* σ^{u} *is not refutable.*

Proof. The $\Rightarrow$-statement follows by the soundness of **Gbu**. Conversely, let σ^{u} be not refutable. Then, there is no **Rbu**-derivation π of σ^{u}; otherwise, by Theorem 3, from π we could extract a countermodel for σ^{u}. Since σ^{u} is in normal form, by Lemma 4 the call `F(`σ^{u}`)` returns a **Gbu**-derivation of σ^{u}. □

6 Conclusions and Future Works

We have presented **Gbu**, a terminating sequent calculus for intuitionistic propositional logic. **Gbu** is a notational variant of **G3i**, where sequents are labelled to mark the right-focused phase. Note that focusing techniques reduce the search space limiting the use of contraction, but they do not guarantee termination of proof-search (see, e.g., the right-focused calculus *LJQ* [2]). To get this, one has to introduce extra machinery. An efficient solution is loop-checking implemented by history mechanisms [6,7]. Here we propose a different approach, based on an evaluation relation defined on sequents. Histories require space to store the right formulas already used so to direct and possibly stop the proof-search. Instead, we have to compute evaluation relations when right-implication is treated. We remark that, with an appropriate implementation of the involved data structures (see [4]), the evaluation relation $\vdash_{\hat{\mathcal{E}}}$ defined in Section 2 can be computed in time linear in the size of the arguments. Hence, we get by means of computation what history mechanisms get using memory. Although a strict comparison is hard, to stress the difference between the two approaches we provide an example where **Gbu** outperforms history-based calculi. Let $\sigma = [\Gamma^{\rightarrow} \overset{\mathrm{u}}{\Rightarrow} \bot]$, where $\Gamma^{\rightarrow} = \{p_1 \rightarrow \bot, \ldots, p_n \rightarrow \bot\}$ and the p_i's are distinct propositional variables. The only rule that can be used to derive σ is $\rightarrow L$. For every $p_i \rightarrow \bot$ chosen as main formula, the right-hand premise is provable in **Gbu**, while the left-hand premise $\sigma_i^{\mathrm{b}} = [\Gamma^{\rightarrow} \overset{\mathrm{b}}{\Rightarrow} p_i]$ is not. Thus, we have a backtrack point which forces the application of $\rightarrow L$ in all possible ways. Being σ_i^{b} blocked, the unprovability of σ_i^{b} is immediately certified. With the calculi in [7], the search process is similar, but to assert the unprovability of $[\Gamma^{\rightarrow} \Rightarrow p_i]$ one has to chain up to n applications of $\rightarrow L$ and build an history set containing all the p_i's.

Differently from the history mechanisms, **Gbu** only exploits the information in the left-hand side of a sequent. We are investigating the use of more expressive evaluation relations to better grasp the information conveyed by a sequent and further reduce the search space. Finally, we aim to extend the use of these techniques to other logics having a Kripke semantics.

References

1. Chagrov, A., Zakharyaschev, M.: Modal Logic. Oxford University Press (1997)
2. Dyckhoff, R., Lengrand, S.: LJQ: A Strongly Focused Calculus for Intuitionistic Logic. In: Beckmann, A., Berger, U., Löwe, B., Tucker, J.V. (eds.) CiE 2006. LNCS, vol. 3988, pp. 173–185. Springer, Heidelberg (2006)
3. Ferrari, M., Fiorentini, C., Fiorino, G.: Contraction-free linear depth sequent calculi for intuitionistic propositional logic with the subformula property and minimal depth counter-models. Journal of Automated Reasoning 51(2), 129–149 (2013)
4. Ferrari, M., Fiorentini, C., Fiorino, G.: Simplification rules for intuitionistic propositional tableaux. ACM Transactions on Computational Logic (TOCL) 13(2), 14:1–14:23 (2012)
5. Gabbay, D.M., Olivetti, N.: Goal-Directed Proof Theory. Springer (2000)
6. Heuerding, A., Seyfried, M., Zimmermann, H.: Efficient loop-check for backward proof search in some non-classical propositional logics. In: Miglioli, P., Moscato, U., Ornaghi, M., Mundici, D. (eds.) TABLEAUX 1996. LNCS, vol. 1071, pp. 210–225. Springer, Heidelberg (1996)
7. Howe, J.M.: Two loop detection mechanisms: A comparision. In: Galmiche, D. (ed.) TABLEAUX 1997. LNCS, vol. 1227, pp. 188–200. Springer, Heidelberg (1997)
8. 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. 217–231. Springer, Heidelberg (1998)
9. Pinto, L., Dyckhoff, R.: Loop-free construction of counter-models for intuitionistic propositional logic. In: Behara, M., et al. (eds.) Symposia Gaussiana, Conference A, pp. 225–232. Walter de Gruyter, Berlin (1995)
10. Troelstra, A.S., Schwichtenberg, H.: Basic Proof Theory. Cambridge Tracts in Theoretical Computer Science, vol. 43. Cambridge University Press (1996)

Model Checking General Linear Temporal Logic

Tim French, John M^{c}Cabe-Dansted, and Mark Reynolds*

School of Computer Science and Software Engineering
University of Western Australia
Crawley, Perth Western Australia 6009
{tim.french,john.mccabe-dansted,mark.reynolds}@uwa.edu.au

Abstract. We address the problem of model checking temporal formulas with Until and Since over general linear time. General linear time allows us to go beyond discrete natural numbers time and consider temporal models with continuous properties for applications such as distributed systems and message passing, and to even more unrestricted contexts thrown up by natural language semantics and AI modelling of human reasoning. We use a recently formalised compositional language that is capable of describing models of any satisfiable formula. Given a model described in this model expression language and a temporal logic formula, the algorithm decides whether the formula is satisfied in the model. As for standard natural-numbers time model checkers, the computational time required by the algorithm is linear in the size of the model expression. We illustrate this result briefly with some examples including a system that needs to interact with a environment exhibiting Zeno behaviours.

1 Introduction

Traditionally, temporal logics have used a discrete model of time [16]. However, many systems have to interact asynchronously with an external environment or other components, and a continuous model of time may be more appropriate for specifying and reasoning about their behaviour. For applications involving natural language semantics, philosophical reasoning and analysing human reasoning, even stranger models of time may need to be contemplated. Here we only assume that time is linear and we investigate what it means to undertake the model-checking task in this context. As well as developing a very general notion of temporal model-checking for such applications, we also aim to eventually see if such a general procedure can perhaps be modified easily to give specific model-checking techniques for particular flows of time.

We use the well-established propositional temporal language with Until and Since introduced in [12]. This gives us the temporal logic, US/L, of this language over the class of all linear flows of time. By restricting our attention to particular sets of linear flows of time we can define other logics with the general language: fixing on real numbers time gives us the useful continuous-time logic

* The work was partially supported by the Australian Research Council.

RTL[17], while fixing on the natural numbers gives us a temporal logic easily inter-translatable with standard PLTL with past.

Model checking is an important task in the verification of systems. It is just a formal version of the task of checking whether a system does obey a specification. Typical existing continuous time model checking algorithms take as input a timed graph such as a timed automata [1,2] and a temporal formula. In these graphs, state transitions are labelled with (possibly zero) durations. By considering all runs of such an automaton, we find a corresponding set of temporal models—the set of possible behaviours of the system. Model checking the system equates to deciding whether the given formula is satisfied by all the models in this set. Our version of model checking is similar but we have a different way of specifying the set of models.

The two main differences between existing model checking approaches and ours, as presented here, are as follows. The timed automata approaches usually deal with metric information (limits on the durations between events) while we do not. However, we do handle models exhibiting quite general temporal behaviours while timed automata approaches have limitations in dealing with the infinitely short and dense types of behaviours. In particular, we will examine what are known as Zeno behaviours: when there are infinite changes of state in a bounded interval of time. In this paper we only briefly describe some toy examples of such situations. However, it is becoming increasingly recognised that they do pose an issue for formal approaches to hybrid systems and the like [14]. A cycle of zero length transitions for a timed automaton provides a hint that Zeno behaviour could occur; however, it does not allow us to specify what happens after an accumulation point. Similar but more complex are properties of fractal signals or mathematical rules such as "between each pair of rationals there is an irrational".

To allow representation of a full range of models of US/L, [4] presented a compositional language of model expressions (MEs), following pioneering work by [13,3,18]. Structures are built as sequences of simpler structures via combinations of four well-known operations which we re-visit below. The motivation for the development of the model expression language was to provide a simple way of finitely describing general linear structures, objects which are usually infinite and complicated, in a simple standard formal way, and allowing enough expressibility so that any satisfiable formula of US/L has a model finitely representable in this language [3,4]. The recent work in [4] goes further and gives an efficient procedure for synthesising the model expression of a model of any given satisfiable formula. However, having finite representations has also opened the possibility for an algorithm for model checking formulas against those representations.

The idea of a model checking procedure for US/L against the model expressions of [4] was mooted in [5] but this paper is the first detailed account of an algorithm with a proof of correctness and analysis of its performance. An implementation of the model checker is available online [15].

In the next two sections, we define the temporal logic US/L and the formal notation for describing its models. In section 4, we present a few toy examples of model checking with these languages. Section 5 presents the model checking procedure in detail and we prove it correct in the next section. Section 7 briefly considers the complexity of the algorithm and reports on some experiments in empirical tests of running times on large inputs. Section 8 concludes.

2 The Logic

Fix a countable set $\mathbf{L}$ of atoms. Here, frames $(T, <)$, or flows of time, will be irreflexive linear orders. Structures $\mathbf{T} = (T, <, h)$ will have a frame $(T, <)$ and a valuation h for the atoms, i.e. for each atom $p \in \mathbf{L}$, $h(p) \subseteq T$.

The well-formed formulas of the language $L(U, S)$ are generated by the 2-place connectives U and S along with classical $\neg$ and $\wedge$. That is, we define the set of formulas recursively to contain the atoms and for formulas α and β we include $\neg\alpha$, $\alpha \wedge \beta$, $U(\alpha, \beta)$ and $S(\alpha, \beta)$. (Note the prefix use of U and S as in [3,8,12].)

The logic US/L consists of formulas of $L(U, S)$ evaluated at points in (linear) structures $\mathbf{T} = (T, <, h)$. We write $\mathbf{T}, x \models \alpha$ when α is true at the point $x \in T$. This is defined recursively as follows. Suppose that we have defined the truth of formulas α and β at all points of $\mathbf{T}$. Then for all points x:

$\mathbf{T}, x \models p$ iff $x \in h(p)$, for p atomic;
$\mathbf{T}, x \models \neg\alpha$ iff $\mathbf{T}, x \not\models \alpha$;
$\mathbf{T}, x \models \alpha \wedge \beta$ iff both $\mathbf{T}, x \models \alpha$ and $\mathbf{T}, x \models \beta$;
$\mathbf{T}, x \models U(\alpha, \beta)$ iff there is $y > x$ in T such that $\mathbf{T}, y \models \alpha$ and for all $z \in T$ such that $x < z < y$ we have $\mathbf{T}, z \models \beta$; and
$\mathbf{T}, x \models S(\alpha, \beta)$ iff there is $y < x$ in T such that $\mathbf{T}, y \models \alpha$ and for all $z \in T$ such that $y < z < x$ we have $\mathbf{T}, z \models \beta$.

The logic is discussed more fully in other papers, for example: [19,21]. See those references for investigations of the "strict" versus "non-strict" connectives, infix versus postfix operators, various abbreviations, etc.

We use the following abbreviations in illustrating the logic: $F\alpha = U(\alpha, \top)$, "alpha will be true (sometime in the future)" $G\alpha = \neg F(\neg\alpha)$, "alpha will always hold (in the future)"; and their mirror images P and H. We also use the standard classical abbreviations $\alpha \vee \beta = \neg(\alpha \wedge \beta)$, "alpha or beta" $\alpha \rightarrow \beta = \neg\alpha \vee \beta$, "alpha implies beta", $\alpha \leftrightarrow \beta = (\alpha \rightarrow \beta) \wedge (\beta \rightarrow \alpha)$ "alpha iff beta".

3 Building Structures

As in [4], we define a notation which allows the description of a temporal structure in terms of simple basic structures via a small number of ways of putting structures together to form larger ones.

The general idea is simple: using singleton structures (the flow of time is one point), we build up to more complex structures by the recursive application of four operations. They are:

– concatenation of two structures, consisting of one followed by the other;
– ω repeats of some structure laid end to end towards the past;
– ω repeats laid end to end towards the future;
– and making a densely thorough *shuffle* of copies from a finite set of structures.

These operations are well-known from the study of linear orders (see, e.g. , [3]).

Model Expressions are an abstract syntax for defining models that are constructed using the following set of primitive operators:

$$\mathcal{I} ::= \ a \mid \lambda \mid \mathcal{I} + \mathcal{J} \mid \overleftarrow{\mathcal{I}} \mid \overrightarrow{\mathcal{I}} \mid \langle \mathcal{I}_0, \ldots, \mathcal{I}_n \rangle$$

where $a \in \Sigma$, and Σ is some alphabet[1]. We refer to these operators, respectively, as *a letter*, *the empty order*, *concatenation*, *lead*, *trail*, and *shuffle.*

Definition 1. *[Correspondence] Given $\Sigma = 2^{\mathbf{L}}$, a model expression $\mathcal{I}$* corresponds *to a structure as follows:*

– *λ is the empty sequence and corresponds to the (pseudo)frame[2] $(\emptyset, <, h)$ where $<$ and h are empty relations.*
– *a corresponds to any single point structure $(\{x\}, <, h)$ where $<$ is the empty relation and $h(p) = \{x\}$ if and only if $p \in a$.*
– *$\mathcal{I} + \mathcal{J}$ corresponds to a structure $(T, <, h)$ if and only if T is the disjoint union of two sets U and V where $\forall u \in U$, $\forall v \in V$, $u < v$ and $\mathcal{I}$ corresponds to $(U, <^U, h^U)$ and $\mathcal{J}$ corresponds to $(V, <^V, h^V)$. ($<^U, h^U$ refers to the restriction of the relations $<$ and h to apply only to elements of U).*
– *$\overleftarrow{\mathcal{I}}$ corresponds to the structure $(T, <, h)$ if and only if T is the disjoint union of sets $\{U_i | i \in \omega\}$ where for all i, for all $u \in U_i$, for all $v \in U_{i+1}$, $v < u$, and $\mathcal{I}$ corresponds to $(U_i, <^{U_i}, h^{U_i})$.*
– *$\overrightarrow{\mathcal{I}}$ corresponds to the structure $(T, <, h)$ if and only if T is the disjoint union of sets $\{U_i | i \in \omega\}$ where for all i, for all $u \in U_i$, for all $v \in U_{i+1}$, $u < v$, and $\mathcal{I}$ corresponds to $(U_i, <^{U_i}, h^{U_i})$.*
– *$\langle \mathcal{I}_0, \ldots, \mathcal{I}_n \rangle$ corresponds to the structure $(T, <, h)$ if and only if T is the disjoint union of sets $\{U_i | i \in \mathbb{Q}\}$ where*
  1. *for all $i \in \mathbb{Q}$ $(U_i, <^{U_i}, h^{U_i})$ corresponds to some $\mathcal{I}_j$ for $j \leq n$,*
  2. *for every $j \leq n$, for every $a \neq b \in \mathbb{Q}$, there is some k in the open interval (a, b) where $\mathcal{I}_j$ corresponds to $(U_k, <^{U_k}, h^{U_k})$,*
  3. *for every $a < b \in \mathbb{Q}$ for all $u \in U_a$, for all $v \in U_b$, $u < v$.*

For convenience we define "$\mathcal{I}$ corresponds to $\mathbf{T}$" to be equivalent to "$\mathbf{T}$ corresponds to $\mathcal{I}$". The MEs are important for modelling US/L formulas. Every satisfiable US/L formula ϕ has an ME $\mathcal{I}$ such that a structure $\mathbf{T}$ that corresponds to $\mathcal{I}$ will satisfy ϕ, yet MEs are minimal in that excluding any operator (except λ) will render them unable to represent models for some US/L formulas.

We will give an illustration of the non-trivial operations below. The *lead* operation, $\mathcal{I} = \overleftarrow{\mathcal{J}}$ has ω submodels, each corresponding to $\mathcal{J}$, and each preceding the last, as illustrated in Fig. 1.

[1] Typically, we will let $\Sigma = 2^{\mathbf{L}}$ so the letter indicates the atoms true at a point.

[2] Of course, the empty pseudo-frame is not counted as a frame and we do not allow empty structures.

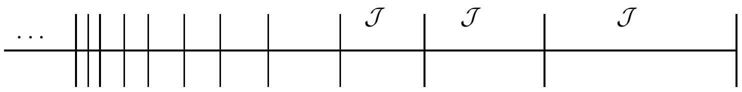


Fig. 1. The lead operation, where $\mathcal{I} = \overleftarrow{\mathcal{J}}$

The *trail* operator is the mirror image of *lead*, whereby $\mathcal{I} = \overrightarrow{\mathcal{J}}$ has ω structures, each corresponding to $\mathcal{J}$ and each proceeding the earlier structures.

The *shuffle* operator is harder to represent with a diagram. The model expression $\mathcal{I} = \langle \mathcal{I}_1, \dots \mathcal{I}_n \rangle$ corresponds to a dense, thorough mixture of intervals corresponding to $\mathcal{I}_1$, ..., $\mathcal{I}_n$, without endpoints. We define the shuffle operation using the rationals, $\mathbb{Q}$ as they are a convenient order with the required properties.

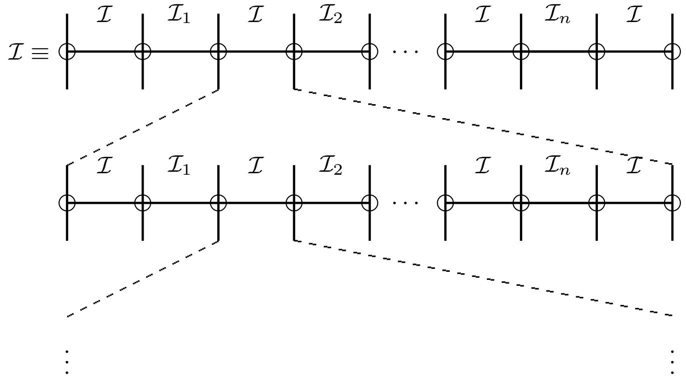


Fig. 2. The shuffle operation, where $\mathcal{I} = \langle \mathcal{I}_1, \dots, \mathcal{I}_n \rangle$

The definition of model expressions is not deterministic, as the construct for the shuffle $\langle \mathcal{I}_1, \dots, \mathcal{I}_n \rangle$ does not specify how the structures corresponding to $\mathcal{I}_1, \dots, \mathcal{I}_n$ are mapped to $\mathbb{Q}$. This is inconsequential, and as long as the mapping is dense for each i from 1 to n, the resulting structures will be isomorphic [4].

It is important to note that such a model expression language is capable of expressing a model for any satisfiable formula.

Theorem 1. *If ϕ is a satisfiable formula of $L(U, S)$ then there is an ME $\mathcal{I}$ and a structure $\mathbf{T}$, and point x such that $\mathbf{T}$ corresponds to $\mathcal{I}$, and $\mathbf{T}, x \vDash \phi$. [4]*

The following proposition follows trivially from the definition of correspondence.

Proposition 1. *Given an ME $\mathcal{I}$ that corresponds to some structure $(T, <, h)$ it is the case that $h(p) = T$ iff every letter within $\mathcal{I}$ contains p. Likewise $h(p)$ is non-empty iff some letter within $\mathcal{I}$ contains p.*

In proofs that use several restrictions of the same structure, we will find the following abbreviations useful.

Definition 2. *We define* $\mathbf{T}_U = (U, <_U, h_U)$ *to be the* restriction *of a structure* $\mathbf{T} = (T, <, h)$ *to a set* $U \subseteq T$*; formally for each* x, y *in* U *we have* $x <_U y$ *iff* $x < y$*, and for each* $p \in \mathbf{L}$ *we have* $h_U(p) = h(p) \cap U$*.*

Lemma 1. *Say that an ME* $\mathcal{K}$ *is of the form* $\overleftarrow{\mathcal{I}}$*,* $\overrightarrow{\mathcal{I}}$ *or* $\langle \mathcal{I}_0, \ldots, \mathcal{I}_n \rangle$ *and corresponds to a structure* $\mathbf{T} = (T, <, h)$*. Take the sets* U_i *from the definition of correspondence of* $\mathcal{K}$ *to* $\mathbf{T}$ *and let* S *be* $\bigcup_{i>j} U_i$ *for some* $j \in \mathbb{Q}$*. Note that for the cases of* $\overleftarrow{\mathcal{I}}$*,* $\overrightarrow{\mathcal{I}}$ *we have* $i \in \omega$ *and so in these cases we could equivalently require that* $j \in \omega$*. Then* $\mathcal{K}$ *also corresponds to* $\mathbf{T}_S$*.*

Proof. This proposition is trivial for $\overleftarrow{\mathcal{I}}$ and $\overrightarrow{\mathcal{I}}$. For $\langle \mathcal{I}_0, \ldots, \mathcal{I}_n \rangle$ the function $f(i) = \frac{1}{i-j}$ gives us an order preserving bijection from rationals greater than j to all rationals. We let $V_{f(i)} = U_i$ for each $i > j$ and we see that $\mathcal{K}$ corresponds to $\bigcup_{i \in \mathbb{Q}} V_i = \bigcup_{i>j} U_i$. □

We now define presatisfaction. Intuitively $\mathbf{T} = (T, <, h)$ *presatisfies* $U(p, q)$ means $U(p, q)$ would be true at a point immediately prior to all points in T.

Definition 3. *We say that a structure* $\mathbf{T} = (T, <, h)$ presatisfies $U(p, q)$ *iff there exists a* $y \in h(p)$ *such that for each* $x \in T$ *we have* $x < y \to x \in h(q)$*.*

4 Examples

In this section we will briefly discuss a paradox discovered by the Ancient Greek philosopher Zeno, and a system that interacts with an environment that has Zeno properties. Zeno properties are not normally formally defined, but for clarity we can loosely define a Zeno property to be a property that only occurs if an infinite number of state changes occur within a bounded amount of time [14].

Zeno is reputed [11] to have presented a number of famous paradoxes. These paradoxes have a common theme, that to achieve motion we need to move past an infinite number of segments. The most famous is the Achilles and the Tortoise paradox, where each time Achilles catches up to where the tortoise was, the Tortoise has moved on. No matter how often Achilles catches up, the Tortoise will still be ahead. We will formalise a simpler paradox from Zeno in US/L.

Say we are walking towards a wall. We will have to halve the distance between us and the wall in infinite number of times, yet we eventually reach the wall.

We see that: we will halve the distance; whenever we have halved the distance, we will halve the distance again, but only after a period of not halving the distance; once we have reached the wall we will not halve the distance anymore; and finally, we reach the wall. Where h represents "we have halved the distance" and r represents "we have reached the wall". We represent this using the formula:

$$Fh \wedge G(h \to U(h, \neg h)) \wedge G(r \to G\neg h) \wedge Fr \ .$$

We see that this does not cause a contradiction as this formula is satisfied at the leftmost point x of any structure $\mathbf{T}$ corresponding to $\{h\} + \overrightarrow{\emptyset} + \{r\}$.[3]

Although a system will generally only be able to perform finitely many operations per second it may interact with an environment that exhibits Zeno properties. Perhaps the most famous example of such an environment is the Bouncing Ball example, where at each bounce we assume the ball loses half its velocity. Then each bounce takes half the time of the last, and that an infinite number of bounces occur before the ball finally stops, which requires only finite time. We do not claim that any real system works precisely this way, as there are any number of other forces at play, particularly as the height of the "bounces" approaches the Planck length. Never-the-less useful models of real world behaviour admit Zeno properties, and adding real world limitations preventing Zeno behaviour may make the model more complex without increasing its usefulness.

At some sporting event, it may be that a player is awarded a point if the ball bounces twice. An automated system may detect such bounces and make a ruling as to whether to award a point. The system may poll a given sensor, to determine whether a bounce has occurred since the previous polling event. If the system detects two bounces it awards a point. Where b indicates that a bounce has just occurred, s indicates that system has just checked its sensor, and e indicates the end of the round.

The player deserves a point precisely if $\theta = F(b \wedge F(b \wedge Fe))$ holds while the system awards a point precisely where the formula $\theta_s = F(b \wedge F(s \wedge F(b \wedge F(s \wedge Fe))))$ holds. The result is correct if $\theta \leftrightarrow \theta_s$ holds. We can verify the result is correct for a run of the system against the environment described by the ME:

$$\{s\} + \{b\} + \{s\} + \{s\} + \{b\} + \{s\} + \overrightarrow{\{b\}} + \{e\}$$

5 Model Checking

We will begin this section by providing the preliminary definitions needed to define the model checking procedure. It will be defined at a high level to make the intuitions and correctness clear. The implementation is also available [15]. First, we will define our model checking problem.

Definition 4. *We define the* model checking problem *as follows: given an ME $\mathcal{I}$ and formula ϕ, determine whether there exists a structure $\mathbf{T} = (T, <, h)$ corresponding to $\mathcal{I}$ and point $x \in T$ such that $\mathbf{T}, x \vDash \phi$.*

It is common to define the model checking problem as determining whether a formula ϕ is true at a given point. To solve this variation of the problem, with our model checking procedure, we can add a special atom p_0 and model check the formula $p_0 \rightarrow \phi$. We will not explicitly consider MEs that contain the empty sub-ME λ as these can be trivially reduced to an equivalent ME without λ.

[3] Note that this ME cannot correspond to an interval of the reals. For the reals we could replace $\emptyset$ with $\langle \emptyset \rangle$ and use the Dedekind closure found in [4].

The model checking procedure we will define follows the traditional approach of iteratively replacing formulas with atoms. The result of adding a formula α as an atom to an ME $\mathcal{I}$ is "$\text{add\_atom}_{\alpha}\left(\mathcal{I}\right)$" which will be defined later in this section. We will only consider formulas where all subformulas have been replaced with atoms, that is formulas of the form: $p \wedge q$, $\neg p$, $U(p,q)$ and $S(p,q)$. We define $\text{add\_atom}_{p\wedge q}\left(\mathcal{I}\right)$ as the ME that results when each letter within $\mathcal{I}$ that contains both p and q has $p \wedge q$ added and likewise define $\text{add\_atom}_{\neg p}\left(\mathcal{I}\right)$ as the ME where each letter that does not contain p has $\neg p$ added or formally:

Definition 5. *Let p, q be atoms, ϕ be a formula of the form $p \wedge q$ or $\neg p$ and a be a letter. If $\phi = p \wedge q$ and $p,\, q \in a$ or $\phi = \neg p$ and $p, \notin a$ we let $\text{add\_atom}_{\phi}\left(a\right)$ be $a \cup \{\phi\}$; otherwise we let $\text{add\_atom}_{\phi}\left(a\right)$ be a. We define $\text{add\_atom}_{\phi}\left(\mathcal{I}\right)$ recursively as follows:*

1. $\text{add\_atom}_{\phi}\left(\mathcal{I} + \mathcal{J}\right) = \text{add\_atom}_{\phi}\left(\mathcal{I}\right) + \text{add\_atom}_{\phi}\left(\mathcal{J}\right)$
2. $\text{add\_atom}_{\phi}\left(\overleftarrow{\mathcal{I}}\right) = \overleftarrow{\text{add\_atom}_{\phi}\left(\mathcal{I}\right)}$
3. $\text{add\_atom}_{\phi}\left(\overrightarrow{\mathcal{I}}\right) = \overrightarrow{\text{add\_atom}_{\phi}\left(\mathcal{I}\right)}$
4. $\text{add\_atom}_{\phi}\left(\langle \mathcal{I}_0, \ldots, \mathcal{I}_n \rangle\right) = \langle \text{add\_atom}_{\phi}\left(\mathcal{I}_0\right), \ldots, \text{add\_atom}_{\phi}\left(\mathcal{I}_n\right) \rangle$

The case of $U(p,q)/S(p,q)$ is less simple. We will now take a fixed formula of the form $U(p,q)$, and show how to add it as an atom to an ME.

For each Boolean $\dashv \in \{\top, \bot\}$ and ME $\mathcal{K}$ we will define a Boolean $\text{pre}\left(\mathcal{K}, \dashv\right)$. Say that $\mathcal{K}$ corresponds to an interval $\mathbf{T}_{\mathcal{K}}$ of $\mathbf{T}$. Informally, $\dashv$ represents whether $U(p,q)$ would be true at a point added immediately after $\mathbf{T}_{\mathcal{K}}$, and $\text{pre}\left(\mathcal{K}, \dashv\right)$ represents whether $U(p,q)$ would be true at a point added immediately prior to $\mathbf{T}_{\mathcal{K}}$. In the proof of correctness this will be formalised in terms of presatisfaction.

Definition 6. *We define a function "pre" from Booleans and MEs to Booleans such that: for any Boolean $\dashv$ and pair of MEs $\mathcal{I}, \mathcal{J}$*

1. $\text{pre}\left(\mathcal{I} + \mathcal{J}, \dashv\right) = \text{pre}\left(\mathcal{I}, \text{pre}\left(\mathcal{J}, \dashv\right)\right)$
2. $\text{pre}\left(\overrightarrow{\mathcal{I}}, \dashv\right) = \text{pre}\left(\mathcal{I}, \dashv\right)$
3. $\text{pre}\left(a, \dashv\right) = p \in a \vee \left(\dashv \wedge q \in a\right)$
4. $\text{pre}\left(\mathcal{J}, \dashv\right) = \left(\dashv \vee \exists l \in L\left(\mathcal{J}\right) \text{ s.t. } p \in l\right) \wedge \forall l \in L\left(\mathcal{J}\right), q \in l$; where $\mathcal{J}$ is of the form $\overleftarrow{\mathcal{I}}$ or $\langle \ldots \rangle$ and $L\left(\mathcal{J}\right)$ is the set of letters within $\mathcal{J}$.

An important property of pre is that $\text{pre}\left(\mathcal{I}, \dashv\right) \equiv \text{pre}\left(\mathcal{I}, \text{pre}\left(\mathcal{I}, \dashv\right)\right)$. This property is quite useful for dealing with leads as we only have to treat the last occurrence of $\mathcal{I}$ specially. For example, if we add $U\left(p,p\right)$ as an atom to $\overleftarrow{\{p\}}$, $U\left(p,p\right)$ would be true everywhere except the last point so which can be represented using the finite ME $\overleftarrow{\{p, U\left(p,p\right)\}} + \{p\}$. When considering the use of pre in part 3 of the definition below it is also helpful to look back to Fig. 2 and note that the fragment of $\mathcal{I} = \langle I_0, \ldots, I_n \rangle$ that follows an I_j in $\mathcal{I}$ itself resembles the whole of $\mathcal{I}$.

Definition 7. *We define add_atom*$_{U(p,q)}(\mathcal{I})$ *as* $t(\mathcal{I},\bot)$*: where* t *is a function that takes an ME and a Boolean as input, and outputs an ME as follows: for any Boolean* $\dashv$*, pair of MEs* $\mathcal{I},\mathcal{J}$ *and sequence of MEs* $\mathcal{I}_0,\ldots,\mathcal{I}_n$

1. $t(\mathcal{I}+\mathcal{J},\dashv)=t(\mathcal{I},pre(\mathcal{J},\dashv))+t(\mathcal{J},\dashv)$

1. $t\left(\overleftarrow{\mathcal{I}},\dashv\right)=\overleftarrow{t(\mathcal{I},\mathrm{pre}(\mathcal{I},\dashv))}+t(\mathcal{I},\dashv)$
2. $t\left(\overrightarrow{\mathcal{I}},\dashv\right)=\overrightarrow{t(\mathcal{I},\mathrm{pre}(\mathcal{I},\dashv))}$
3. $t(\mathcal{K},\dashv)=\langle t(\mathcal{I}_0,\dashv'),\ldots,t(\mathcal{I}_n,\dashv')\rangle$ where $\mathcal{K}=\langle\mathcal{I}_0,\ldots,\mathcal{I}_n\rangle$ and $\dashv'=\mathrm{pre}(\mathcal{K},\dashv)$
4. $t(a,\dashv)=\begin{cases}a & \text{if } \dashv=\bot\\ a\cup\{U(p,q)\} & \text{if } \dashv=\top\end{cases}$

The Since operator is the mirror image of the Until operator. It is easy to reverse add_atom$_{U(p,q)}(\mathcal{I})$ to provide a definition of add_atom$_{S(p,q)}(\mathcal{I})$, see [7].

We define the length of an ME to be the total number of occurrences of letters and operators in the ME, so e.g. $\langle\{\},\{p,q,r,z\}\rangle$ $\{\}+\{\}$ and $\overleftrightarrow{\{p\}}$ all have a length of 3. Note that adding an until as an atom only increases the length of MEs that contain $\leftarrow$. Despite the semantic complexity of a shuffle it does not increase the computational complexity of model checking.

Having provided all these preliminary definitions it is now trivial to provide a formal definition of our model checking procedure.

Definition 8. *The* model checking procedure *takes as input an ME* $\mathcal{I}$ *and formula* ϕ*. We enumerate the subformulas* $\phi_1,\ldots,\phi_n$ *of* ϕ *from shortest to longest (so* $\phi_n=\phi$*), let* $\mathcal{I}_0=\mathcal{I}$*, and* $\mathcal{I}_i=$ *add_atom*$_{\phi_i}(\mathcal{I}_{i-1})$ *for each* $i\in\{0,\ldots,n\}$*. Finally, we return "true" if there is a letter* a *in* $\mathcal{I}_n$ *such that* $\phi\in a$*, and "false" otherwise.*

6 Proof of Correctness

In this section we prove the correctness of the model checking procedure. First we will define the concept of adding a formula as atom to structure $\mathbf{T}$.

Definition 9. *Say* $\mathbf{T}=(T,<,h)$ *then* $\mathbf{T}$ *with* α *added as an atom is* $\mathbf{T}^\alpha=(T,<,h')$ *where* $h'(p)=h(p)$ *for all* $p\in\mathbf{L}$ *and for each point* $x\in T$ *we have* $x\in h'(\alpha)$ *iff* $\mathbf{T},x\vDash\alpha$*.*

We now formalise and prove the correctness of add_atom in the next lemma.

Lemma 2. *Say* $\mathcal{I}$ *corresponds to* $\mathbf{T}$ *using* $\mathbf{L}$ *as our set of atoms and* α *is a formula of the form* $p\wedge q$*,* $\neg p$*,* $U(p,q)$ *or* $S(p,q)$*. Then add_atom*$_\alpha(\mathcal{I})$ *corresponds to* $\mathbf{T}^\alpha=(T,<,h')$ *using* $\mathbf{L}\cup\alpha$ *as our set of atoms.*

The case where α is of the form $p\wedge q$ or $\neg p$ is trivial. For a proof of correctness for $U(p,q)$ see Sect. 6.1. Since the S operator is essentially the U operator with the direction of time reversed we may prove the correctness of $S(p,q)$ in the same way as $U(p,q)$.

Theorem 2. *Given an ME $\mathcal{I}$ and formula ϕ, the model-checking procedure halts, and it returns "true" iff there exists a structure $\mathbf{T} = (T, <, h)$ and point $x \in T$ such that $\mathcal{I}$ corresponds to $\mathbf{T}$ and $\mathbf{T}, x \vDash \phi$.*

Proof. We see that the definitions of pre and t are not circular and so the model-checking procedure halts.

Let $\mathbf{T}$ be a structure corresponding to $\mathcal{I}$. Take the enumeration $\phi_1, \ldots, \phi_n$ of the subformulas of ϕ used by the model checker. Let $\mathbf{L}_0 = \mathbf{L}$, $\mathbf{T}_0 = \mathbf{T}$ and $\mathcal{I}_0 = \mathcal{I}$. For each $i \in \{0, \ldots, n\}$ let $\mathbf{L}_i = \mathbf{L}_{i-1} \cup \{\phi_i\}$, and let $\mathbf{T}_i = (T_i, <_i, h_i) = \mathbf{T}_{i-1}^{\phi_i}$, and as in the model checking procedure we let $\mathcal{I}_i = \text{add\_atom}_{\phi_i}(\mathcal{I}_{i-1})$. For each i let P_i be the proposition that $\mathcal{I}_i$ corresponds to $\mathbf{T}_i$ using $\mathbf{L}_i$ as our set of atoms. We see that P_0 is trivially true. From Lemma 2 we see that if P_{i-1} is true then P_i is true. Thus by induction P_n is true. We see that from Proposition 1 that there exists a letter within $\mathcal{I}_n$ that contains ϕ iff there exists $y \in h_n(\phi)$. Finally from the definition of $\mathbf{T}_n$ we see that $T, x \vDash \phi$ iff $y \in h_n(\phi)$. $\square$

6.1 Correctness of Adding $U(p, q)$ as an Atom

We will now prove the correctness of $\text{add\_atom}_{U(p,q)}(\mathcal{I})$. We fix a structure $\mathbf{T} = (T, <, h)$ over the set of atoms $\mathbf{L}$ and a structure $\mathbf{S} = \mathbf{T}^{U(p,q)}$. Whenever we discuss correspondence of an ME to $\mathbf{S}$ or a restriction of $\mathbf{S}$, we will interpret the correspondence using $\mathbf{L} \cup \{U(p, q)\}$ as our set of atoms. We will first prove that the function "pre" accurately predicts whether an interval presatisfies a formula.

Lemma 3. *Let $\mathcal{K}$ be an ME that corresponds to some interval $\mathbf{T}_W = (W, <_{\mathcal{K}}, h_{\mathcal{K}})$ of $\mathbf{T} = (T, <, h)$. Let $R = \{y : \forall x \in W,\ x < y\}$, that is the set of points after the interval $\mathbf{T}_W$, and let $\mathbf{T}_R$ be the restriction of $\mathbf{T}$ to the points in T_R. We let $\dashv$ be "$\mathbf{T}_R$ presatisfies $U(p, q)$". Then $\mathbf{T}_{W \cup R}$ presatisfies $U(p, q)$ iff pre $(\mathcal{I}, \dashv)$ is true.*

Proof. Say that $\mathcal{K}$ is a minimal ME that provides a counterexample to this lemma. Say $\mathcal{K}$ is of the form:

a It trivially follows from definition of pre and presatisfaction that a does not provide a counterexample.

$\mathcal{I} + \mathcal{J}$ Following the definition of correspondence, we divide W into U and V such that $\mathcal{I}$ and $\mathcal{J}$ correspond to $\mathbf{T}_U$ and $\mathbf{T}_V$ respectively. Since $\mathcal{K}$ is minimal, $\mathcal{J}$ is not a counterexample and so we see that $\mathbf{T}_{V \cup R}$ presatisfies $U(p, q)$ iff pre $(\mathcal{J}, \dashv)$. Likewise $\mathcal{I}$ is not a counterexample, so $\mathbf{T}_{U \cup (V \cup R)}$ presatisfies $U(p, q)$ iff pre $(\mathcal{I}, \text{pre}(\mathcal{J}, \dashv)) = \text{pre}(\mathcal{K}, \dashv)$. Thus we do not have a counterexample.

$\overrightarrow{\mathcal{I}}$ Following the definition of correspondence we divide W into $U_0, U_1, \ldots$. Consider the following four cases:

– pre $(\mathcal{I}, \bot) = \bot$ and pre $(\mathcal{I}, \top) = \bot$: then since $\mathcal{I}$ is not a counterexample, we see that $\mathbf{T}_{W \cup R}$ does not presatisfy $U(p, q)$ (regardless of whether $\mathbf{T}_{(W - U_0) \cup R}$ presatisfies $U(p, q)$). Since pre $\left(\overrightarrow{\mathcal{I}}, \dashv\right) = \text{pre}(\mathcal{I}, \dashv) = \bot$, we do not have a counterexample.

- $\operatorname{pre}(\mathcal{I}, \bot) = \top$ and $\operatorname{pre}(\mathcal{I}, \top) = \top$: We see that $\mathbf{T}_{W \cup R}$ presatisfies $U(p, q)$ regardless, so again this is not a counterexample.
- $\operatorname{pre}(\mathcal{I}, \bot) = \bot$ and $\operatorname{pre}(\mathcal{I}, \top) = \top$: From the definition of pre we see that every letter of $\mathcal{I}$ (and thus $\mathcal{K}$) has q but does not have p. Since $\mathcal{K}$ corresponds to $\mathbf{T}_W$, $h_W(q) = W$ and $h_W(p) = \emptyset$. Hence $\mathbf{T}_{W \cup R}$ presatisfies $U(p, q)$ iff $\mathbf{T}_R$ does.
- finally note that there is no ME $\mathcal{I}$ where $\operatorname{pre}(\mathcal{I}, \bot) = \top$ and $\operatorname{pre}(\mathcal{I}, \top) = \bot$.

$\overleftarrow{\mathcal{I}}$ or $\langle \ldots \rangle$. Consider whether $\mathbf{T}_{W \cup R}$ presatisfies $U(p, q)$.

- Yes: then there exists $y \in h_{W \cup R}(p)$ such that for each point $x < y$ in $T_{W \cup R}$ we have $x \in h(q)$.
  - Say $y \in R$, then every point in W is less than y and so it must be the case that $h_W(q) = W$. Since $h_W(q) = W$ we see that every letter $\mathcal{K}$ has q and so $\operatorname{pre}(\mathcal{K}, \top) = \top$. We see that $\mathbf{T}_R$ presatisfies $U(p, q)$ and so $\dashv = \top$.
  - Say $y \notin R$. Then $y \in W$. We see that if there were a letter within $\mathcal{K}$ that did not contain q, then there would be a point $x < y$ in W such that $x \notin h(q)$, contradicting the definition of presatisfaction. Hence every letter of $\mathcal{K}$ contains q. Since $y \in W$ we see that a letter within $\mathcal{K}$ contains p. Thus $\operatorname{pre}(\mathcal{K}, \dashv) = \top$ regardless of $\dashv$.
- No: Since $\mathcal{K}$ provides a counterexample $\operatorname{pre}(\mathcal{K}, \dashv) = \top$. Thus every letter within $\mathcal{K}$ has q.
  - Say a letter within $\mathcal{K}$ contains p. Then we see that there is a point y such that $y \in h_W(p)$ and furthermore that every point $x < y$ in W is in $h_W(q)$. Thus $\mathbf{T}_W$ presatisfies $U(p, q)$, giving us a contradiction.
  - Say that no letter within $\mathcal{K}$ has p. Then from the definition of pre we see that $\dashv = \top$ and so $\mathbf{T}_R$ presatisfies pUq. Since every point in W satisfies p we see that $\mathbf{T}_{W \cup R}$ also presatisfies $U(p, q)$, giving us a contradiction.

By the principle of contradiction we conclude that no counterexample exists. □

Lemma 4. *Let $\mathcal{K}$ be an ME corresponding to some interval $\mathbf{T}_W = (W, <_{\mathcal{K}}, h_{\mathcal{K}})$ of $\mathbf{T} = (T, <, h)$. Let $R = \{y \in T : \forall x \in W, x < y\}$, that is the set of points after $\mathbf{T}_W$. Let $\dashv$ be "$\mathbf{T}_R$ presatisfies $U(p, q)$". Then $t(\mathcal{K}, \dashv)$ corresponds to $\mathbf{S}_W$.*

Like Lemma 3, the proof of Lemma 4 proceeds by considering each possible form of a minimal ME that provides a contradiction. [7]

We now get correctness of $\text{add\_atom}_{U(p,q)}(\mathcal{I})$ from the following corollary.

Corollary 1. *If $\mathcal{I}$ corresponds to $\mathbf{T}$ then $\text{add\_atom}_{U(p,q)}(\mathcal{I})$ corresponds to $\mathbf{S}$.*

7 Complexity

In this section we will give a brief summary of our complexity results and benchmarks. Many more details and benchmarks can be found in [7].

It is easy to implement the algorithm such that it is polynomial with respect to the largest ME produced by the algorithm and the size of the input formula [15]. We see that after u Until operators have been processed $\overleftarrow{\mathcal{I}}$ gives us something of the form $\overleftarrow{\mathcal{I}_0} + \mathcal{I}_1 + \cdots + \mathcal{I}_n$, potentially increasing the size of the ME at most $n+1$ times. As leads can be nested this means the final ME may be as much as $(u+1)^L$ times larger than the size of input ME, where L is the number of leads. The case of Since operators is similar and from this we can get an $|\phi|^{|\mathcal{I}_0|}$ upper bound on the growth factor. Unfortunately this is exponential in the size of the ME. Our implementation stores MEs as Directed Acyclic Graphs (DAG) with identical subMEs are deduplicated. For convenience we will write $t\,(\mathcal{I}, \dashv)$ as $\mathcal{I}^{\dashv}$. Each unique subME produced by adding an Until formula is of the form $\mathcal{I}^{\top}$, $\mathcal{I}^{\perp}$,$\overleftarrow{\mathcal{I}^{\top}}$, or $\overleftarrow{\mathcal{I}^{\perp}}$ where $\mathcal{I}$ is a subME of the original ME. Thus adding an atom increases the size of the DAG by at most four times. This gives us a $4^{|\phi|}$ upper bound on the growth factor. This upper bound can be refined to $|\phi|\,2^{|\phi|}$ [7]. These upperbounds on the growth allow an implementation that is linear in time with respect to the input ME, for formulas of fixed size [7,15].

In a follow on publication [6] we will show that the model checking problem is PSPACE-Complete. We will show that our model-checking problem is PSPACE-hard by a reduction from solving satisfiability of Quantified Boolean Formulas, and present a Model Checking algorithm that does not store the interim MEs allowing it to run using only a polynomial amount of space (but inefficient in time).

Theorem 3. *Our Model-Checking problem is PSPACE-complete. [6]*

Having considered theoretical complexity, we will use benchmarks to quantify actual performance. We choose random formulas of size n using the following simple recursive procedure: if n is 1, choose randomly from six atoms; if n is 2 choose a negation of the randomly chosen atom, otherwise with equal chance choose to start either with "$\neg$" or a binary operator. If we choose to start with a binary operator we then randomly pick one from the set $\{U, S, \wedge, \vee\}$. If unary, let the remainder of the formula be a random formula of size $n-1$; if binary choose a random split of $n-1$ into n_0 and n_1 such that $n_0 + n_1 + 1 = n$ and let the left child be a random formula of size n_0 and the right child be a random formula of size n_1. Randomly generated MEs are similar but where $n = 1$ we randomly choose a subset of the 6 atoms, and choose one of $\{\leftarrow, \rightarrow\}$ in place of $\neg$. We see that there are 2^6 possible subsets of atoms, which is enough to encode non-trivial problems such a prenex QBF satisfiability problem with 2^6 atoms.

In Table 1, the i^{th} row represents the case where we model check a random ME of size 10^i against a random formula of size 100. Recall that the formula column is the number of unique sub-formula, which will be slightly less than the size of the input formula. Likewise the "InputME" column may be smaller than 10^i. We see that the performance of the algorithm is plausibly linear in the size of the ME for a fixed formula. We are able to model check non-trivial MEs, for example checking the ME of size 10^7 takes only half an hour. Note that the deduplication done when first parsing the input is at least $\mathcal{O}\,(n \ln n)$. This could

Table 1. Fixed size formula: $|\phi| = 100$, $|I_i| = 10^i$

i	Formula	InputME	FinalME	MB	CPU
1	75	10	12	1	0.00
2	74	96	170	3	0.02
3	73	722	2233	6	0.47
4	75	5694	15444	14	2.02
5	73	51551	120463	109	19.07
6	72	453679	1174276	800	231.52
7	71	4027606	8313117	6297	1514.78

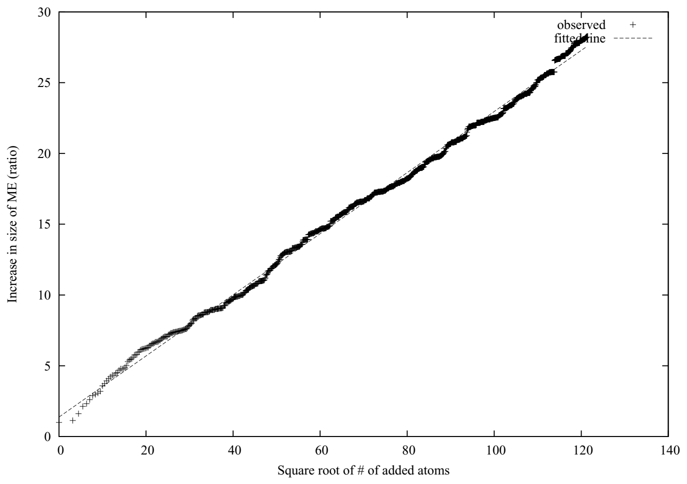


Fig. 3. Plot of growth of $|\mathcal{I}_n|/|\mathcal{I}_0|$ vs. $\sqrt{n}$ for $2^{15} \times 2^{15}$ problem

be disabled, to get "true" linear time, but this extra $\ln n$ only occurs before the model checker starts and provides significantly better performance in practice.

Let $\mathcal{I}_n$ be our initial ME $\mathcal{I}_0$ after n atoms have been added, and let $r_n = |\mathcal{I}_n|/|\mathcal{I}_0|$ be the ratio between the input ME and the final ME. Since exponential growth in r_n is the only way that we can get exponential growth in the time or space used by this algorithm, an obvious question is how quickly it grows in practice. We plot how quickly r_n grows in the 2^{15} by 2^{15} problem, against $\sqrt{n}$ (see Fig. 3). We see that the graph closely fits a straight line. This suggests that the growth is of order $\sqrt{n}$. We will now attempt to quantify how accurate $\sqrt{n}$ is as a predictor of r_n for randomly generated problems in general.

We note than an interim (or final) ME $\mathcal{I}_n$ has $n+6$ atoms, since each iteration adds an additional atom, an each randomly generated ME begins with 6 atoms. Let $a_n = n + 6$ be the number of atoms used in $\mathcal{I}_n$. We have found that in all cases $r_n = \lambda_n \sqrt{a_n}$ where $0.113 < \lambda_n < 0.466$. This result held not just for every n, but also for all randomly generated $\mathcal{I}$ and ϕ considered. These included $\mathcal{I}$ of size up to 10^7 and ϕ of size up to 10^6 (see [7] for details). Since $\sqrt{a_n}$ can be used to predict r_n to within an order of magnitude for all these cases it is plausible that for randomly generated model checking problems, r_n is of order $\sqrt{a_n}$. In any case, r_n appears to be sublinear. While these randomly generated results may or may not apply to a given real world application, it is clear that there is a large class of problems for which r_n will not grow exponentially.

8 Conclusions

In this paper we have investigated the idea of model checking in general linear temporal logic. We have managed to formalise the task using the logic US/L and the model expression language of [4]. We have provided an algorithm, proved it correct and analysed its performance.

We have an $|\mathcal{I}|\,|\phi|\,2^{\phi}$ upper bound on the number of unique sub-MEs in the resulting ME, and have a similar bound on worst case performance. As this model checking problem can be solved in time linear with respect to the length of input model it is expected to be tractable in practice, and this expectation was supported by benchmarks on random formulas. For example, we have checked a randomly generated ME of size 10^7 against a randomly generated formula of size 100, using 6 atoms in only half an hour. Interestingly, despite the semantic complexity of the shuffle operator it is easy to model check, both in theory and in practice. Also of interest, we have found that our model checking problem is polynomial if we constrain either the size of the model *or* the formula. The model checking problem is also known to be PSPACE-complete [7,6], so the complexity is quite similar to that of LTL. We also note the similarity of the complexity of model-checking to synthesis and satisfiability. Synthesis [4] and Satisfiability [17] checking temporal logic with Until and Since are also PSPACE-complete over the reals, and over general linear time as well [20].

As mentioned, we hope in the future to be able to make easily specialised versions of the model checker for logics of the language with Until and Since over useful specific flows of time. In particular we will be examining RTL.

References

1. Alur, R., Courcoubetis, C., Dill, D.L.: Model-checking for real-time systems. In: LICS 1990, pp. 414–425. IEEE Computer Society (1990)
2. Alur, R., Dill, D.L.: Automata for modeling real-time systems. In: Paterson, M. (ed.) ICALP 1990. LNCS, vol. 443, pp. 322–335. Springer, Heidelberg (1990)
3. Burgess, J.P., Gurevich, Y.: The decision problem for linear temporal logic. Notre Dame J. Formal Logic 26(2), 115–128 (1985)
4. French, T., McCabe-Dansted, J.C., Reynolds, M.: Synthesis for temporal logic over the reals. In: Bolander, T., Braüner, T., Ghilardi, S., Moss, L.S. (eds.) Advances in Modal Logic 2012, pp. 217–238. College Publications (2012)
5. French, T., McCabe-Dansted, J., Reynolds, M.: Indiscrete models: Model building and model checking over linear time. In: Lodaya, K. (ed.) ICLA 2013. LNCS, vol. 7750, pp. 50–68. Springer, Heidelberg (2013)

6. French, T., M^{c}Cabe-Dansted, J.C., Reynolds, M.: Complexity of model checking general linear time. In: TIME 2013 (accepted, to appear, 2013)
7. French, T., McCabe-Dansted, J.C., Reynolds, M.: Model checking for compositional models of general linear time: Long version. Tech. rep., CSSE, UWA (Dec 2012), `http://www.csse.uwa.edu.au/~john/papers/ModelCheckZeno_tech.pdf`
8. Gabbay, D., Hodkinson, I., Reynolds, M.: Temporal Logic: Mathematical Foundations and Computational Aspects, vol. 1. Oxford University Press (1994)
9. Gabbay, D.M., Hodkinson, I.M., Reynolds, M.A.: Temporal expressive completeness in the presence of gaps. In: Oikkonen, J., Väänänen, J. (eds.) Logic Colloquium 1990, Proceedings ASL European Meeting 1990, Helsinki. Lecture Notes in Logic, vol. 2, pp. 89–121. Springer (1993)
10. Gabbay, D.M., Pnueli, A., Shelah, S., Stavi, J.: On the temporal analysis of fairness. In: 7th ACM Symp. on Princ. of Prog. Languages, Las Vegas, pp. 163–173 (1980)
11. Hugett, N.: Zeno's paradoxes: 3.2 achilles and the tortoise. In: Zalta, E. (ed.) Stanford Encyclopedia of Philosophy. Chapman and Hall, Boca Raton (2010)
12. Kamp, H.: Tense logic and the theory of linear order. Ph.D., UCLA (1968)
13. Läuchli, H., Leonard, J.: On the elementary theory of linear order. Fundamenta Mathematicae 59, 109–116 (1966)
14. Mosterman, P.: 15.6 pathological behaviour classes, hybrid dynamic systems: Modeling and execution. In: Fishwick, P. (ed.) Handbook of Dynamic System Modelling, ch. 15, pp. 15–22 to 15–23. Chapman and Hall, Boca Raton (2007)
15. McCabe-Dansted, J.C.: Model checker for general linear time (online applet and data) (2012), `http://www.csse.uwa.edu.au/~mark/research/Online/mechecker.html`
16. Pnueli, A.: The temporal logic of programs. In: Proceedings of the 18th Symposium on Foundations of Computer Science, Providence, RI, pp. 46–57 (1977)
17. Reynolds, M.: The complexity of the temporal logic over the reals. Annals of Pure and Applied Logic 161(8), 1063–1096 (2010), doi:10.1016/j.apal.2010.01.002
18. Reynolds, M.: Continuous temporal models. In: Stumptner, M., Corbett, D., Brooks, M. (eds.) AI 2001. LNCS (LNAI), vol. 2256, pp. 414–425. Springer, Heidelberg (2001)
19. Reynolds, M.: Dense time reasoning via mosaics. In: TIME 2009: Proceedings of the 2009 16th International Symposium on Temporal Representation and Reasoning, pp. 3–10. IEEE Computer Society, Washington, DC (2009)
20. Reynolds, M.: The complexity of temporal logics over linear time. Journal of Studies in Logic 3, 19–50 (2010)
21. Reynolds, M.: A tableau for until and since over linear time. In: Combi, C., Leucker, M., Wolter, F. (eds.) TIME, pp. 41–48. IEEE (2011)

Semantically Guided Evolution of $\mathcal{SHI}$ ABoxes

Ulrich Furbach and Claudia Schon

University of Koblenz-Landau, Germany
{uli,schon}@uni-koblenz.de

Abstract This paper presents a method for the evolution of $\mathcal{SHI}$ ABoxes which is based on a compilation technique of the knowledge base. For this the ABox is regarded as an interpretation of the TBox which is close to a model. It is shown, that the ABox can be used for a semantically guided transformation resulting in an equisatisfiable knowledge base. We use the result of this transformation to efficiently delete assertions from the ABox. Furthermore, insertion of assertions as well as repair of inconsistent ABoxes is addressed. For the computation of the necessary actions for deletion, insertion and repair, the E-KRHyper theorem prover is used.

1 Introduction

Description Logic knowledge bases consist of two parts: the TBox and the ABox. The TBox contains the terminological knowledge and describes the world using so called concepts and roles. The ABox contains knowledge about individuals, stating to which concepts they belong to and via which roles they are connected. There is a considerable amount of work introducing update algorithms and mechanisms for Description Logic knowledge bases, which is of great interest to the Semantic Web community (see [11,16] for details). It is an indisputable fact, that in practice, knowledge bases are subject to frequent changes ([10]) and that even the construction of a knowledge base can be seen as an iterative process. On the other hand this abets inconsistencies in knowledge bases. Therefore the removal of inconsistencies from knowledge bases is of great interest as well ([13]). In this paper we are interested in an evolution of the knowledge base on the instance level. For this, we consider the TBox to be fixed and consistent. We address three different operations on the instance level of the knowledge base: *deletion*, *insertion* and *repair*. Instance-level deletion means the deletion of an instance assertion from the deductive closure or the knowledge base by removing as few assertions as possible. Instance-level insertion means adding an instance assertion to the knowledge base. In both cases it is important that the resulting knowledge base is consistent. For the task of ABox repair we are given an inconsistent knowledge base with consistent terminological part. The aim is to remove assertions from the ABox such that the resulting ABox together with the TBox is consistent. In all three tasks the changes performed should be minimal. This corresponds to the goal of maintaining as much from the original ABox as possible. This view of minimal change corresponds to a *formula based* approach as opposed to a *model based* approach as investigated in [16]. In the model based approach the set of models of the knowledge base resulting form a change operation should be as close as possible to the set of models of the original knowledge base.

In [15], [7] and [5] instance level deletion, insertion and repair are addressed for DL-Lite knowledge bases. In [14] inconsistent DL-Lite ABoxes are considered. [14] establishes inconsistency-tolerant semantics in order to be able to use those inconsistent ABoxes for query answering. [19] studies the complexity of reasoning under inconsistent-tolerant semantics. Algorithms for the calculation of minimal repair of DL-Lite ABoxes suggested in [14] test the satisfiability of every single ABox assertion and every pair of ABox assertions w.r.t. the TBox. Since for DL-Lite the satisfiability test is tractable, this approach is reasonable. However the ExpTime completeness of consistency testing of $\mathcal{SHI}$ ABoxes forbids such an approach. Further the algorithms suggested in [14] cannot be used for $\mathcal{SHI}$ ABoxes, because these algorithms exploit the following nice property of DL-Lite: as shown in [5], in DL-Lite the unsatisfiability of an ABox w.r.t. a TBox is either caused by a single assertion or a pair of assertions. However in $\mathcal{SHI}$ an arbitrary number of assertions can cause unsatisfiability w.r.t. a TBox.

Our approach is motivated by the observation that a consistent ABox can be seen as a (partial) model of the TBox, which can be used to guide the reasoning process, as proposed in [6]. In [3] this approach was used for model-based diagnosis, where an initial interpretation, which is very close to a model, was used to compute the deviations of a minimal model to this interpretation. In [1] the same approach was applied to view deletion in databases. In our case it is reasonable to assume, that the ABox is very close to a model of the TBox. We use this assumption to semantically guide the construction of instance-based deletion, insertion and repair of ABoxes. As in [3], we gradually revise the assumption of the given ABox being a model for the TBox. This leads to a natural construction of *minimal* instance deletions/insertions and repairs of ABoxes.

The advantage of this approach is that there is no need to define new algorithms for updates and repair, which have to be proven correct. Instead we will use a static compilation of the knowledge base according to the update or repair requirement. We prove that this transformation preserves the necessary semantics. A theorem prover can be used to compute the necessary update and repair actions. A hypertableau-based theorem prover like E-KRHyper is very well suited for this task, because the transformation enables it to calculate only the deviation of the ABox. Since E-KRHyper has recently been extended to deal with knowledge bases given in $\mathcal{SHIQ}$ [4], we chose to use this theorem prover.

Our approach is related to axiom pinpointing. For a given consequence, axiom pinpointing is the task to find the minimal subsets of the knowledge base under consideration, having this consequence. See [2] for details. In [12] laconic and precise justifications are introduced. Given an ontology and an entailment, a justification is a minimal subset of that ontology such that the entailment still holds in the subset. Roughly spoken, laconic justifications are not allowed to contain superfluous parts. In contrast to axiom pinpointing and justifications, we calculate subsets of the ABox and not of the whole knowledge base. In [20] incoherent TBoxes, i.e. TBoxes containing an unsatisfiable concept, are investigated. [11] considers so called syntactic ABbox updates. Similar to our approach, assertions are added to or removed from the ABox. In contrast to our approach, it is neither guaranteed that the removed assertion is not contained in the deductive closure nor that the result of adding the assertion is consistent.

In Section 2 we give both syntax and semantics of the Description Logic $\mathcal{SHI}$. In addition to that, we introduce the notion of DL-clauses as used in [17]. In Section 3 we give definitions for instance-level deletion, insertion and repair. Section 4 introduces the so called $\mathcal{K}^*$-transformation which in Section 5 is used to calculate the instance-level deletion, insertion and repair. The $\mathcal{K}^*$-transformation is implemented and in Section 6 we present first experimental results. Proofs of all theorems, propositions and lemmas can be found in [9].

2 $\mathcal{SHI}$ and DL-Clauses

First, we introduce the Description Logic $\mathcal{SHI}$. Given a set of *atomic roles* N_R, the set of *roles* is defined as $N_R \cup \{R^- \mid R \in N_R\}$, where R^- denotes the *inverse role* corresponding to the atomic role R. Let further Inv be a function on the set of roles that computes the inverse of a role, with $Inv(R) = R^-$ and $Inv(R^-) = R$. A *role inclusion axiom* is an expression of the form $R \sqsubseteq S$, where R and S are atomic or inverse roles. A *transitivity axiom* is of the form $Trans(S)$ for S an atomic or inverse role. An RBox $\mathcal{R}$ is a finite set of role inclusion axioms and transitivity axioms. $\sqsubseteq^*$ denotes the reflexive, transitive closure of $\sqsubseteq$ over $\{R \sqsubseteq S, Inv(R) \sqsubseteq Inv(S) \mid R \sqsubseteq S \in \mathcal{R}\}$. A role R is *transitive* in $\mathcal{R}$ if there exists a role S such that $S \sqsubseteq^* R$, $R \sqsubseteq^* S$, and either $Trans(S) \in \mathcal{R}$ or $Trans(Inv(S)) \in \mathcal{R}$. If no transitive role S with $S \sqsubseteq^* R$ exists, R is called *simple*.

Let N_C be the set of *atomic concepts*. The set of *concepts* is then defined as the smallest set containing $\top$, $\bot$, A, $\neg C$, $C \sqcap D$, $C \sqcup D$, $\exists R.C$ and $\forall R.C$ for $A \in N_C$, C and D concepts and R a role.

A *general concept inclusion* (GCI) is of the form $C \sqsubseteq D$, and a TBox $\mathcal{T}$ is a finite set of GCIs.

Given a set of individuals N_I, an ABox $\mathcal{A}$ is a finite set of assertions of the form $A(a)$ and $R(a, b)$, with A an atomic concept, R an atomic role and a, b individuals from N_I. Note that in our setting, the ABox is only allowed to contain assertions about the belonging of individuals to atomic concepts and roles.

A knowledge base $\mathcal{K}$ is a triple $(\mathcal{R}, \mathcal{T}, \mathcal{A})$ with signature $\Sigma = (N_C, N_R, N_I)$. The tuple $\mathcal{I} = (\cdot^\mathcal{I}, \Delta^\mathcal{I})$ is an *interpretation* for $\mathcal{K}$ iff $\Delta^\mathcal{I}$ is a nonempty set and $\cdot^\mathcal{I}$ assigns an element $a^\mathcal{I} \in \Delta^\mathcal{I}$ to each individual a, a set $A^\mathcal{I} \subseteq \Delta^\mathcal{I}$ to each atomic concept A, and a relation $R^\mathcal{I} \subseteq \Delta^\mathcal{I} \times \Delta^\mathcal{I}$ to each atomic role R. $\cdot^\mathcal{I}$ then assigns values to more complex concepts and roles as described in Table 1. $\mathcal{I}$ is a *model* of $\mathcal{K}$ ($\mathcal{I} \models \mathcal{K}$) if it satisfies all axioms and assertions in $\mathcal{R}$, $\mathcal{T}$ and $\mathcal{A}$ as shown in Table 1. A TBox $\mathcal{T}$ is called consistent, if there is an interpretation

Table 1. Model-theoretic semantics of $\mathcal{SHI}$. R^+ is the transitive closure of R.

Concepts and Roles	
$\top^\mathcal{I} = \Delta^\mathcal{I}$	$(R^-)^\mathcal{I} = \{(y,x) \mid (x,y) \in R^\mathcal{I}\}$
$\bot^\mathcal{I} = \emptyset$	$(\forall R.C)^\mathcal{I} = \{x \mid \forall y : (x,y) \in R^\mathcal{I} \Rightarrow y \in C^\mathcal{I}$
$(\neg C)^\mathcal{I} = \Delta^\mathcal{I} \backslash C^\mathcal{I}$	$(\exists R.C)^\mathcal{I} = \{x \mid \exists y : (x,y) \in R^\mathcal{I} \wedge y \in C^\mathcal{I}\}$
$(C \sqcup D)^\mathcal{I} = C^\mathcal{I} \cup D^\mathcal{I}$	
$(C \sqcap D)^\mathcal{I} = C^\mathcal{I} \cap D^\mathcal{I}$	
TBox & RBox axioms	**ABox axioms**
$C \sqsubseteq D \Rightarrow C^\mathcal{I} \subseteq D^\mathcal{I}$	$C(a) \Rightarrow a^\mathcal{I} \in C^\mathcal{I}$
$R \sqsubseteq S \Rightarrow R^\mathcal{I} \subseteq S^\mathcal{I}$	$R(a,b) \Rightarrow (a^\mathcal{I}, b^\mathcal{I}) \in R^\mathcal{I}$
$\mathit{Trans}(R) \Rightarrow (R^\mathcal{I})^+ \subseteq R^\mathcal{I}$	

satisfying all axioms in $\mathcal{T}$. A concept C is called *satisfiable w.r.t.* $\mathcal{R}$ and $\mathcal{T}$ iff there exists a model $\mathcal{I}$ of $\mathcal{R}$ and $\mathcal{T}$ with $C^\mathcal{I} \neq \emptyset$.

In the sequel we adapt the notion of DL-clauses introduced in [17] to the Description Logic $\mathcal{SHI}$. These DL-clauses allow to use existent theorem provers which are based on the hypertableau calculus to compute models or to decide satisfiability. DL-clauses are universally quantified implications of the form $\bigvee V_j \leftarrow \bigwedge U_i$:

Definition 1. *([17]) An atom is of the form $B(s)$, $R(s,t)$, $\exists R.B(s)$ or $\exists R.\neg B(s)$ for B an atomic concept and s and t individuals or variables. An atom not containing any variables is called a ground atom. A DL-clause is of the form $V_1 \vee \ldots \vee V_n \leftarrow U_1 \wedge \ldots \wedge U_m$ with V_i atoms and U_j atoms of the form $B(s)$ or $R(s,t)$ and $m \geq 0$ and $n \geq 0$. If $n = 0$, we denote the left hand side (head) of the DL-clause by $\bot$. If $m = 0$, we denote the right hand side (body) of the DL-clause by $\top$.*

Definition 2. *(Semantics of DL-clauses; [17]) Let $V_1 \vee \ldots \vee V_n \leftarrow U_1 \wedge \ldots \wedge U_m$ be a DL-clause and N_V a set of variables, disjoint from N_I. Let further $\mathcal{I} = (\Delta^\mathcal{I}, \cdot^\mathcal{I})$ be an interpretation and $\mu : N_V \rightarrow \Delta^\mathcal{I}$ be a variable mapping. Let $a^{\mathcal{I},\mu} = a^\mathcal{I}$ for an individual a and $x^{\mathcal{I},\mu} = \mu(x)$ for a variable x. Satisfaction of an atom, a DL-clause, and set of DL-clauses N in $\mathcal{I}$ and μ is defined as follows:*

$\mathcal{I}, \mu \models C(s)$	*if* $s^{\mathcal{I},\mu} \in C^\mathcal{I}$
$\mathcal{I}, \mu \models R(s,t)$	*if* $\langle s^{\mathcal{I},\mu}, t^{\mathcal{I},\mu} \rangle \in R^\mathcal{I}$
$\mathcal{I}, \mu \models \bigvee_{j=1}^{n} V_j \leftarrow \bigwedge_{i=1}^{m} U_i$	*if* $\mathcal{I}, \mu \models V_j$ *for some* $1 \leq j \leq n$ *whenever* $\mathcal{I}, \mu \models U_i$ *for each* $1 \leq i \leq m$
$\mathcal{I} \models \bigvee_{j=1}^{n} V_j \leftarrow \bigwedge_{i=1}^{m} U_i$	*if* $\mathcal{I}, \mu \models \bigvee_{j=1}^{n} V_j \leftarrow \bigwedge_{i=1}^{m} U_i$ *for all mappings* μ
$\mathcal{I} \models N$	*if* $\mathcal{I} \models r$ *for each DL-clause* $r \in N$

We will not give the transformation into DL-clauses. The details can be found in [17]. The transformation avoids an exponential blowup by using the well-known structural transformation [18] and can be computed in polynomial time.

By $\Xi(\mathcal{T})$ ($\Xi(\mathcal{A})$) we denote the set of DL-clauses for a TBox $\mathcal{T}$ (an ABox $\mathcal{A}$). For a knowledge base $\mathcal{K} = (\mathcal{T}, \mathcal{A})$, $\Xi(\mathcal{K}) = \Xi(\mathcal{T}) \cup \Xi(\mathcal{A})$. According to [17] for every interpretation $\mathcal{I}$, $\mathcal{I} \models \mathcal{K}$ iff $\mathcal{I} \models \Xi(\mathcal{T})$ and $\mathcal{I} \models \mathcal{A}$.

Since we assume the ABox assertions to be atomic, the ABox itself corresponds to a set of DL-clauses.

Example 1. The TBox $\mathcal{T} = \{B \sqsubseteq \exists R.C, \exists R.C \sqsubseteq D, D \sqsubseteq C\}$ corresponds to the set of DL-clauses $\Xi(\mathcal{T}) = \{\exists R.C(x) \leftarrow B(x), D(x) \leftarrow R(x,y) \wedge C(y), C(x) \leftarrow D(x)\}$.

Sometimes it is convenient to regard both the body and the head of a DL-clause C as a set of atoms like $C = \mathbf{H} \leftarrow \mathbf{B}$. This allows us to write $A \in \mathbf{B}$ ($A \in \mathbf{H}$) if atom A occurs in the body (in the head) of DL-clause C. The signature of a set of DL-clauses is the set of atomic concepts and atomic roles occurring in the DL-clause. The size of a DL-clause C is defined as the numbers of atoms occurring in C and is denoted $\mathit{size}(C)$. The size of a set of DL-clauses N denoted by $\mathit{size}(N)$ is the sum of sizes of all DL-clauses in N. In the sequel we need a function extracting the concept/role from an atom:

Definition 3. *(Symbol Extraction Function) Let A be an atom. Then $\sigma(A)$ is defined as follows:*

$$\sigma(A) = \begin{cases} B & \text{if } A = B(s) \text{ for some atomic concept } B, \\ R & \text{if } A = R(s,t) \text{ for some atomic role } R, \\ \exists R.B & \text{if } A = \exists R.B(s) \text{ for some atomic role } R \text{ and} \\ & B = E \text{ or } \neg E \text{ for some atomic concept } E. \end{cases}$$

By $\sigma(N)$ for a set of atoms N we denote the union of $\sigma(A)$ for all atoms $A \in N$.

In the following it is convenient for us to regard an interpretation as the set of ground atoms assigned to true by the interpretation. A set of ground atoms and an interpretation can be seen as equivalent, since every set of ground atoms uniquely determines a Herbrand interpretation. Now we can introduce the idea of minimal models to DL-clauses.

Definition 4. *(Minimal Model for a Set of DL-Clauses) Let DL be a set of DL-clauses. An Interpretation $\mathcal{I}$ is called a minimal model for DL, iff $\mathcal{I}$ is a model for DL and further there is no model $\mathcal{I}'$ for DL such that $\mathcal{I}' \subset \mathcal{I}$.*

Next we define minimality of models w.r.t. a set of ground atoms. We will later use this notion in order to minimize the number of ABox assertions which are to be deleted.

Definition 5. *(Γ Minimal Model) Let DL be a set of DL-clauses and Γ be a set of ground atoms. An interpretation $\mathcal{I}$ is a Γ-minimal model for DL iff $\mathcal{I}$ is a model for DL and further there is no model $\mathcal{I}'$ for DL with $\mathcal{I}' \cap \Gamma \subset \mathcal{I} \cap \Gamma$.*

3 ABox Evolution

We address three different operations on the instance level of the knowledge base: deletion, insertion and repair. The first scenario we are considering is the following: given a knowledge base $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ with a consistent TBox $\mathcal{T}$, we want to remove an ABox assertion for example $A(a)$ from the ABox. In general it is not sufficient to only delete $A(a)$ from the ABox, because $A(a)$ can still be contained in the deductive closure. So the task is to determine a minimal set of ABox assertions, which have to be deleted from the ABox in order to prevent that $A(a)$ is a logical consequence of the knowledge base. This leads to the following definition.

Definition 6. *(Minimal Instance Deletion) Let $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ be a knowledge base where $\mathcal{T}$ is consistent. A ground atom D of the form $A(a)$ or $R(a, b)$ with $D \in \mathcal{A}$ is called delete request. Further $\mathcal{A}' \subseteq \mathcal{A}$ is called minimal instance deletion of D from $\mathcal{A}$ if $\mathcal{T} \cup \mathcal{A}' \not\models D$ and there is no $\mathcal{A}''$ with $\mathcal{A}' \subset \mathcal{A}'' \subseteq \mathcal{A}$ and $\mathcal{T} \cup \mathcal{A}'' \not\models D$.*

Example 2. We consider the ABox

$$\mathcal{A} = \{B(a), D(a), C(b), R(b, b), R(a, a)\}$$

together with the TBox given in Example 1. The delete request $D(a)$ has a minimal instance deletion

$$\mathcal{A}' = \{C(b), R(b, b), R(a, a)\}$$

Next we want to repair an ABox which is not consistent w.r.t. its TBox.

Definition 7. *(Minimal ABox Repair) Let $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ be a knowledge base where $\mathcal{T}$ is consistent. $\mathcal{A}' \subseteq \mathcal{A}$ is called minimal ABox repair of $\mathcal{A}$ if $\mathcal{T} \cup \mathcal{A}'$ is consistent and there is no $\mathcal{A}''$ with $\mathcal{A}' \subset \mathcal{A}'' \subseteq \mathcal{A}$ and $\mathcal{T} \cup \mathcal{A}''$ consistent.*

Note that we define the notion of a minimal ABox repair in a way, that it is also applicable to an ABox which is consistent to its TBox. In this case, the minimal ABox repair corresponds to the original ABox.

The third instance level operation we address is insertion of an assertion into an existing ABox. The problem that arises when considering insertion is, that the resulting ABox might be inconsistent w.r.t. its TBox.

Example 3. Let us consider the set of DL-clauses

$$\Xi(\mathcal{T}) = \{\bot \leftarrow C(x) \wedge D(x)\}$$

together with the ABox

$$\mathcal{A} = \{C(a)\}$$

Adding the assertion $D(a)$ into $\mathcal{A}$ leads to $\mathcal{A}' = \{C(a), D(a)\}$, which is inconsistent w.r.t. $\mathcal{T}$.

We avoid inconsistent results by the next definition.

Definition 8. *(Minimal Instance Insertion) Let $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ be a knowledge base with $\mathcal{T}$ consistent and D a ground atom of the form $A(a)$ or $R(a, b)$. An ABox $\mathcal{A}'$ is called minimal instance insertion of D into $\mathcal{A}$ if*

- *$D \in \mathcal{A}'$,*
- *$(\mathcal{A}' \setminus D) \subseteq \mathcal{A}$,*
- *$\mathcal{T} \cup \mathcal{A}'$ is consistent and there is no $\mathcal{A}''$ with $D \in \mathcal{A}''$ and $(\mathcal{A}' \setminus D) \subset (\mathcal{A}'' \setminus D) \subseteq \mathcal{A}$ and $\mathcal{T} \cup \mathcal{A}''$ is consistent.*

4 $\mathcal{K}^*$-Transformation

We will solve the tasks defined in Section 3 by using the $\mathcal{K}^*$-transformation which will be introduced in this section. As discussed in the introduction we want to use the ABox of the knowledge base as a partial model, which will guide our transformation.

Considering the task of deleting a given instance, we want to determine a minimal set of ABox assertions which have to be deleted in order to prevent the instance from being contained in the deductive closure of the knowledge base. The idea of the transformation we are about to use was introduced in [3]. We replace occurrences of an atom $A(a)$ in a clause by $\neg \mathit{NegA}(a)$. This transformation can be seen as switching the sides in the clause representation of DL-clauses. This makes sense, when a bottom-up proof procedure like E-KRHyper is used: a fact $A(a) \leftarrow$ changes the side and the clause becomes $\leftarrow \mathit{NegA}(a)$. As a consequence $A(a)$ is not derived explicitly. It is assumed to be in the model until the opposite has to be derived.

Deducing an atom $\mathit{NegA}(a)$ means that we have to revise the ABox and that we have to remove atom $A(a)$ from the ABox. By using this transformation we only need to calculate the atoms we have to remove from the ABox. All remaining atoms will be kept in the ABox. Since it is reasonable to expect the ABox to be very large, it is advantageous to calculate only the deviation from the original ABox.

Definition 9. *The Neg and the ABox function map atoms to renamed atoms:*

- *For atomic concepts A and an individual or variable a:*
  - $\mathit{Neg}(A(a)) = \mathit{NegA}(a)$
  - $\mathit{ABox}(A(a)) = \mathit{ABoxA}(a)$
- *For atomic roles R and individuals or variables a, b:*
  - $\mathit{Neg}(R(a,b)) = \mathit{NegR}(a,b)$
  - $\mathit{ABox}(R(a,b)) = \mathit{ABoxR}(a,b)$

We slightly abuse notation by using the *Neg* function to rename atomic concepts and atomic roles: for B an atomic concept or an atomic role: $\mathit{Neg}(B) = \mathit{NegB}$. Further for a set of atoms P, $\mathit{Neg}(P)$ is defined as: $\mathit{Neg}(P) = \{\mathit{Neg}(A) \mid A \in P\}$. So we can use the *Neg* function to rename atoms, sets of atoms and atomic concepts and roles.

Definition 10. [1] *(Renaming) Let DL be a set of DL-clauses and S a set of atomic concepts and atomic roles. Let $C \in DL$ be $C = \mathbf{H} \leftarrow \mathbf{B}$. Then $R_S(C)$, the* renaming of *C w.r.t. S is*

[1] Due to the helpful remarks of an anonymous reviewer of the DL Workshop, this definition was revised. These changes also affect the results presented in the experiments.

$$
\begin{aligned}
&\mathsf{R}_S(C) = \\
&\{C\} && (1)\\
&\cup \\
&\{(\bigvee_{\substack{A\in\mathbf{H},\\ \sigma(A)\notin S}} A) \vee (\bigvee_{\substack{B\in\mathbf{B},\\ \sigma(B)\in S}} Neg(B)) \leftarrow (\bigwedge_{\substack{B\in\mathbf{B},\\ \sigma(B)\notin S}} B) \wedge (\bigwedge_{\substack{A\in\mathbf{H},\\ \sigma(A)\in S}} Neg(A))\} && (2)\\
&\cup \\
&\{\bot \leftarrow R(x,y) \wedge NegR(x,y) \mid \exists A \in (\mathbf{H}\cup\mathbf{B}) \textit{ with } \sigma(A) = R \in S \textit{ or } \exists A \in \mathbf{H} \\
&\qquad \textit{of the form } A = \exists R.C(z) \textit{ and } R \in S\} && (3)\\
&\cup \\
&\{\bot \leftarrow D(x) \wedge NegD(x) \mid \exists A \in (\mathbf{H}\cup\mathbf{B}) \textit{ with } \sigma(A) = D \in S \textit{ or } \\
&\qquad \exists A \in \mathbf{H} \textit{ of the form } A = \exists R.D(z) \textit{ and } R \in S\} && (4)
\end{aligned}
$$

For a set of DL-clauses DL, the renaming $\mathsf{R}_S(DL)$ w.r.t. S is defined as the union of the renaming of all its clauses.

Note that renaming is a bijective function on a set of DL-clauses. Further renaming can be performed in time linear to the size of the set of DL-clauses times the size of S.

The next proposition states the fact, that renaming preserves satisfiability. Furthermore given a model for a set of DL-clauses DL, it is possible to calculate a model for the renamed set of DL-clauses $\mathsf{R}_S(DL)$ and vice versa.

Proposition 1. *(Renaming Models) Let DL be a set of DL-clauses, S a set of atomic concepts and atomic roles and $\mathcal{I}$ an interpretation. Then $\mathcal{I} \models DL$ iff $\mathcal{I}^S \models \mathsf{R}_S(DL)$, where $\mathcal{I}^S$ and $\mathcal{I}$ have the same domain and the same interpretation of individuals. In addition to that the interpretation of all roles and concepts occurring in DL coincide. Further $(Neg(B))^{\mathcal{I}^S} = \overline{B^{\mathcal{I}}}$ for all concepts names $B \in S$ and $(Neg(R))^{\mathcal{I}^S} = \overline{R^{\mathcal{I}}}$ for all atomic roles $R \in S$.*

Definition 11. *($\mathcal{K}^*$-Transformation) Let $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ be a knowledge base (where $\mathcal{T}$ is consistent). Let S be the set of atomic concepts and atomic roles occurring in $\mathcal{A}$. Then $\mathcal{K}^*$ is the clause set obtained by renaming $\Xi(\mathcal{T})$ w.r.t. S and adding the set of DL-clauses $\{ABox(A) \leftarrow \top \mid \textit{for all assertions } A \in \mathcal{A}\}$.*

We have to add $\{ABox(A) \leftarrow \top \mid \text{for all assertions } A \in \mathcal{A}\}$ to the result of renaming for two reasons: first of all we have to introduce the individuals occurring in the ABox to the theorem prover. Furthermore it is helpful to calculate minimal deletions.

Proposition 2. *Let $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ be a knowledge base and S be the set of atomic concepts and atomic roles occurring in $\mathcal{A}$ and $\mathcal{T}$. Then $\Xi(\mathcal{T})$, $\mathsf{R}_S(\Xi(\mathcal{T}))$ and $\mathcal{K}^*$ are equisatisfiable.*

Example 4. We consider the set of DL-clauses given in Example 1 together with the ABox $\mathcal{A} = \{B(a), D(a), C(b), R(b,b), R(a,a)\}$. Then $S = \{B, D, C, R\}$. Renaming the DL-clauses given in Example 1 w.r.t. S leads to $\mathcal{K}^*$:

$$\begin{aligned}
\exists R.C(x) &\leftarrow B(x).\\
\exists R.C(x) \vee \mathit{NegB}(x) &\leftarrow \top.\\
D(x) &\leftarrow R(x,y) \wedge C(y).\\
\mathit{NegR}(x,y) \vee \mathit{NegC}(y) &\leftarrow \mathit{NegD}(x).\\
C(x) &\leftarrow D(x).\\
\mathit{NegD}(x) &\leftarrow \mathit{NegC}(x).\\
\bot &\leftarrow R(x,y) \wedge \mathit{NegR}(x,y).\\
\bot &\leftarrow C(x) \wedge \mathit{NegC}(x).\\
\bot &\leftarrow B(x) \wedge \mathit{NegB}(x).\\
\bot &\leftarrow D(x) \wedge \mathit{NegD}(x).\\
\mathit{ABoxB}(a) &\leftarrow \top.\\
\mathit{ABoxD}(a) &\leftarrow \top.\\
\mathit{ABoxC}(b) &\leftarrow \top.\\
\mathit{ABoxR}(b,b) &\leftarrow \top.\\
\mathit{ABoxR}(a,a) &\leftarrow \top.
\end{aligned}$$

In the worst case the $\mathcal{K}^*$-transformation quadruples the size of a set of DL-clauses: S is the set of all concepts/roles occurring in the clause set. The ABox contains b assertions and the TBox consists of a single clause: $C = H_1 \vee \ldots \vee H_i \leftarrow B_1 \wedge \ldots \wedge B_j$ with $n = i + j$. W.l.o.g. the symbols of all atoms occurring in C are concepts. This set of DL-clauses has the size $n + b$. Renaming results in:

$$\begin{aligned}
\{H_1 \vee \ldots \vee H_i &\leftarrow B_1 \wedge \ldots \wedge B_j,\\
\mathit{Neg}(B_1) \vee \ldots \vee \mathit{Neg}(B_j) &\leftarrow \mathit{Neg}(H_1) \wedge \ldots \wedge \mathit{Neg}(H_i),\\
\bot &\leftarrow \sigma(H_1)(x) \wedge \mathit{Neg}(\sigma(H_1))(x),\\
&\vdots\\
\bot &\leftarrow \sigma(H_i)(x) \wedge \mathit{Neg}(\sigma(H_i))(x),\\
\bot &\leftarrow \sigma(B_1)(x) \wedge \mathit{Neg}(\sigma(B_1))(x),\\
&\vdots\\
\bot &\leftarrow \sigma(B_j)(x) \wedge \mathit{Neg}(\sigma(B_j))(x),\\
&\cup\{\mathit{ABox}(A) \leftarrow \top \mid \text{for all assertions } A \in \mathcal{A}\}
\end{aligned}$$

The first clause is the original clause from the TBox. Its size is n. The second clause is created by renaming and has size n. Then n clauses of size 2 follow. At the end of the clause set are b clauses of the form $\mathit{ABox}(A)$ each of size 1. All in all the resulting set of clauses has the size $n + n + 2 * n + b \leq 4 * (n + b)$, which is four times higher than the size of the original set of DL-clauses.

5 Using the $\mathcal{K}^*$-Transformation for ABox Evolution

Firstly we address deletion: Recall that according to the definition of the *Neg* function, $Neg(\mathcal{A})$ is defined as $\{Neg(A) \mid A \in \mathcal{A}\}$. Next we show how to use $Neg(\mathcal{A})$-minimal models to calculate minimal instance deletions. For a given model M we construct $Del(M) = \{A \in \mathcal{A} \mid Neg(A) \in M\}$. Intuitively $Del(M)$ constitutes the set of ABox assertions supposed to be deleted from the ABox to obtain a minimal instance deletion.

Theorem 1. *Let $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ be a knowledge base where $\mathcal{T}$ is consistent, S the set of atomic concepts and atomic roles occurring in $\mathcal{A}$, and D a delete request. Let M^S be a $Neg(\mathcal{A})$-minimal model for $\mathcal{K}^* \cup \{Neg(D) \leftarrow \top\}$. Then $\mathcal{A} \setminus Del(M^S)$ is a minimal instance deletion of D from $\mathcal{A}$.*

Proof by first showing $\mathcal{T} \cup (\mathcal{A} \setminus Del(M^S)) \not\models D$ by constructing a model according to Proposition 1 for $\mathcal{T} \cup (\mathcal{A} \setminus Del(M^S)) \cup \{\leftarrow D\}$ from M^S. And then showing that there is no $Del' \subset Del(M^S)$ with $\mathcal{T} \cup (\mathcal{A} \setminus Del') \not\models D$. See [9] for details.

Example 5. Now we delete $D(a)$ from the DL-clauses of our running example. For this, we add the clause $NegD(a) \leftarrow$ to the result of the $\mathcal{K}^*$ transformation given in Example 4. For lack of space we only give the relevant part of a $Neg(\mathcal{A})$ minimal model for this set of clauses:

$$M = \{ABoxB(a), ABoxD(a), ABoxC(b), ABoxR(b,b), ABoxR(a,a),$$
$$NegD(a), NegB(a), \ldots\}$$

This model gives us the minimal deletion: $\mathcal{A}' = \{C(b), R(b,b), R(a,a)\}$

Note that Theorem 1 can further be used for minimal deletion of a delete request D which belongs to the deductive closure of the knowledge base but is not contained in the ABox $\mathcal{A}$. (Meaning $D \notin \mathcal{A}$ but $\mathcal{T} \cup \mathcal{A} \models D$). In this case we only have to make sure, that $\sigma(D) \in S$. If $\sigma(D)$ does not occur in $\mathcal{A}$ we have to add D manually to S in order to render the instance deletion possible.

The $\mathcal{K}^*$-transformation introduced in Definition 11 can be used to repair an ABox, which is inconsistent w.r.t. its TBox. The basic idea is to replace each occurrence of $\perp$ in $\mathcal{T}$ by a new atom *false* and further add *false* to S. After that, we use the $\mathcal{K}^*$-transformation and construct the minimal instance deletion of *false* from the ABox. The resulting ABox is a minimal ABox repair.

Lemma 1. *Let $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ be a knowledge base with consistent $\mathcal{T}$, $\mathcal{T}_{false}$ the TBox obtained from $\mathcal{T}$ by replacing every occurrence of $\perp$ by false, $\mathcal{A}_{false}$ be $\mathcal{A} \cup \{false\}$ and $\mathcal{K}_{false} = (\mathcal{T}_{false}, \mathcal{A}_{false})$. Let S be the set of atomic concepts and roles occurring in $\mathcal{A}$ and $\mathcal{T}$ plus false. Then there is a $Neg(\mathcal{A})$-minimal model for $\mathcal{K}^*_{false} \cup \{Negfalse \leftarrow \top\}$.*

Corollary 1. *Let $\mathcal{K} = (\mathcal{T}, \mathcal{A})$, $\mathcal{T}_{false}$ and S be defined as in Lemma 1. Then $\mathcal{A} \setminus Del(M)$ is a minimal ABox repair for $\mathcal{A}$ for all $Neg(\mathcal{A})$-minimal models M for $\mathcal{K}^*_{false} \cup \{Negfalse \leftarrow \top\}$.*

Corollary 1 follows immediately from Theorem 1 with $D = \mathit{false}$. See [9] for both proofs. Lemma 1 together with Corollary 1 implies, that such a minimal ABox repair can always be constructed.

Next we consider a special case of deletion. For a given knowledge base $\mathcal{K} = (\mathcal{T}, \mathcal{A})$, Theorem 1 can only be used to construct a minimal instance deletion of D from $\mathcal{A}$ if $\mathcal{K}^* \cup \{\mathit{Neg}(D) \leftarrow \top\}$ is satisfiable. However if $\mathcal{K}^* \cup \{\mathit{Neg}(D) \leftarrow \top\}$ is not satisfiable, there is no $\mathit{Neg}(\mathcal{A})$-minimal model for $\mathcal{K}^* \cup \{\mathit{Neg}(D) \leftarrow \top\}$ and therefore we cannot use Theorem 1 for the construction of a minimal instance deletion.

Example 6. Let $\mathcal{T}$ be a TBox containing the assertion $\top \sqsubseteq C$ stating that everything belongs to the concept C. This corresponds to the DL-clause $C(x) \leftarrow \top$. Let us further consider the ABox: $\mathcal{A} = \{C(a), B(a), C(b), B(b)\}$ The $\mathcal{K}^*$-transformation leads to

$$\begin{aligned}\mathcal{K}^* = \{&C(x) \leftarrow \top,\\ &\bot \leftarrow \mathit{NegC}(x),\\ &\bot \leftarrow C(x) \wedge \mathit{NegC}(x),\\ &\mathit{ABoxC}(a),\\ &\mathit{ABoxB}(a),\\ &\mathit{ABoxC}(b),\\ &\mathit{ABoxB}(b)\}\end{aligned}$$

If we now want to delete $C(a)$ from $\mathcal{A}$, we have to construct $\mathit{Neg}(\mathcal{A})$-minimal models for $\mathcal{K}^* \cup \{\mathit{NegC}(a) \leftarrow \top\}$. However $\mathcal{K}^* \cup \{\mathit{NegC}(a) \leftarrow \top\}$ is unsatisfiable. So we are not able to construct a minimal instance deletion of $C(a)$ from $\mathcal{A}$ using Theorem 1. Taking a closer look at the TBox reveals the problem: the TBox claims, that everything has to belong to the concept C. So the only way to remove $C(a)$ from $\mathcal{A}$ is to remove individual a entirely from the ABox.

The next Theorem uses this idea and states how to construct minimal ABox deletions in the case that $\mathcal{K}^* \cup \{\mathit{Neg}(D) \leftarrow \top\}$ is unsatisfiable. Please note that the requirement of $\mathcal{T} \cup \mathcal{A}$ being consistent in the next theorem is not a limitation since we are always able to repair an ABox which is inconsistent with respect to its TBox using Corollary 1.

Theorem 2. *Let $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ be a knowledge base with $\mathcal{T} \cup \mathcal{A}$ consistent. Let further S be the set of atomic concepts and roles occurring in $\mathcal{A}$ and let D be a delete request with $\mathit{Ind}(D)$ the set of individuals occurring in D. If $\mathcal{K}^* \cup \{\mathit{Neg}(D) \leftarrow \top\}$ is unsatisfiable, then $\mathcal{A}' \subseteq \mathcal{A}$ is a minimal instance deletion of D from $\mathcal{A}$, where $\mathcal{A}'$ is obtained from $\mathcal{A}$ by removing all ABox assertions containing an individual from $\mathit{Ind}(D)$.*

With the help of Theorem 1 and 2 we are now able to construct minimal instance-level deletions independent from the satisfiability of $\mathcal{K}^* \cup \{\mathit{Neg}(D) \leftarrow \top\}$.

Next we address the insertion of an assertion into an existing ABox. This can be obtained, by first adding the assertion to the ABox and afterwards constructing all possible minimal repairs for the resulting ABox. If the added assertion is not contained in any of these minimal ABox repairs, then it is not possible to insert the assertion into the ABox without rendering the ABox inconsistent w.r.t. its TBox. If there is a minimal repair containing the added assertion, then the insertion is possible and the respective minimal ABox repair gives us the result of the insertion.

Example 7. In the Example 3, we can repair $\mathcal{A}'$. There are two minimal ABox repairs for $\mathcal{A}'$: $\mathcal{A}'' = \{C(a)\}$ and $\mathcal{A}''' = \{D(a)\}$. The first minimal repair corresponds to deleting the previously inserted $D(a)$ and therefore is not desirable. The second minimal repair however allows us to keep the inserted assertion.

6 Experimental Results

We developed a prototypical implementation for deletion of ABox assertions using the $\mathcal{K}^*$-transformation. We use the E-KRHyper theorem prover to construct the $Neg(\mathcal{A})$-minimal models which lead us to the minimal deletions. Another theorem prover able to handle DL-clauses is HermiT [17]. However HermiT is not able to calculate $Neg(\mathcal{A})$-minimal models. This is why we chose the E-KRHyper theorem prover for our implementation. All tests were carried out on a computer featuring an AMD Phenom X6 1090T @ 3.2GHz and 8GB RAM. To the best of our knowledge, there is no system performing deletion of ABox assertions as described in this paper. This is why we cannot compare our system to another system.

In Section 4 we briefly discussed the complexity of the entire $\mathcal{K}^*$-transformation. There is a linear blow up of the knowledge base and there is also polynomial time complexity for performing the transformation. The real costs for performing the deletion, insertion and repair are caused by the theorem prover which has to compute the $Neg(\mathcal{A})$-minimal models. For an overview about this issue we refer to [8]. We use E-KRHyper for the construction of $Neg(\mathcal{A})$-minimal models. For this we extended E-KRHyper by a feature to construct Γ-minimal models in a bottom-up way. This extension renders it possible to give E-KRHyper a set of DL-clauses together with a set of predicate symbols P and an integer i. Then E-KRHyper only constructs models containing at most i instances of P predicates. During reasoning, E-KRHyper discards all models with more than i instances of P predicates. If E-KRHyper is not able to find a model with i or less instances of P predicates, it terminates by stating that the maximal number of instances is reached. We use this feature to construct $Neg(\mathcal{A})$-minimal models: for S the set of concepts and roles occurring in the knowledge base, we first call E-KRHyper with $\mathcal{K}^*$, the set $Neg(S)$ and $i = 1$. We successively increase i until E-KRHyper either gives us a model or a proof for the unsatisfiability of the set of DL-clauses. This ensures that the first model given by E-KRHyper is a $Neg(\mathcal{A})$-minimal model.

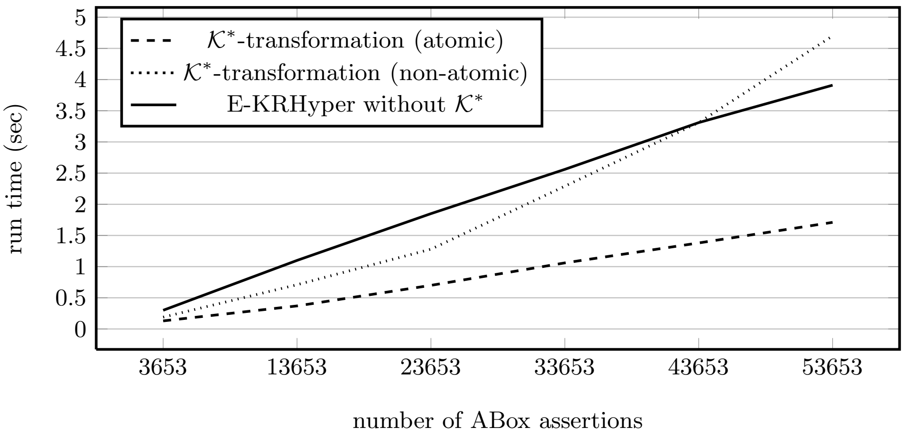


Fig. 1. Time used for atomic and non-atomic deletions

We use the $\mathcal{ALHI}$ ontology VICODI [2] for testing our approach. The smallest version of this ontology consists of 223 axioms in the TBox and RBox and 53653 ABox assertions. The larger versions of this ontology are generated by duplicating the assertions of the original ABox several times and changing the names of the individuals in the assertions. Unfortunately the repetitive structure of the larger versions of the ontology, resulting from this construction, is not suitable to test the efficiency of our approach. This is why we focus on the smallest version of the VICODI ontology. We construct different versions with increasing numbers of ABox assertions. The TBox and RBox remain unchanged. For each version of the so created ontologies we used 1000 different ABox assertions as a delete request D, calculated the $\mathcal{K}^*$-transformation and used E-KRHyper to calculate the minimal ABox deletion. In Figure 1 we show the results for the different ABox sizes we considered. For most of the delete request considered, it was sufficient to only remove the delete request itself from the ABox. We call those cases atomic deletions. If more than one ABox assertion has to be deleted, we speak of non-atomic deletions. Figure 1 gives information on the average time used for a delete request leading to an atomic deletion as well as leading to a non-atomic deletion. Another way to determine atomic deletions is to use E-KRHyper without the $\mathcal{K}^*$-transformation. If we want to test, if D can be removed from the ABox by deleting only D from the ontology KB, we can test $KB \setminus \{D\} \cup \{\neg D\}$ for satisfiability using E-KRHyper. Satisfiability of $KB \setminus \{D\} \cup \{\neg D\}$ implies, that $KB \setminus \{D\} \not\models D$. Meaning that D can be deleted atomically. Note that this test can only be used for atomic deletions and is completely useless for the calculation of non-atomic deletions. You can find the time used for those atomic deletions computed by E-KRHyper without the $\mathcal{K}^*$-Transformation in Figure 1. Comparing the lines for E-KRHyper and atomic deletions using the $\mathcal{K}^*$-transformation shows, that the $\mathcal{K}^*$-Transformation is faster in calculating atomic deletions.

[2] `http://www.vicodi.org`

In addition to that the $\mathcal{K}^*$-Transformation is able to calculate non-atomic deletions as well and is therefore better suited for deletion than E-KRHyper. Figure 1 reveals another nice property of the $\mathcal{K}^*$-transformation: increasing the size of the ABox only leads to a harmless increase of the time necessary to calculate the minimal deletion. We owe this property to the fact, that we only calculate the deviation from the original ABox. For the calculation of non-atomic deletions more than one run of E-KRHyper is necessary. This explains why non-atomic deletions take longer than atomic deletions. However the time necessary to calculate a non-atomic deletion only increases moderately when the size of the ABox under consideration is increased.

7 Conclusion and Future Work

In this paper we give a semantically guided compilation technique, the so called $\mathcal{K}^*$-transformation, for $\mathcal{SHI}$ knowledge bases. The transformed knowledge base is equisatisfiable to the original one. A theorem prover can be used for the computation of the necessary actions for deletion, insertion and repair from the result of the $\mathcal{K}^*$-transformation. Especially theorem provers based on a hypertableau calculus are suited for these computations. The approach is implemented and we introduced first experimental results using the theorem prover E-KRHyper.

In future work, we want to extend our implementation to enable it to do ABox repair and insertion of assertions as well.

Since E-KRHyper is able to handle the DL $\mathcal{SHIQ}$, we plan to extend our approach to qualified number restrictions.

References

1. Aravindan, C., Baumgartner, P.: Theorem proving techniques for view deletion in databases. Journal of Symbolic Computation 29, 2000 (2000)
2. Baader, F., Peñaloza, R.: Axiom pinpointing in general tableaux. J. Log. Comput. 20(1), 5–34 (2010)
3. Baumgartner, P., Fröhlich, P., Furbach, U., Nejdl, W.: Semantically Guided Theorem Proving for Diagnosis Applications. In: Pollack, M.E. (ed.) IJCAI 1997, Nagoya. Morgan Kaufmann (1997)
4. Bender, M., Pelzer, B., Schon, C.: System description: E-KRHyper 1.4 - Extensions for unique names and description logic. In: Bonacina, M.P. (ed.) CADE 2013. LNCS, vol. 7898, pp. 126–134. Springer, Heidelberg (2013)
5. Calvanese, D., Kharlamov, E., Nutt, W., Zheleznyakov, D.: Updating aboxes in DL-Lite. In: Laender, A.H.F., Lakshmanan, L.V.S. (eds.) AMW. CEUR Workshop Proceedings, vol. 619. CEUR-WS.org (2010)
6. Chu, H., Plaisted, D.A.: Semantically guided first-order theorem proving using hyper-linking. In: Bundy, A. (ed.) CADE 1994. LNCS, vol. 814, pp. 192–206. Springer, Heidelberg (1994)
7. De Giacomo, G., Lenzerini, M., Poggi, A., Rosati, R.: On instance-level update and erasure in description logic ontologies. J. Log. and Comput. 19 (2009)

8. Dix, J., Furbach, U., Niemelä, I.: Nonmonotonic reasoning: Towards efficient calculi and implementations. In: Robinson, J.A., Voronkov, A. (eds.) Handbook of Automated Reasoning, pp. 1241–1354. Elsevier, MIT Press (2001)
9. Furbach, U., Schon, C.: Semantically guided evolution of SHI aboxes. Reports of the Faculty of Informatics 4/2013, Universität Koblenz-Landau (2013), `http://www.uni-koblenz.de/FB4/Publications/Reports`
10. Grau, B.C., Ruiz, E.J., Kharlamov, E., Zhelenyakov, D.: Ontology evolution under semantic constraints. In: Proc. of the 13th Int. Conference on Principles of Knowledge Representation and Reasoning (2012)
11. Halashek-Wiener, C., Parsia, B., Sirin, E.: Description logic reasoning with syntactic updates. In: Meersman, R., Tari, Z. (eds.) OTM 2006, Part I. LNCS, vol. 4275, pp. 722–737. Springer, Heidelberg (2006)
12. Horridge, M., Parsia, B., Sattler, U.: Laconic and precise justifications in OWL. In: Sheth, A.P., Staab, S., Dean, M., Paolucci, M., Maynard, D., Finin, T., Thirunarayan, K. (eds.) ISWC 2008. LNCS, vol. 5318, pp. 323–338. Springer, Heidelberg (2008)
13. Horridge, M., Parsia, B., Sattler, U.: Explaining inconsistencies in OWL ontologies. In: Godo, L., Pugliese, A. (eds.) SUM 2009. LNCS, vol. 5785, pp. 124–137. Springer, Heidelberg (2009)
14. Lembo, D., Lenzerini, M., Rosati, R., Ruzzi, M., Savo, D.F.: Inconsistency-tolerant semantics for description logics. In: Hitzler, P., Lukasiewicz, T. (eds.) RR 2010. LNCS, vol. 6333, pp. 103–117. Springer, Heidelberg (2010)
15. Lenzerini, M., Savo, D.F.: On the evolution of the instance level of DL-Lite knowledge bases. In: Rosati, R., Rudolph, S., Zakharyaschev, M. (eds.) Description Logics. CEUR Workshop Proceedings, vol. 745. CEUR-WS.org (2011)
16. Liu, H., Lutz, C., Milicic, M., Wolter, F.: Foundations of instance level updates in expressive description logics. Artificial Intelligence 175(18), 2170–2197 (2011)
17. Motik, B., Shearer, R., Horrocks, I.: Optimized Reasoning in Description Logics Using Hypertableaux. In: Pfenning, F. (ed.) CADE 2007. LNCS (LNAI), vol. 4603, pp. 67–83. Springer, Heidelberg (2007)
18. Plaisted, D.A., Greenbaum, S.: A structure-preserving clause form translation. J. Symb. Comput. 2(3), 293–304 (1986)
19. Rosati, R.: On the complexity of dealing with inconsistency in description logic ontologies. In: IJCAI 2011. AAAI Press (2011)
20. Schlobach, S., Cornet, R.: Non-standard reasoning services for the debugging of description logic terminologies. In: Gottlob, G., Walsh, T. (eds.) IJCAI. Morgan Kaufmann (2003)

PSYCHE: A Proof-Search Engine Based on Sequent Calculus with an LCF-Style Architecture

Stéphane Graham-Lengrand

CNRS - École Polytechnique, France

Abstract. PSYCHE is a modular proof-search engine designed for either interactive or automated theorem proving, and aiming at two things: a high level of confidence about the output of the theorem proving process and the ability to apply and combine a wide range of techniques. It addresses the first aim by adopting and extending the LCF architecture to guarantee, using private types, not only the correctness but also the completeness of proof search. It addresses the second by offering a much more appropriate API than just the primitives corresponding to the inference rules of the logic in natural deduction: it uses instead a focused sequent calculus for polarised classical logic. Finally, PSYCHE features the ability to call decision procedures such as those used in Sat-Modulo-Theories solvers. We therefore illustrate PSYCHE by using it for SMT-solving.

1 PSYCHE in Brief

PSYCHE [11], the *Proof-Search factorY for Collaborative HEurisics*, is a modular platform for automated or interactive theorem proving, built on an architecture (similar to LCF) where a small *kernel* interacts with *plugins* and *decision procedures*:

– The kernel is based on a proof-search engine *à la* Prolog, offering an API to perform incremental and goal-directed constructions of proof-trees in (a standard but carefully chosen) Sequent Calculus.
– PSYCHE can produce proof objects (and print them in LaTeX format).
– Plugins can be programmed to drive the kernel, using its API, through the search space towards an answer *provable* or *not provable*; soundness of the answer only relies on the kernel via the use of a private type for answers (similar to LCF's *theorem* type).
– Plugins can be interactive.
– PSYCHE offers a *memoisation* feature to help program efficient plugins.
– The kernel is parameterised by a procedure deciding the consistency of collections of literals with respect to a background theory, just as in SAT-modulo-theories (SMT) solvers.

The current version 1.5 of PSYCHE features a kernel designed for propositional logic modulo theories (same logic as that of $\mathsf{DPLL}(\mathcal{T})$ used in SMT-solving), and decision procedures for the empty theory and Linear Rational Arithmetic (LRA). It is a program of about 4200 lines of OCaml 4.00, using hash-consing in most data structures for efficiency.

2 Motivation

Psyche's architecture is designed for the ambition of allowing various theorem proving techniques (generic or problem-specific) to collaborate on a common platform, whilst giving high confidence in the answers produced.

Interfacing the numerous techniques and tools available for theorem proving is legitimately receiving a lot of attention: Automated Theorem Provers, SAT/SMT-solvers, Proof assistants, etc. While trust is already an issue even for a tool running on its own, it becomes even more of an issue when different tools interact. *Proof-checking* is one way of addressing this, being permissive in the algorithms used for theorem proving, as long as they output some proof objects that can be checked. Another way is the LCF-style [7], where only a small kernel of primitives needs to be trusted, and anything smarter (e.g. the interaction between sophisticated techniques) boils down to calls to the primitives.

In the context of proof-checking, a natural way to interact with different (already implemented) techniques, is the black box approach, where an external tool is called and its output is converted back into a proof that can be checked by the system [2,3]. It is somewhat more surprising that, despite the highly programmable possibilities of the LCF architecture, the most successful integration of automated reasoning techniques in an LCF-based proof assistant such as Isabelle [8] seems to also use variants of the black box approach (as very impressively demonstrated by Sledgehammer) [12,10].

Psyche aims at producing answers that are correct by construction, not having to rely on proof-checking; it therefore adopts the LCF philosophy (although it can produce proof objects), also because having a simple trusted kernel is a convenient starting point for different techniques to collaborate. But the goal here is to open the black boxes and program their algorithms directly with calls to the kernel's API, as plugins for Psyche.

Such a deeper level of integration opens up the perspective of interleaving the use of different techniques: An external tool requires an input problem that it can entirely treat; but implementing the *steps* of its algorithm as small progressions in the search-space covered by the main system, allows more possibilities, such as running the technique *up-to-a-point*, where a switch to another technique may be appropriate (e.g. depending on newly generated goals).

The challenge is for the kernel to offer an appropriate API of proof-search or proof-construction primitives, to allow the efficient implementation of theorem proving techniques as plugins. Most LCF-style systems offer primitives corresponding to the inference rules of Natural deduction, or a Hilbert-style system. This is a very fine-grained level, that leaves most (if not all) of the work to the plugin and makes its implementation cumbersome: these formalisms are more about proof-construction than proof-search.

Psyche makes the choice of a bigger grain, and leaves to the kernel some real proof-search computation, but where no decision needs to be made. For this we use a *focused* sequent calculus $\mathsf{LK}^p(\mathcal{T})$ [4,5], which extends the logic programming paradigm to *polarised* classical logic modulo a background theory $\mathcal{T}$. Polarities and Focusing [6,1] are tools that can be used to describe effective proof-search strategies in Sequent Calculus, hugely narrowing the search-space offered by Gentzen's original rules. In our case, they also specify a sensible division of labour between PSYCHE's kernel and PSYCHE's plugins, redesigning the standard LCF-style API.

3 Overview and General Architecture

The kernel contains the mechanisms for exploring the proof-search space in a sound and complete way, taking into account branching and backtracking. It has no strategy regarding the order in which branches are explored, and this lack of intelligence makes its code rather short. If it reaches a proof, then that proof is correct by construction, and if the entire search space is explored and no proof is found, then the kernel correctly outputs that no proof exists.

The plugins then drive the kernel by specifying in which order the branches of the search space should be explored and to which depth, something that is expected to depend on the kind of problem that is being treated. The quality of the plugin is how fast it drives the kernel towards a answer *Provable*/*NotProvable*.

This already departs from the traditional LCF-style in that some actual proof-search computation is performed in the kernel, not just atomic steps of proof-construction:

In traditional LCF, each inference rule of the logic (as on the right-hand side) will give rise to a primitive of the kernel's API:

$$\frac{\mathsf{prem}_1 \quad \dots \quad \mathsf{prem}_n}{\mathsf{conc}}\,\mathsf{name}$$

```
name: thm -> ··· -> thm -> thm
```

In PSYCHE's kernel, such an inference rule will be wrapped in the kernel's unique API primitive:

```
machine: statement -> output
```

such that `machine(conc)` will trigger the recursive calls `machine(prem_1)`,..., `machine(prem_n)`.

PSYCHE's general architecture is illustrated by its main top-level call (slightly reworded for clarity):

```
Plugin.solve(Kernel.machine(Parser.parse input))
```

PSYCHE has a collection of parsers (currently one for DIMACS and one for SMT-Lib2) and calls the appropriate one on PSYCHE's input. The resulting abstract syntax tree is fed to the kernel's `machine` function that will initiate the search. This produces a value of type `output` that is given to the plugin to work with, so as to finally produce an answer *provable* or *not provable*. This could give the impression that the plugin performs computation after the kernel has finished its work, but this is not quite true, as illustrated by the nature of type `output`:

```
type output = Final of answer | Fake of coin -> output
```

which describes the kernel as a *slot machine*: when it is run, it outputs

- either a final answer *provable* or *not provable*
- or a fake output that represents unfinished computation: in order for computation to continue, the plugin needs to "insert another coin in the slot machine"; proof-search will then resume according to the inserted coin.

To summarise, the kernel performs proof-search as long as there is no decision to be made (on which backtrack may later be needed), and when it hits such a point, it stops and asks for another coin to indicate how to proceed next. The plugin drives the kernel in the exploration of the proof-search space by inserting carefully chosen coins, hoping that one day the machine will stop with the jackpot: a value of the form `Final(...)`.

Now while this architecture somewhat departs from LCF, it does share with it the distrust of anything outside the kernel: when concerned with the soundness of the answer (whichever it be), the plugin is here considered as an adversary, so PSYCHE defines the type `answer` as abstract, i.e. a private type that only the kernel can inhabit (just like the `thm` type of LCF). PSYCHE's type

```
answer = Provable of statement*proof | NotProvable of statement
```

can be read by the plugin and the top-level if need be, but cannot be inhabited by them. That way, a plugin cannot cheat about PSYCHE's answer: the worst it can do is to crash PSYCHE's runs. In PSYCHE as in traditional LCF, inhabitation of the abstract type (in case of PSYCHE, with a value of the form `Provable(...)`) explicitly or implicitly constructs a proof of the statement. But contrary to LCF, PSYCHE also gives guarantees when the output is *not provable*: it can only occur when the kernel has entirely explored the search-space unsuccessfully.

4 PSYCHE's Kernel

As described above, the kernel's API has the slot machine as its only primitive, controlled by the *coins* that are inserted in it. In order for efficient plugins to be conveniently programmed, the kernel's primitive needs to accept a rather expressive range of coins that can specify a smart exploration of the search-space. This depends on the inference system that is used in the kernel for the incremental and bottom-up construction of proof-trees, and on identifying the inference rules that the kernel will perform automatically from those that will pause computation and prompt the plugin for new directions.

For this, our sequent calculus $\mathsf{LK}^p(\mathcal{T})$ [4,5] describing classical logic modulo a theory $\mathcal{T}$ uses *polarities* and *focusing* (see e.g. [1]). The connectives and literals of classical logic are tagged with polarities: $+$ and $-$. Polarities do not affect the (classical) provability of formulae, but only the shape of proofs and therefore the structure of the proof-search space. *Focusing* is the phenomenon whereby the inference rules decomposing the connectives of the same polarity can be chained without losing completeness (thus narrowing the search-space), see [5] for a full description. But in brief, focusing organises the proof-search process as an alternance between two kinds of phases: *synchronous* and *asynchronous*.

- An asynchronous phase decomposes the formulae of the sequent whose main connective is negative, using *invertible* inference rules (the premises are provable if the conclusion is): these represent no backtrack point in proof-search.
- A synchronous phase starts with the selection of a positive formula: the formula and its sub-formulae are then decomposed recursively (before doing anything else in the sequent) as long as these remain positive. When these become negative, another asynchronous phase starts.

We use focusing to divide the labour between PSYCHE's kernel and plugins: The kernel applies the asynchronous steps automatically without any instruction from the plugin, and then stops and asks for another coin describing the next synchronous phase, where smart choices may have to be made (starting with the selection of the positive formula to work on). An important consequence of this division of labour is that **every kernel call terminates**, because the length of each phase is bounded by the size of the formula(e) being decomposed.

The choice of polarities on connectives and literals affects the kernel-plugin interaction. For instance the polarity of $\vee$ will determine whether it is decomposed automatically by the kernel (second rule, asynchronous) or with a smart choice by the plugin (first rule, synchronous):

$$\frac{\Gamma \vdash A_i}{\Gamma \vdash A_1 \vee^+ A_2} \qquad \frac{\Gamma \vdash A_1, A_2, \Gamma'}{\Gamma \vdash A_1 \vee^- A_2, \Gamma'}$$

The code of the kernel is rather small (575 lines) and purely functional. Continuation-Passing-Style (CPS) is used to minimise the use of the stack and provide a natural way to represent the progression of the kernel within the search space: the API function `machine: statement->output` actually wraps the real (tail-)recursive function

```
search: statement->(output->'a)->'a
```

with the identity continuation. Continuations are heavily used for branching and backtracking (e.g. when `search` applies a rule with several premises, it makes a recursive call on one of the branches and stacks up the others in the passed continuation; similarly when the plugin chooses to explore one branch, the kernel records the other ones -forcing in the end the entire exploration of the search-space), and naturally implement a slot machine waiting for its coin.

5 Plugins

A plugin is an OCaml module of a fixed module type declaring a function `solve: output->answer` (again, `answer` is for the plugin an abstract type).

However, it is likey that the sophisticated strategies/heuristics that the plugin is meant to implement rely on some clever choice of data-structures for formulae, sets of formulae, sets of literals. So the plugin and the kernel have to agree on those three data-structures that are communicated both ways during the interaction. In PSYCHE 1.5, the kernel is parameterised by the data-structures, and the plugin provides them.

We first tested Psyche's architecture with a basic plugin `Naive`, which implements collections as lists and inserts the first available coin in the slot machine, whenever asked. This works fine for small tautologies, printable on a screen.

But the first real aim was to capture in Psyche some propositional SAT and SAT-Modulo-Theories solving techniques, making $\mathsf{DPLL}(\mathcal{T})$ technology available in a generic proof-search framework like Psyche. For this we describe in [5] how to see $\mathsf{DPLL}(\mathcal{T})$, canonically expressed as a transition system [9], as a simple bottom-up proof-construction mechanism in $\mathsf{LK}^p(\mathcal{T})$. More practically, every rule of $\mathsf{DPLL}(\mathcal{T})$ can be seen as the insertion of a particular coin in Psyche's slot machine. We implemented this as two different plugins for Psyche: `DPLL_Pat` and `DPLL_WL`. These remain toy plugins, because, although it is now clear from [5] how to perform each rule of $\mathsf{DPLL}(\mathcal{T})$ in Psyche, we still have to decide which rule to apply. This is where the two plugins differ: `DPLL_Pat` looks up the applicability of $\mathsf{DPLL}(\mathcal{T})$'s rules by using Patricia tries to implement sets of clauses, while `DPLL_WL` looks it up using the technique of *watched literals*.

Just like $\mathsf{DPLL}(\mathcal{T})$-based solvers are made efficient by using features such as backjumping and lemma learning, our plugins are made more efficient by the use of memoisation, which avoids re-doing, for some open branch, the same steps as those used in a previously completed branch. Psyche 1.5 therefore offers a memoisation module, to be used by plugins to record values of (the abstract) type `answer`. And the kernel's slot machine accepts from the plugin, as a special coin carrying such a value, "here is an already found answer that also applies to the current goal". The kernel accepts the value as closing the current branch (one way or another) **without any proof-checking** (since the abstract type ensures the value came as an earlier output of the kernel); it only checks that the value applies to the current goal. Now for a memoised answer *Provable* to be reusable as often as possible, it is useful to prune the provable sequent, just before it is tabled, from the formulae and literals that were not used in its proof. This is easy to do for the complete proofs of $\mathsf{LK}^p(\mathcal{T})$ (eager weakening are applied a posteriori by inspection of the inductive structure). Psyche's kernel actually performs the pruning on-the-fly whenever an inference is added to complete proofs, so that, whenever it outputs `Final(sequent,proof)`, the sequent is already pruned. This is Psyche's way of performing *conflict analysis*, a key process of SMT-solving.

6 Conclusion and Perspectives

Psyche is run from the command-line, taking as input the indicated file(s) or directory(ies) (or the standard input): `psyche [OPTION]... [FILE/DIR]...` Version 1.5 is distributed with a DIMACS parser, which we used to test Psyche on (propositional) SAT benchmarks, and an SMTLib2 parser (unmodified from the Alt-Ergo prover), which we used to test it on `QF_LRA` benchmarks (making use of the distributed simplex algorithm for LRA). The results are available on Psyche's website [11]. Since Psyche has no ambition to beat state-of-the-art SAT- and SMT-solvers, it works well on small instances but its performance starts declining between 20Kb and 100Kb of input problem size (of course this is no appropriate measure of difficulty): There is no intrinsic problem of scalability, but the current plugins and decision procedures are illustrative toys (the heuristics for applying $\mathsf{DPLL}(\mathcal{T})$ rules in the current plugins are still basic, and so is the decision procedure for LRA -e.g. it is not incremental). What we offer here is a platform and its modularity: anyone with better (or different) heuristics or decision procedures can simply write them as OCaml modules of our predefined module types, and Psyche will seamlessly run with them, keeping the same LCF-style guarantees. Moreover, nothing in Psyche's proof-engine relies on the input being sets of clauses, so Psyche might offer a convenient framework to generalise the known techniques for the satisfiability of formulae in clausal form.

The short-to-medium term plans are as follows:

– **kernel**: handle existential variables and propagate first-order unifiers through branching, construct and store proof-terms rather than whole proofs;
– **theories**: improve the procedure for LRA (e.g. making it incremental) and implement other theories (Congruence Closure, LIA, bit vectors, etc);
– **plugins**: implement a user-interactive plugin asking which coins to insert, improve the $\mathsf{DPLL}(\mathcal{T})$ plugins to better handle non-clausal formulae, and implement other theorem proving techniques as plugins: *analytic tableaux*, *clausal tableaux* (e.g. *connections*) and *resolution* are all done in the theory.

In the long-term, we plan to investigate whether $\mathsf{LK}^p(\mathcal{T})$ may help mixing first-order reasoning with theories (i.e. investigate instantiations in presence of a theory), and prove Psyche's correctness in a proof assistant (since the functional kernel seems small enough and the plugins need not be certified).

Acknowledgements. Contributors of Psyche include S. Graham-Lengrand, A. Mahboubi, A. Bernadet, M. Farooque, Damien Rouhling and M. Vegreville.

References

1. Andreoli, J.M.: Logic programming with focusing proofs in linear logic. J. Logic Comput. 2(3), 297–347 (1992)
2. Armand, M., Faure, G., Grégoire, B., Keller, C., Théry, L., Werner, B.: A modular integration of SAT/SMT solvers to Coq through proof witnesses. In: Jouannaud, J.-P., Shao, Z. (eds.) CPP 2011. LNCS, vol. 7086, pp. 135–150. Springer, Heidelberg (2011)
3. Besson, F., Cornilleau, P.-E., Pichardie, D.: Modular SMT proofs for fast reflexive checking inside Coq. In: Jouannaud, J.-P., Shao, Z. (eds.) CPP 2011. LNCS, vol. 7086, pp. 151–166. Springer, Heidelberg (2011)
4. Farooque, M., Graham-Lengrand, S.: Sequent calculi with procedure calls. Technical report, Laboratoire d'Informatique de l'Ecole Polytechnique (January 2013), `http://hal.archives-ouvertes.fr/hal-00779199`
5. Farooque, M., Graham-Lengrand, S., Mahboubi, A.: A bisimulation between DPLL(T) and a proof-search strategy for the focused sequent calculus. In: Momigliano, A., Pientka, B., Pollack, R. (eds.) Proceedings of the 2013 International Workshop on Logical Frameworks and Meta-Languages: Theory and Practice (LFMTP 2013), Boston, USA. ACM Press (September 2013)

6. Girard, J.-Y.: Linear logic. Theoret. Comput. Sci. 50(1), 1–101 (1987)
7. Gordon, M.J., Milner, R.J., Wadsworth, C.P.: Edinburgh LCF. LNCS, vol. 78. Springer, Heidelberg (1979)
8. The Isabelle theorem prover, `http://isabelle.in.tum.de/`
9. Nieuwenhuis, R., Oliveras, A., Tinelli, C.: Solving SAT and SAT Modulo Theories: From an abstract Davis–Putnam–Logemann–Loveland procedure to DPLL(T). J. of the ACM Press 53(6), 937–977 (2006)
10. Paulson, L.C., Blanchette, J.C.: Three years of experience with Sledgehammer, a practical link between automatic and interactive theorem provers. In: Sutcliffe, G., Schulz, S., Ternovska, E. (eds.) IWIL 2010. EPiC Series, vol. 2, pp. 1–11. EasyChair (2012)
11. Psyche: the Proof-Search factorY for Collaborative HEuristics, `http://www.lix.polytechnique.fr/~lengrand/Psyche/`
12. Weber, T.: SMT solvers: New oracles for the HOL theorem prover. International Journal on Software Tools for Technology Transfer (STTT) 13(5), 419–429 (2011)

Understanding Resolution Proofs through Herbrand's Theorem*

Stefan Hetzl[1], Tomer Libal[2], Martin Riener[3], and Mikheil Rukhaia[4]

[1] Institute of Discrete Mathematics and Geometry, Vienna University of Technology
[2] Microsoft Research - Inria Joint Center / École Polytechnique
[3] Institute of Computer Languages, Vienna University of Technology
[4] Inria Saclay / École Polytechnique

Abstract. Computer-generated proofs are usually difficult to grasp for a human reader. In this paper we present an approach to understanding resolution proofs through Herbrand's theorem and the implementation of a tool based on that approach.

The information we take as primitive is which instances have been chosen for which quantifiers, in other words: an expansion tree. After computing an expansion tree from a resolution refutation, the user is presented this information in a graphical user interface that allows flexible folding and unfolding of parts of the proof.

This interface provides a convenient way to focus on the relevant parts of a computer-generated proof. In this paper, we describe the proof-theoretic transformations, the implementation and demonstrate its usefulness on several examples.

1 Introduction

Computer-generated proofs are often difficult to understand for a human reader. This is usually due to a combination of several factors such as the use of deduction formats more suited for proof search than for proof presentation, overwhelming detail in formal proofs, insufficient user interfaces or also extreme proof length. One of the key problems for understanding formal proofs is to distinguish relevant information from irrelevant information.

In systems with quantifiers, such as classical first-order logic, the most important information is typically which instances have been chosen for which quantifiers. On a purely logical level this insight is embodied by Herbrand's theorem [1,2] which characterizes first-order validity in terms of quantifier instances and propositional validity. In this paper we present an approach to understanding computer-generated proofs through the lens of Herbrand's theorem which takes the information about which instances have been chosen for which quantifiers in a proof as fundamental and thus abstracts from the propositional part of the proof.

* Supported by the joint ANR/FWF-project STRUCTURAL, the joint ANR/FWF-project ASAP, the FWF-project ASCOP, the WWTF-project VRG12-04 and the Vienna PhD school in Informatics.

A data structure which is well-suited for representing this information are expansion trees, introduced by Miller in [3]. A first step in distinguishing relevant from irrelevant information is made by displaying an expansion tree instead of a proof in, e.g., a resolution or a tableau calculus. This removes the propositional layer from a proof. However, not all quantifiers are equally important for understanding a proof, for example often we want to consider a proof modulo a simple theory and are hence not interested in the instances of the axioms of that theory. As such distinctions between important and unimportant information depend on the context and are difficult to automate, we let the user decide what information he wants to see in a graphical user interface by allowing a flexible folding and unfolding of expansion trees by point-and-click interactions.

We have implemented our tool in the GAPT-system[1] which is a framework for data structures, algorithms and user interfaces for analyzing and transforming formal proofs. It contains data structures for example formulas, sequents, resolution proofs, sequent proofs and algorithms, e.g., unification, skolemization, cut-elimination, cut-elimination by resolution [4].

The use of Herbrand's theorem for understanding proofs is a well-established technique. It plays the key role in Luckhardt's (manual) analysis [5] of Roth's theorem where it has been used to obtain polynomial bounds (which were obtained independently and by purely mathematical as opposed to logical methods by Bombieri and van der Poorten in [6]). The extraction and analysis of Herbrand-sequents as described by Hetzl et al. [7] has also been used in the computer-assisted analysis of Fürstenberg's topological proof of the infinity of primes by Baaz et al. [8] which yielded Euclid's original argument via cut-elimination. Herbrand's theorem and methods based on it have furthermore also been used in a number of smaller case studies such as [9] by Baaz et al. or [10] by Urban. In the context of the GAPT-system, Herbrand's theorem also plays a central role for the development of techniques for lemma generation, see Hetzl et al. [11,12], and the tool described in this paper is routinely used there.

The general problem of human-readable presentations of computer-generated proofs is well known and a number of other approaches exist in the literature. Horacek [13] presents an approach to transforming computer-generated proofs to a structure that more closely resembles mathematical proofs in natural language. The TRAMP-system by Meier [14] transforms resolution proofs into natural deduction proofs at the assertion level. The interactive derivation viewer IDV by Trac et al. [15] displays a derivation in the TPTP-format as a directed acyclic graph. In [16], Denzinger and Schulz show how to obtain human-readable proof presentations in the context of distributed equational reasoning.

Closest to our approach is the work of Pfenning [17,18], an algorithm for extracting an expansion tree from a resolution refutation by doing grounding, deskolemization and the change of deduction format from refutation to proof in one pass. The contribution of this paper is twofold: we describe a more modular algorithm that first changes the proof format from a resolution refutation to a positive proof in the sequent calculus and only in a second step extracts an

[1] Generic Architecture for Proof Transformations, `http://www.logic.at/gapt`

expansion tree from the sequent calculus proof thus generated. Aside from higher modularity and less implementation effort in the context of the GAPT-system, this procedure has the practical advantage of allowing a translation to a dag-like sequent calculus proof in case grounding of the resolution refutation is too expensive. Secondly we describe the theory and implementation of a convenient graphical user interface for displaying expansion trees which is available on the web.

2 Expansion Trees

The language of first-order logic we consider consists of *variables* and *n-ary function- and predicate- symbols.* As usual, *terms* are built from variables and function symbols in an inductive fashion. 0-ary function symbols are called *constants. Formulas* are built from predicates and the connectives $\neg, \wedge, \vee, \rightarrow, \forall, \exists$. A *substitution* is a function mapping variables to terms and its application is extended to terms and formulas in the usual way.

Let A be a formula, an occurrence of a subformula B in A is called *negative* if it is in the scope of an odd number of occurrences of $\neg$ and it is called *positive* otherwise. A quantifier occurrence in a formula A is called *strong* if it is a positive universal or a negative existential and *weak* otherwise. So in other words: the strong quantifiers are exactly those that a transformation to negation normal form would turn into universal quantifiers.

Expansion trees were introduced by Miller in [19,3]. These structures record the substitutions for quantifiers in the original formula and the formulas resulting from instantiations. Informally, an expression $QxA(x) +^{t_1} E_1 +^{t_2} \cdots +^{t_n} E_n$ is an expansion tree, where $Q \in \{\forall, \exists\}$ and $t_1, \ldots, t_n$ are terms such that E_i is again an expansion tree representing $A(t_i)$ for all $i = 1, \ldots, n$.

We deviate from [3] in that we define expansion trees for blocks of quantifiers as this is more natural for display purposes. A vector $(x_1, \ldots, x_k)$ of variables is often abbreviated as $\bar{x}$. Also in contrast to [3] we only consider expansion trees of formulas that do not contain strong quantifiers. In our context of automated deduction, strong quantifiers are removed by Skolemization. As Skolem functions often possess a natural mathematical interpretation (see e.g. [8]), we opt for displaying expansion trees that still include the Skolem functions in order to increase their readability.

Definition 1 (Expansion tree). *Expansion trees and a function* Sh *(for* shallow*) which maps an expansion tree to a formula are defined inductively as follows:*

- *A is an atomic expansion tree for A being an atom and $\mathrm{Sh}(A) = A$.*
- *If E_1, E_2 are expansion trees, then so are $\neg E_1$, $E_1 \wedge E_2$, $E_1 \vee E_2$ and $E_1 \rightarrow E_2$ with $\mathrm{Sh}(\neg E_1) = \neg \mathrm{Sh}(E_1)$, $\mathrm{Sh}(E_1 \wedge E_2) = \mathrm{Sh}(E_1) \wedge \mathrm{Sh}(E_2)$, etc.*
- *Let $A(\bar{x})$ be a formula and $\bar{t_1}, \ldots, \bar{t_n}$ $(n \geqslant 1)$ be a list of (vectors of) terms. Let $E_1, \ldots, E_n$ be expansion trees with $\mathrm{Sh}(E_i) = A(\bar{t_i})$ for $1 \leqslant i \leqslant n$ and let $Q \in \{\forall, \exists\}$, then $Q\bar{x}A(\bar{x}) +^{\bar{t_1}} E_1 +^{\bar{t_2}} \cdots +^{\bar{t_n}} E_n$ is an expansion tree with $\mathrm{Sh}(Q\bar{x}A(\bar{x}) +^{\bar{t_1}} E_1 +^{\bar{t_2}} \cdots +^{\bar{t_n}} E_n) = Q\bar{x}A(\bar{x})$.*

We now define another function Dp (for *deep*) which maps an expansion tree to a quantifier-free formula: its full expansion.

Definition 2. Dp *maps an expansion tree to a formula as follows:*

$$\begin{aligned} \mathrm{Dp}(E) &= E \textit{ for an atomic expansion tree } E, \\ \mathrm{Dp}(\neg E) &= \neg \mathrm{Dp}(E), \\ \mathrm{Dp}(E_1 \circ E_2) &= \mathrm{Dp}(E_1) \circ \mathrm{Dp}(E_2) \textit{ for } \circ \in \{\wedge, \vee, \rightarrow\}, \\ \mathrm{Dp}(\exists \bar{x} A +^{\bar{t_1}} E_1 +^{\bar{t_2}} \cdots +^{\bar{t_n}} E_n) &= \mathrm{Dp}(E_1) \vee \cdots \vee \mathrm{Dp}(E_n), \\ \mathrm{Dp}(\forall \bar{x} A +^{\bar{t_1}} E_1 +^{\bar{t_2}} \cdots +^{\bar{t_n}} E_n) &= \mathrm{Dp}(E_1) \wedge \cdots \wedge \mathrm{Dp}(E_n). \end{aligned}$$

In [3], a notion of *expansion proof* was defined from expansion trees using two conditions: *acyclicity* and *tautology*. The acyclicity condition ensures that there are no cycles between the strong quantifier nodes in the expansion tree. Since we deal with formulas that do not contain strong quantifiers, there is no need for this condition.

Definition 3 (Expansion proofs). *An expansion tree E is called an expansion proof of a formula A without strong quantifiers if* $\mathrm{Sh}(E) = A$ *and* $\mathrm{Dp}(E)$ *is a tautology.*

The meaning of expansion proofs is that they encode a proof of validity of the formula they represent. Expansion proofs can be directly translated into sequent calculus or natural deduction proofs, see e.g. [3].

Theorem 1 (Soundness & Completeness). *A formula without strong quantifiers has an expansion proof iff it is valid.*

Proof. In [3]. □

3 Display Expansion Trees

In this section we formally define the data structure of *display expansion tree*, a structure that builds on expansion trees by allowing a flexible *degree* of unfolding. While an expansion tree E induces only the two formulas $\mathrm{Sh}(E)$ and $\mathrm{Dp}(E)$, the display expansion trees based on E turn span the whole spectrum between $\mathrm{Sh}(E)$ and $\mathrm{Dp}(E)$. In our tool, the user will then be able to navigate this spectrum through a comfortable point-and-click interface.

Each quantifier node in a display expansion tree will have one of three states: *open, closed* and *expanded.*

Definition 4 (Display expansion tree). *Display expansion trees are defined as follows:*

- *A is an atomic display expansion tree for an atom A.*
- *If E_1, E_2 are display expansion trees, then so are $\neg E_1$, $E_1 \wedge E_2$, $E_1 \vee E_2$ and $E_1 \rightarrow E_2$.*

– *Let $A(\bar{x})$ be a formula and $\bar{t_1}, \dots, \bar{t_n}$ ($n \geqslant 1$) be a list of (vectors of) terms. Let $E_1, \dots, E_n$ be display expansion trees with* $\mathrm{Sh}(E_i) = A(\bar{t_i})$ *for* $1 \leqslant i \leqslant n$ *and let $Q \in \{\forall, \exists\}$, then:*
  1. *$Q^{\mathrm{c}}\bar{x}A(\bar{x}) +^{\bar{t_1}} E_1 +^{\bar{t_2}} \cdots +^{\bar{t_n}} E_n$ is a display expansion tree (this block of quantifiers is called* closed*).*
  2. *$Q^{\mathrm{o}}\bar{x}A(\bar{x}) +^{\bar{t_1}} E_1 +^{\bar{t_2}} \cdots +^{\bar{t_n}} E_n$ is a display expansion tree (this block of quantifiers is called* open*).*
  3. *$Q^{\mathrm{e}}\bar{x}A(\bar{x}) +^{\bar{t_1}} E_1 +^{\bar{t_2}} \cdots +^{\bar{t_n}} E_n$ is a display expansion tree (this block of quantifiers is called* expanded*).*

  Under the global side condition: If $Q\bar{x}$ is open or expanded, then all quantifiers between $Q\bar{x}$ and the root must be expanded.

The differences in the status of these quantifier blocks will be apparent once we explain how to show a display expansion tree to a user. To that aim we first define the notion of *display formula.*

Definition 5 (display formula). *Display formulas are defined inductively as follows:*

– *If A is an atom, then A is a display formula.*
– *If A, B are display formulas, then so are $\neg A$, $A \wedge B$, $A \vee B$ and $A \to B$.*
– *If $A(\bar{x})$ is a display formula, then $Q\bar{x}A(\bar{x})$ and $Q\bar{x}\langle\bar{t_1}; \dots; \bar{t_n}\rangle A(\bar{x})$ are display formulas for $Q \in \{\forall, \exists\}$, where $\bar{t_i}$ are vectors of terms (which represent substitution instances for $\bar{x}$).*
– *If $A_1, \dots, A_n$ are display formulas, then $\bigwedge\langle A_1, \dots, A_n\rangle$ and $\bigvee\langle A_1, \dots, A_n\rangle$ are display formulas for the n-ary connectives $\bigwedge$ and $\bigvee$.*

Given a display expansion tree E what we show to a user is the display formula $\mathrm{Dy}(E)$ defined as follows.

Definition 6. Dy *maps a display expansion tree to a display formula:*

$$\begin{aligned}
\mathrm{Dy}(E) &= E \textit{ for atomic } E,\\
\mathrm{Dy}(\neg E) &= \neg\mathrm{Dy}(E),\\
\mathrm{Dy}(E_1 \circ E_2) &= \mathrm{Dy}(E_1) \circ \mathrm{Dy}(E_2) \textit{ for } \circ \in \{\wedge, \vee, \to\},\\
\mathrm{Dy}(Q^{\mathrm{c}}\bar{x}A(\bar{x}) +^{\bar{t_1}} E_1 +^{\bar{t_2}} \cdots +^{\bar{t_n}} E_n) &= Q\bar{x}A(\bar{x}) \textit{ for } Q \in \{\forall, \exists\},\\
\mathrm{Dy}(Q^{\mathrm{o}}\bar{x}A(\bar{x}) +^{\bar{t_1}} E_1 +^{\bar{t_2}} \cdots +^{\bar{t_n}} E_n) &= Q\bar{x}\langle t_1; \dots; t_n\rangle A(\bar{x}),\\
\mathrm{Dy}(\exists^{\mathrm{e}}\bar{x}A(\bar{x}) +^{t_1} E_1 +^{\bar{t_2}} \cdots +^{t_n} E_n) &= \bigvee\langle \mathrm{Dy}(E_1), \dots, \mathrm{Dy}(E_n)\rangle,\\
\mathrm{Dy}(\forall^{\mathrm{e}}\bar{x}A(\bar{x}) +^{t_1} E_1 +^{\bar{t_2}} \cdots +^{t_n} E_n) &= \bigwedge\langle \mathrm{Dy}(E_1), \dots, \mathrm{Dy}(E_n)\rangle
\end{aligned}$$

The user can hence control the formula that he sees by changing the status of quantifier nodes of a display expansion tree. A display expansion tree in a particular state of partial unfolding lies hence between the shallow formula and the deep formula of the underlying expansion tree. This observation can be made precise as follows:

Definition 7. Fm *maps display formulas to formulas:*

$$\begin{aligned}
\mathrm{Fm}(A) &= A \textit{ for a formula } A,\\
\mathrm{Fm}(Q\bar{x}A(\bar{x})) &= Q\bar{x}A(\bar{x}) \textit{ for } Q \in \{\forall, \exists\},\\
\mathrm{Fm}(Q\bar{x}\langle t_1; \dots; t_n\rangle A(\bar{x})) &= Q\bar{x}A(\bar{x}) \textit{ for } Q \in \{\forall, \exists\},\\
\mathrm{Fm}(\bigvee\langle A_1, \dots, A_n\rangle) &= A_1 \vee \dots \vee A_n,\\
\mathrm{Fm}(\bigwedge\langle A_1, \dots, A_n\rangle) &= A_1 \wedge \dots \wedge A_n.
\end{aligned}$$

Proposition 1. *Let E be a display expansion tree. If all quantifiers in E are closed, then* $\mathrm{Fm}(\mathrm{Dy}(E)) = \mathrm{Sh}(E)$*. If all quantifiers in E are expanded, then* $\mathrm{Fm}(\mathrm{Dy}(E)) = \mathrm{Dp}(E)$*.*

Proof. By induction on the structure of E. □

4 Transforming Resolution Proofs to Expansion Trees

In this section we give an algorithm for constructing expansion proofs from resolution refutations. We proceed by first transforming a resolution refutation of the negation of a formula F into a proof of F in the sequent calculus [20]. The second step is to transform the sequent calculus proof into an expansion tree proof. While the second part is done in a similar way to other sources [3], the first part, to the best of knowledge of the authors, was not described earlier in this form.

The proofs we expect as input to our algorithm are resolution refutations of sets of clauses. A *clause* is a disjunction of literals, a literal is an atom or the negation of an atom. We will sometimes write a clause in the format $A_1, \dots, A_n \vdash B_1, \dots, B_m$ for B_i being the positive and A_j the negative literals. This notation facilitates the connection to the sequent calculus. We will denote clauses by uppercase Greek letters and formulas by uppercase Latin letters. Substitutions and *most general unifiers* are defined as usual and will be denoted by lowercase Greek letters. Terms are denoted by lowercase Latin letters. We will also write $s[t]$ in order to emphasize that t is a subterm of s.

The version of the *resolution calculus* presented in Fig. 1 forms, if one drops the `(Instance)` rule, the minimal version required in order to obtain completeness for first-order logic with equality. This makes our algorithm applicable, via elementary translations, to the proofs obtained by most resolution theorem provers in the market. The redundant `(Instance)` rule allows us to take advantage of simpler proof formats generated by some theorem provers, such as Prover9. Let C be a set of clauses, a tree over the rules in Fig. 1 with leaves from C is a refutation of C if:

- the root of the tree is the empty clause □.
- when we apply a binary rule on clauses Γ and Δ, their sets of free variables must be disjoint.

$$\frac{A \vee \Gamma \qquad \neg B \vee \Delta}{\Gamma\sigma \vee \Delta\sigma}\ \texttt{(Resolve)}^{1} \qquad \frac{A \vee B \vee \Gamma}{A\sigma \vee \Gamma\sigma}\ \texttt{(Factor)}^{1}$$

$$\frac{t = s \vee \Gamma \qquad A[r] \vee \Delta}{A[s]\sigma \vee \Gamma\sigma \vee \Delta\sigma}\ \texttt{(Paramod)}^{2} \qquad \frac{\Gamma}{\Gamma\sigma}\ \texttt{(Variant)}^{3}$$

$$\frac{\Gamma}{\Gamma\sigma}\,\texttt{(Instance)} \qquad \frac{}{x = x}\,\texttt{(Reflexivity)}$$

1. σ is a most general unifier of A and B.
2. σ is a most general unifier of t and r.
3. σ is a variable renaming.

Fig. 1. The resolution calculus

4.1 Transforming Resolution Proofs to Sequent Proofs

The calculus we will use is presented in Fig. 2. It extends the classical sequent calculus [20] with additional equality rules. The rules for the connectives $\exists, \vee$ and $\rightarrow$ are analogous to the ones presented. We will denote multisets of formulas by uppercase Latin letters. *sequents* are pairs of multisets of formulas denoted by $\Gamma \vdash \Delta$. The formulas in the upper sequents that do not occur in the lower sequent are called *auxiliary formulas* of the rule, those in the lower sequent are called the *principal formulas*.

In order to be able to relate refutations with proofs, we require the following auxiliary definitions.

$$\frac{}{A \vdash A}\ \texttt{(Ax)} \qquad \frac{A, \Gamma \vdash \Delta}{\Gamma \vdash \Delta, \neg A}\ (\neg : r) \qquad \frac{\Gamma \vdash \Delta, A}{\neg A, \Gamma \vdash \Delta}\ (\neg : l)$$

$$\frac{\Gamma \vdash \Delta, A \qquad \Lambda \vdash \Pi, B}{\Gamma, \Lambda \vdash \Delta, \Pi, A \wedge B}\ (\wedge : r) \qquad \frac{A, \Gamma \vdash \Delta}{A \wedge B, \Gamma \vdash \Delta}\ (\wedge : l_1) \qquad \frac{B, \Gamma \vdash \Delta}{A \wedge B, \Gamma \vdash \Delta}\ (\wedge : l_2)$$

$$\frac{\Gamma \vdash \Delta, A[x]}{\Gamma \vdash \Delta, \forall y.A[y]}\ (\forall : r)^{1} \qquad \frac{A[t], \Gamma \vdash \Delta}{\forall y.A[y], \Gamma \vdash \Delta}\ (\forall : l)^{2} \qquad \frac{\Gamma \vdash \Delta, A, A}{\Gamma \vdash \Delta, A}\ (Contr : r)$$

$$\frac{A, A, \Gamma \vdash \Delta}{A, \Gamma \vdash \Delta}\ (Contr : l) \qquad \frac{\Gamma \vdash \Delta}{\Gamma \vdash \Delta, A}\ (Weak : r) \qquad \frac{\Gamma \vdash \Delta}{A, \Gamma \vdash \Delta}\ (Weak : l)$$

$$\frac{\Gamma \vdash \Delta, A \qquad A, \Lambda \vdash \Pi}{\Gamma, \Lambda \vdash \Delta, \Pi}\ (Cut) \qquad \frac{}{\vdash t = t}\ \texttt{(Reflexivity)}$$

$$\frac{\Gamma \vdash \Delta, t = s \qquad \Lambda \vdash \Pi, A[s]}{\Gamma, \Lambda \vdash \Delta, \Pi, A[t]}\ (Eq : r) \qquad \frac{\Gamma \vdash \Delta, t = s \qquad A[s], \Lambda \vdash \Pi}{A[t], \Gamma, \Lambda \vdash \Delta, \Pi}\ (Eq : l)$$

1. x does not occur free in Γ, Δ or in $\forall y.A[y]$.
2. t does not contain variables bound in A.

Fig. 2. The sequent calculus

Definition 8. *Given two sequents s_1 and s_2 of the forms $\Gamma \vdash \Delta$ and $\Pi \vdash \Lambda$, their product $s_1 \times s_2$ is $\Gamma, \Pi \vdash \Delta, \Lambda$. Given two sets of sequents S_1 and S_2, their product $S_1 \times S_2$ is the set containing all possible products between sequents of each set.*

Definition 9 (Clause normal forms). *Let A, A_1, A_2 denote formulas without weak quantifiers and B, B_1, B_2 denote formulas without strong quantifiers and let P denote atoms. Define the mappings $\mathit{CNF}^+(A)$ and $\mathit{CNF}^-(B)$ by the following mutual induction:*

$$
\begin{array}{ll}
\mathit{CNF}^+(P) = \{\vdash P\} & \mathit{CNF}^+(A_1 \wedge A_2) = \mathit{CNF}^+(A_1) \cup \mathit{CNF}^+(A_2) \\
\mathit{CNF}^-(P) = \{P \vdash\} & \mathit{CNF}^+(A_1 \vee A_2) = \mathit{CNF}^+(A_1) \times \mathit{CNF}^+(A_2) \\
\mathit{CNF}^+(\neg B) = \mathit{CNF}^-(B) & \mathit{CNF}^-(B_1 \wedge B_2) = \mathit{CNF}^-(B_1) \times \mathit{CNF}^-(B_2) \\
\mathit{CNF}^-(\neg A) = \mathit{CNF}^+(A) & \mathit{CNF}^-(B_1 \vee B_2) = \mathit{CNF}^-(B_1) \cup \mathit{CNF}^-(B_2) \\
\mathit{CNF}^+(\forall x.A) = \mathit{CNF}^+(A) & \mathit{CNF}^-(\exists x.B) = \mathit{CNF}^-(B)
\end{array}
$$

The case of $\rightarrow$ is defined by combining the cases of $\vee$ and $\neg$.

The point of the above two transformation is that $\mathtt{CNF}^+(A)$ is logically equivalent to A while $\mathtt{CNF}^-(B)$ is logically equivalent to $\neg B$, this definition hence avoids an explicit transformation to negation-normal form before computing a clause set. This transformation is extended to sequents as follows:

Definition 10 (Clause normal forms of sequents). *Let $A_1, \ldots, A_n \vdash B_1, \ldots, B_m$ be a sequent without strong quantifiers, then $\mathit{CNF}^-(A_1, \ldots, A_n \vdash B_1, \ldots, B_m) = \mathit{CNF}^+(A_1 \wedge \ldots \wedge A_n \wedge \neg B_1 \wedge \ldots \wedge \neg B_m)$.*

While this transformation is exponential in the worst case it has not created problems for our practical applications as the sequents we consider are typically quite close to a conjunctive normal form. In case it does pose a performance-problem this transformation can easily be replaced by the polynomial structural clause form transformation.

Example 1. Let $\Gamma \vdash \Delta = P(a), \forall x.(P(x) \rightarrow Q(x)) \vdash Q(a)$, then

$$\mathtt{CNF}^-(\Gamma \vdash \Delta) = \{\vdash P(a);\ P(x) \vdash Q(x);\ Q(a) \vdash\}$$

The following algorithm generates a sequent proof of $A, \Pi \vdash \Lambda$ when the clause $\Pi \vdash \Lambda$ is in $\mathtt{CNF}^+(A)$.

```
Data: Formula $A$ and a clause $\Pi \vdash \Lambda$ such that $\Pi \vdash \Lambda \in \mathtt{CNF}^+(A)$
begin
    $A$ is an atom $\Longrightarrow A \vdash A$
    $A = \neg B \Longrightarrow$ apply $(\neg : l)$ to $\mathtt{PCNF}^-(\Pi \vdash \Lambda, B)$
    $A = B \wedge C \Longrightarrow$ apply either $(\wedge : l_1)$ to $\mathtt{PCNF}^+(B, \Pi \vdash \Lambda)$ or $(\wedge : l_2)$ to
    $\mathtt{PCNF}^+(C, \Pi \vdash \Lambda)$
    $A = \forall x.A \Longrightarrow$ apply $(\forall : l)$ to $\mathtt{PCNF}^+(A, \Pi \vdash \Lambda)$
    ...
end
```

Algorithm 1. $\mathtt{PCNF}^+(A, \Pi \vdash \Lambda)$

The dual algorithm $\mathtt{PCNF}^-$ for computing a sequent calculus proof in the case of $\Pi \vdash \Lambda \in \mathtt{CNF}^-(A)$ is defined in a similar way and we obtain:

Lemma 1. *Let A be a formula without weak quantifiers and B be a formula without strong quantifiers and $\Pi \vdash \Lambda$ a clause, then:*

- *if $\Pi \vdash \Lambda \in \mathtt{CNF}^+(A)$ then $\mathtt{PCNF}^+(A, \Pi \vdash \Lambda)$ is a sequent proof of $A, \Pi \vdash \Lambda$.*
- *if $\Pi \vdash \Lambda \in \mathtt{CNF}^-(B)$ then $\mathtt{PCNF}^-(\Pi \vdash \Lambda, B)$ is a sequent proof of $\Pi \vdash \Lambda, B$.*

Proof. By a straightforward induction on the structure of A. □

For actual applications it is more useful to generate proofs of sequents, not just of formulas, i.e. we work in a setting where a sequent $\Gamma \vdash \Delta$ takes the role of the formulas A or B above. To that aim the above algorithms are extended in a straightforward way to an algorithm $\mathtt{PCNF}(\Gamma \vdash \Delta, \Pi \vdash \Lambda)$ that generates a sequent calculus proof of $\Gamma, \Pi \vdash \Delta, \Lambda$ if $\Pi \vdash \Lambda \in \mathtt{CNF}^-(\Gamma \vdash \Delta)$.

Lemma 2. *Let $\Gamma \vdash \Delta$ be a sequent without strong quantifiers and $\Pi \vdash \Lambda \in \mathtt{CNF}^-(\Gamma \vdash \Delta)$, then $\mathtt{PCNF}(\Gamma \vdash \Delta, \Pi \vdash \Lambda)$ is a proof of $\Gamma, \Pi \vdash \Delta, \Lambda$.*

Proof. If $\Pi \vdash \Lambda \in \mathtt{CNF}^-(\Gamma \vdash \Delta)$, then either $\Pi \vdash \Lambda \in \mathtt{CNF}^+(A)$ for some $A \in \Gamma$ or $\Pi \vdash \Lambda \in \mathtt{CNF}^-(B)$ for some $B \in \Delta$ and we can use Lemma 1. □

Example 2. Continuing Example 1 we have $\mathtt{PCNF}(\Gamma \vdash \Delta, \vdash P(a)) =$

$$\frac{P(a) \vdash P(a)}{P(a), \forall x.(P(x) \to Q(x)) \vdash Q(a), P(a)}\ (Weak:*)$$

and $\mathtt{PCNF}(\Gamma \vdash \Delta, P(x) \vdash Q(x)) =$

$$\frac{\dfrac{\dfrac{P(x) \vdash P(x) \quad Q(x) \vdash Q(x)}{P(x), P(x) \to Q(x) \vdash Q(x)}\ (\to: l)}{P(x), \forall x.(P(x) \to Q(x)) \vdash Q(x)}\ (\forall : l)}{P(x), P(a), \forall x.(P(x) \to Q(x)) \vdash Q(a), Q(x)}\ (Weak:*)$$

and $\mathtt{PCNF}(\Gamma \vdash \Delta, Q(a) \vdash)) =$

$$\frac{Q(a) \vdash Q(a)}{Q(a), P(a), \forall x.(P(x) \to Q(x)) \vdash Q(a)}\ (Weak:*)$$

The last algorithm in this section combines the sequent calculus proofs obtained by PCNF and a refutation of $\mathtt{CNF}^-(\Gamma \vdash \Delta)$ into a sequent calculus proof of $\Gamma \vdash \Delta$. This algorithm translates a dag-like refutation into a tree-like proof and hence grounds it. The most important step is to replace a resolution inference by an atomic cut on instances of the sequent calculus proofs obtained from the premises of the resolution inference.

```
Data: a refutation $R$ of $\mathtt{CNF}^-(\Gamma \vdash \Delta)$
$R$ match begin
    An initial clause $\Pi \vdash \Lambda \implies \mathtt{PCNF}(\Gamma \vdash \Delta, \Pi \vdash \Lambda)$
    $R$ is obtained by (Factor) with m.g.u. $\sigma$ from $R'$ $\implies$ apply $(Contr : r)$ or
    $(Contr : l)$ to $\mathtt{LK}(R')\sigma$
    $R$ is obtained by (Resolve) with m.g.u. $\sigma$ from $R_1$ and $R_2$ $\implies$ apply
    $(Cut)$ to $\mathtt{LK}(R_1)\sigma$ and $\mathtt{LK}(R_2)\sigma$
    ...
end
```

Algorithm 2. LK

Theorem 2. *Let $\Gamma \vdash \Delta$ be a sequent without strong quantifiers and let R be a refutation of $\mathtt{CNF}^{-}(\Gamma \vdash \Delta)$, then $\mathtt{LK}(R)$ is a sequent calculus proof of $\Gamma \vdash \Delta$.*

Proof. By a straightforward induction on the structure of the refutation. □

4.2 Transforming Sequent Proofs to Expansion Trees

In this section we describe how to read off expansion trees from sequent calculus proofs. The algorithm presented in this section is based on [3] but in addition deals with quantifier-free cuts and equation rules. The algorithm `merge` used in this algorithm for the merging of two expansion trees is defined in [3].

```
Data: A sequent proof $P$
if (Ax) then
    return an atomic expansion tree for each formula
else
    return expansion trees of upper sequents and replace the trees $E_i$ of the
    auxiliary formulas with $P$ match begin
        $(\wedge : r) \Longrightarrow E_1 \wedge E_2$
        $(\forall : l)$ with principal formula $\forall x.A$ and auxiliary formula
        $A[t/x] \Longrightarrow \forall x.A +^t E$
        $(Contr : l) \Longrightarrow \mathtt{merge}(E_1, E_2)$
        $(Cut) \Longrightarrow \varnothing$
        $(Eq : r) \Longrightarrow E_2$
        ...
    end
end
```

Algorithm 3. `ET`

Theorem 3. *Let P be a sequent proof of s without strong quantifiers, then $\mathtt{ET}(P)$ is a sequent of expansion trees of the formulas in s.*

Proof. By a straightforward induction on the structure of P. □

4.3 Complexity and Scalability

Expansion trees are an inherently ground formalism. This has the consequence that the translation from a (non-ground) resolution refutation to an expansion tree is expontential in the worst case. On the one hand this is a limitation of the algorithms presented here. On the other hand, grounding has a signifcant benefit: the witnesses which typically carry important information can only be read off from a ground proof. We will illustrate this phenomenon in the next section by describing an example for a clause set whose refutation only shows that a certain puzzle can be solved while its expansion tree contains the solution (which necessarily must be a ground term).

An extension of our algorithms which would be useful for such critical cases would be to carry out the computation of ground instances *on demand.* The present user-interface would not change but the implementation of a display expansion tree would: instead of keeping a complete expansion tree in memory, it would only store the original resolution refutation and compute the ground instances of single quantifiers when asked to.

5 Implementation and Examples

Programmed in Scala, GAPT is a framework intended, on the one hand, to allow an easy and intuitive programming of proof theoretical algorithms and applications and on the other hand to be as general and flexible as possible, in order to be able to target the widest range of languages and calculi. To meet these two requirements, we make extensive use of Scala's object-oriented (OO) and functional paradigms. The OO support is used mainly in order to build a complex type system and abstraction between different logics. The functional support is used, as we will see in the rest of this paper, in order to map formulas, proofs and similar data directly to Scala functions and algebraic data structures. The fact that Scala is compatible with Java allowed us to use its built-in libraries in order to supply a comprehensive graphical user interface. GAPT supplies algebraic data structures for terms, formulas, sequents, resolution proofs, sequent proofs and many other logical objects. The functionality described in this paper is available from version 1.4 on. The interested reader is invited to download the current version from `http://www.logic.at/gapt`.

5.1 Import of a Resolution Proof

The GAPT-System contains two methods for importing resolution proofs. The first is based on proof replaying by reproving each inference through forward reasoning [21] in the minimal resolution calculus implemented in GAPT (Fig. 1). A drawback of this method is that the resulting proofs may differ significantly from the ones found by the theorem prover and that search might be inefficient for macro-rules like hyperresolution. Therefore the second method implements a direct import from the format for the Ivy proof checker [22]. Ivy's resolution calculus (Fig. 3) replaces unification by an explicit instance rule which applies the substitution separately. The next step in the extraction of an expansion tree - the transformation to LK - grounds the proof, applying the substitution to the respective subproofs anyway. Therefore we decided to add the instance rule to GAPT's resolution calculus instead of merging instance rules into unifiers within the other inference rules. The flip rule is expanded to a proof of equational symmetry from the equational reflexivity axiom.

The conversion of Prover9's output [23] to the Ivy format is performed by prooftrans which is part of Prover9's LADR distribution. An Ivy proof is represented as a Lisp S-Expression which requires a different naming convention, therefore GAPT's parser has to integrate proper renaming of constants and variables according to these conventions.

We will use a running example to illustrate the work-flow of our tool. The running example is the famous puzzle of a farmer who wants to transport a wolf, a goat and a cabbage across a river using a boat in which he can take at most one of these items with him. The difficulty is that he can neither leave goat and cabbage nor wolf and goat alone on one of the shores. How can he cross the river? This is formalized as problem `PUZ047+1` of the TPTP-library [24]. The formalization uses a 5-ary predicate symbol p whose first four coordinates contain the

$$\frac{A \vee \Gamma \qquad \neg A \vee \Delta}{\Gamma \vee \Delta}\,\texttt{(Resolve)} \qquad \frac{A \vee A \vee \Gamma}{A \vee \Gamma}\,\texttt{(Factor)}$$

$$\frac{t = s \vee \Gamma \qquad A[t] \vee \Delta}{A[s] \vee \Gamma \vee \Delta}\,\texttt{(Paramod)} \qquad \frac{\Gamma}{\Gamma\sigma}\,\texttt{(Instance)}^{1}$$

$$\frac{\Gamma, s = t}{\Gamma, t = s}\,\texttt{(Flip)} \qquad \frac{}{x = x}\,\texttt{(Reflexivity)}$$

1. σ is an arbitrary substitution.

Fig. 3. The Ivy Resolution calculus

positions of farmer, wolf, goat and cabbage and whose fifth position contains the actions already taken. Universally quantified implications describe the possible actions, the additional axiom p(south, south, south, south, start) describes the initial state, the goal is to prove $\exists z\,$p(north, north, north, north, z). For testing our running example we first produce a Prover9 output file by running:

```
$ tptp_to_ladr < PUZ047+1.p > PUZ047+1.in
$ prover9 < PUZ047+1.in > PUZ047+1.out
```

Then we start the command-line interface of GAPT and load the resolution proof from the output file.

```
$ ./cli.sh
scala> val p = loadProver9Proof( "PUZ047+1.out" )
```

This command imports a resolution refutation `p`. One important aspect of this refutation is that **it does not contain the solution to the puzzle**, it merely shows that the puzzle is solvable. The actual solution will be computed by our tool automatically by the transformation of the resolution refutation to an expansion tree. The solution will then be presented in plain sight to the user as instance of the above-mentioned existential quantifier. This example thus illustrates very well the added value of expansion trees over resolution refutations.

5.2 Extraction of an Expansion Tree

As described in Section 4, the extraction of an expansion tree from a resolution refutation proceeds in two phases. First we import a resolution refutation and transform it into a sequent calculus proof following Algorithms 1 and 2.

In addition to the resolution proof, the original input formula is extracted from Prover9's output file. The rationale behind this is that the user expects to see an expansion tree representing the input formula, its clause normal form might be of a significantly different shape. The original formula is transformed into a sequent which forms the end-sequent of the sequent calculus proof that is constructed. This can be carried out by

```
scala> val q = loadProver9LKProof( "PUZ047+1.out" )
```

which creates a sequent calculus proof `q` from the refutation in `PUZ047+1.out`. The expansion trees can then be read off from this sequent calculus proof by:

```
scala> val E = extractExpansionTrees( q )
```

5.3 The Graphical User Interface

ProofTool is the *Graphical User Interface* of the GAPT system [25]. It can be used in two ways: as a pure visualization tool (with the features like zooming, scrolling, searching, etc.) and as a proof manipulator (allowing to call GAPT's proof transformations such as cut-elimination, regularization, skolemization, etc.). The objects ProofTool can render are formulas, sequents, proofs, trees, sequent- and definition-lists. Sequents consisting of expansion trees are handled in a special way to support the interactive visualization features specific to expansion trees. A sequent of expansion trees is displayed in a two column split pane, where one column is for the antecedent and the other is for the consequent of the sequent.

Expansion trees are displayed in ProofTool in the following way: For each expansion tree a display expansion tree is produced by adding the state "closed" to the quantifier nodes occurring in the expansion tree. Then the display formula of the display expansion tree is rendered on the screen. Finally, the display formula can be manipulated by changing the state of quantifiers. A single left-click changes the state from closed to open, from open to expanded, and from expanded to closed. Additionally a context-menu is opened on a right-click to allow a direct state-change.

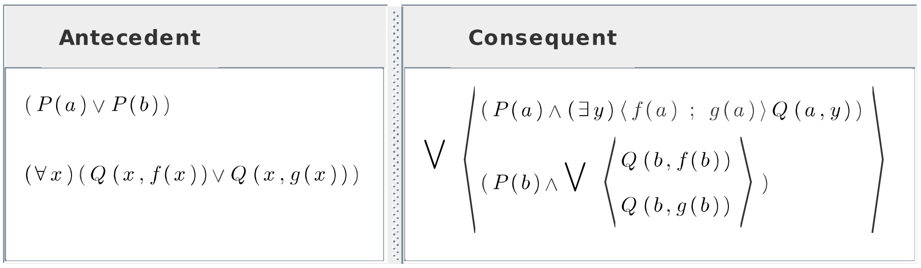
The expansion trees of our running example can be displayed by

```
scala> prooftool( E )
```

which allows the solution to the puzzle to be read off with a single click:

take_goat(go_alone(take_wolf(take_goat(take_cabbage(go_alone(take_goat(start)))))))

In order to illustrate the display of an expansion tree with nested quantifiers we include a screen-shot of a simple example in Figure 4.



Antecedent	Consequent
$(P(a) \vee P(b))$	$\bigvee \left\langle \begin{matrix} (P(a) \wedge (\exists y)\langle f(a)\ ;\ g(a)\rangle Q(a,y)) \\ (P(b) \wedge \bigvee \left\langle \begin{matrix} Q(b,f(b)) \\ Q(b,g(b)) \end{matrix} \right\rangle) \end{matrix} \right\rangle$
$(\forall x)(Q(x,f(x)) \vee Q(x,g(x)))$	

Fig. 4. An expansion of the sequent $P(a) \vee P(b), \forall x(Q(x,f(x)) \vee Q(x,g(x))) \vdash \exists x(P(x) \wedge \exists y Q(x,y))$

6 Conclusion

We have described an approach to understanding resolution proofs through Herbrand's theorem and a tool based on this approach. We have illustrated its usefulness on two examples. The computation of ground instances of quantifiers in combination with a flexible display of an expansion tree in a graphical user interface allows a very quick access to the crucial turning points of a computer-generated proof.

There are several important lines for future work: definitions (i.e. abbreviations of formulas by new predicate symbols) are crucial for human-readable formalizations of mathematical proofs. They can be integrated into this approach in a straightforward way by allowing to fold and unfold them too. Furthermore, just as the original notions of expansion trees [3], our approach and implementation supports higher-order logic as well. The only part of the programs not supporting higher-order logic is the transformation from refutations to sequent proofs, which is customized to the first-order resolution calculus. Another highly interesting extension is to carry out the computation of ground instances on demand as described in Section 4.3. This can be continued much beyond the scope of resolution proofs. For example, by relying on the relationship between cut-elimination and tree grammars established in [26] it would even be possible to do cut-elimination on demand by computing only the instances of a certain quantifier from a proof with cuts.

References

1. Herbrand, J.: Recherches sur la théorie de la démonstration. PhD thesis, Université de Paris (1930)
2. Buss, S.R.: On Herbrand's Theorem. In: Leivant, D. (ed.) LCC 1994. LNCS, vol. 960, pp. 195–209. Springer, Heidelberg (1995)
3. Miller, D.: A Compact Representation of Proofs. Studia Logica 46(4), 347–370 (1987)
4. Baaz, M., Leitsch, A.: Cut-elimination and Redundancy-elimination by Resolution. Journal of Symbolic Computation 29(2), 149–176 (2000)
5. Luckhardt, H.: Herbrand-Analysen zweier Beweise des Satzes von Roth: Polynomiale Anzahlschranken. Journal of Symbolic Logic 54(1), 234–263 (1989)
6. Bombieri, E., van der Poorten, A.: Some quantitative results related to Roth's theorem. Journal of the Australian Mathematical Society 45(2), 233–248 (1988)
7. Hetzl, S., Leitsch, A., Weller, D., Woltzenlogel Paleo, B.: Herbrand Sequent Extraction. In: Autexier, S., Campbell, J., Rubio, J., Sorge, V., Suzuki, M., Wiedijk, F. (eds.) AISC/Calculemus/MKM 2008. LNCS (LNAI), vol. 5144, pp. 462–477. Springer, Heidelberg (2008)
8. Baaz, M., Hetzl, S., Leitsch, A., Richter, C., Spohr, H.: CERES: An Analysis of Fürstenberg's Proof of the Infinity of Primes. Theoretical Computer Science 403(2-3), 160–175 (2008)
9. Baaz, M., Hetzl, S., Leitsch, A., Richter, C., Spohr, H.: Cut-Elimination: Experiments with CERES. In: Baader, F., Voronkov, A. (eds.) LPAR 2004. LNCS (LNAI), vol. 3452, pp. 481–495. Springer, Heidelberg (2005)

10. Urban, C.: Classical Logic and Computation. PhD thesis, University of Cambridge (October 2000)
11. Hetzl, S., Leitsch, A., Weller, D.: Towards Algorithmic Cut-Introduction. In: Bjørner, N., Voronkov, A. (eds.) LPAR-18. LNCS, vol. 7180, pp. 228–242. Springer, Heidelberg (2012)
12. Hetzl, S.: Project Presentation: Algorithmic Structuring and Compression of Proofs (ASCOP). In: Jeuring, J., Campbell, J.A., Carette, J., Dos Reis, G., Sojka, P., Wenzel, M., Sorge, V. (eds.) CICM 2012. LNCS, vol. 7362, pp. 438–442. Springer, Heidelberg (2012)
13. Horacek, H.: Presenting Proofs in a Human-Oriented Way. In: Ganzinger, H. (ed.) CADE 1999. LNCS (LNAI), vol. 1632, pp. 142–156. Springer, Heidelberg (1999)
14. Meier, A.: System Description: TRAMP: Transformation of Machine-Found Proofs into ND-Proofs at the Assertion Level. In: McAllester, D. (ed.) CADE 2000. LNCS, vol. 1831, pp. 460–464. Springer, Heidelberg (2000)
15. Trac, S., Puzis, Y., Sutcliffe, G.: An interactive derivation viewer. Electronic Notes in Theoretical Computer Science 174(2), 109–123 (2007)
16. Denzinger, J., Schulz, S.: Recording, Analyzing and Presenting Distributed Deduction Processes. In: Hong, H. (ed.) 1st International Symposium on Parallel Symbolic Computation (PASCO). Lecture Notes Series in Computing, vol. 5, pp. 114–123. World Scientific Publishing (1994)
17. Pfenning, F.: Analytic and non-analytic proofs. In: Shostak, R.E. (ed.) CADE 1984. LNCS, vol. 170, pp. 394–413. Springer, Heidelberg (1984)
18. Pfenning, F.: Proof Transformations in Higher-Order Logic. PhD thesis, Carnegie Mellon University (1987)
19. Miller, D.: Proofs in Higher-Order Logic. PhD thesis, Carnegie-Mellon University (1983)
20. Gentzen, G.: Untersuchungen über das logische Schließen I. Mathematische Zeitschrift 39(2), 176–210 (1934)
21. Dunchev, C., Leitsch, A., Libal, T., Riener, M., Rukhaia, M., Weller, D., Woltzenlogel-Paleo, B.: System Feature Description: Importing Refutations into the GAPT Framework. In: Proof Exchange for Theorem Proving Second International Workshop, PxTP (2012)
22. Mccune, W., Shumsky, O.: Ivy: A Preprocessor And Proof Checker For First-Order Logic. In: Computer-Aided Reasoning: ACL2 Case Studies. Kluwer Academic Publishers (2000)
23. McCune, W.: Prover9 and mace4 manual - output files (2005-2010), `https://www.cs.unm.edu/~mccune/mace4/manual/2009-11A/output.html`
24. Sutcliffe, G.: The TPTP Problem Library and Associated Infrastructure: The FOF and CNF Parts, v3.5.0. Journal of Automated Reasoning 43(4), 337–362 (2009)
25. Dunchev, C., Leitsch, A., Libal, T., Riener, M., Rukhaia, M., Weller, D., Woltzenlogel-Paleo, B.: ProofTool: GUI for the GAPT Framework (to appear)
26. Hetzl, S.: Applying tree languages in proof theory. In: Dediu, A.-H., Martín-Vide, C. (eds.) LATA 2012. LNCS, vol. 7183, pp. 301–312. Springer, Heidelberg (2012)

A Labelled Sequent Calculus for BBI: Proof Theory and Proof Search

Zhé Hóu, Alwen Tiu, and Rajeev Goré

Logic and Computation Group, Research School of Computer Science
The Australian National University, Canberra, ACT 0200, Australia

Abstract. We present a labelled sequent calculus for Boolean BI (BBI), a classical variant of the logic of Bunched Implication. The calculus is simple, sound, complete, and enjoys cut-elimination. We show that all the structural rules in the calculus, i.e., those rules that manipulate labels and ternary relations, can be localised around applications of certain logical rules, thereby localising the handling of these rules in proof search. Based on this, we demonstrate a free variable calculus that deals with the structural rules lazily in a constraint system. We propose a heuristic method to quickly solve certain constraints, and show some experimental results to confirm that our approach is feasible for proof search. Additionally, we conjecture that different semantics for BBI and some axioms in concrete models can be captured by adding extra structural rules.

1 Introduction

The logic of bunched implications (BI) was introduced to reason about resources using additive connectives $\wedge$, $\vee$, $\rightarrow$, $\top$, $\bot$, and multiplicative connectives $\top^*$, $*$, $-\!*$ [14]. Both parts are intuitionistic so BI is also Intuitionistic logic (IL) plus Lambek multiplicative logic (LM). Changing the additive part to classical logic gives Boolean BI (BBI). BI logics are closely related to separation logic [17], a logic for proving properties of programs. Thus, the semantics and proof theory of BI-logics, particularly for proof search, are important in computer science.

The ternary relational Kripke semantics of BBI-logics come in at least three different flavours: non-deterministic (ND), partial deterministic (PD), and total deterministic (TD) [10]. These semantics give different logics w.r.t. validity, i.e., $BBI_{ND}, BBI_{PD}, BBI_{TD}$ respectively, and all are undecidable [3,10]. The purely syntactic proof theory of BBI also comes in three flavours: Hilbert calculi [16,5], display calculi [1] and nested sequent calculi [15]. All are sound and complete w.r.t. the ND-semantics [5,1,15].

In between the relational semantics and the purely syntactic proof theory are the labelled tableaux of Larchey-Wendling and Galmiche which are sound and complete w.r.t. the PD-semantics [9,8]. They remark that "the adaptation of this tableaux system to BBI_{TD} should be straightforward (contrary to BBI_{ND})" [11]. We return to these issues in Section 7.

The structural rules of display calculi, especially the contraction rule on structures, are impractical for backward proof search. Nested sequents also face similar problems, and although Park et al. showed the admissibility of contraction in an improved nested sequent calculus, it contains other rules that explicitly contract structures. Their iterative deepening automated theorem prover for BBI based on nested sequents is terminating and incomplete for bounded depths, but complete and potentially non-terminating for an unbounded depth [15]. The labelled tableaux of Larchey-Wendling and Galmiche compile all structural rules into PD-monoidal constraints, and are cut-free complete for BBI_{PD} using a potentially infinite counter-model construction [8]. But effective proof search is only a "perspective" and is left as further work [8, page 2].

Surprisingly, many applications of BBI do not directly correspond to its widely used non-deterministic semantics. For example, separation logic models are instances of partial deterministic models [10] while "memory models" for BBI are restricted to have *indivisible units*: "the empty memory cannot be split into non-empty pieces" [3]. Our goal is to give a labelled proof system for BBI based upon the ND-semantics which easily extend to the PD- and TD-semantics, and also these other, more "practical", semantics.

Our labelled sequent calculus LS_{BBI} for BBI adopts some features from existing labelled tableaux for BBI [9] and existing labelled sequent calculi for modal logics [12]. Unlike these calculi, some LS_{BBI}-rules contain substitutions on labels. From a proof-search perspective, labelled calculi are no better than display calculi since they require extra-logical rules to explicitly encode the frame conditions of the underlying (Kripke) semantics. Such rules, which we refer to simply as structural rules, are just as bad as display postulates for proof search since we may be forced to explore all potential models. As a step towards our goal, we show that the applications of these structural rules can be localised around logical rules. Thus these structural rules are only triggered by applications of logical rules, leading to a purely syntax-driven proof search procedure for LS_{BBI}.

Our work is novel from two perspectives. Compared to the labelled tableaux of Larchey-Wendling and Galmiche, we deal with the non-deterministic semantics of BBI, which they have flagged as a difficulty, and obtain a constructive cut-elimination procedure. Compared to the nested sequent calculus of Park et al., our calculus is much simpler, and generally gives much shorter proofs. Note that Park et al. actually gave a labelled variant of their nested sequent calculus, with the same logical rules as ours. However, their structural rules are still just notational variants of the original ones, which are lengthy and do not use ternary relations. We also give some structural rules which we conjecture will give cut-free labelled calculi for all the other semantics mentioned above.

Detailed proofs of all claims except this conjecture are available in [7].

2 Syntax and Semantics of BBI

Let Var be a set of propositional variables. BBI formulae are defined inductively as follows, where $P \in Var$, $\top^*, *, -\!\!*$ are the multiplicative unit, conjunction, and implication respectively:

$$A ::= P \mid \top \mid \bot \mid \neg A \mid A \vee A \mid A \wedge A \mid A \rightarrow A \mid \top^* \mid A * A \mid A -\!\!* A$$

The Kripke semantics of BBI employs a ternary relation of worlds based on a non-deterministic monoid structure, á la Galmiche and Larchey-Wendling [5]. A relational frame is a triple $(\mathcal{M}, \triangleright, \epsilon)$, where $\triangleright \subseteq \mathcal{M} \times \mathcal{M} \times \mathcal{M}$. Following [5], we write $a, b \triangleright c$ instead of $\triangleright(a, b, c)$ and also adopt a single unit ϵ, rather than a set of units [4]. We therefore have the following conditions for all $a, b, c, d \in \mathcal{M}$:

Identity	$\epsilon, a \triangleright b$ iff $a = b$
Commutativity	$a, b \triangleright c$ iff $b, a \triangleright c$
Associativity	$\exists k, (a, k \triangleright d) \& (b, c \triangleright k) \Rightarrow \exists l, (a, b \triangleright l) \& (l, c \triangleright d)$.

Intuitively, the relation $x, y \triangleright z$ means that z can be partitioned into two parts x and y. The identity condition can be read as every world can be partitioned into an empty world and itself. Commutativity captures that partitioning z into x and y is the same as partitioning z into y and x. Finally, associativity means that if z can be partitioned into x and y, and x can further be partitioned into u and v, then all together z consists of u, v and y. Therefore there must exist an element w which is the combination of v and y, such that w and u form z. We do not restrict this monoid to be cancellative, so $x, y \triangleright x$ does not imply $y = \epsilon$.

Let $(\mathcal{M}, \triangleright, \epsilon)$ be a relational frame and $v : Var \to \mathcal{P}(\mathcal{M})$ be a *valuation*. A *forcing relation* "$\Vdash$" between $m \in \mathcal{M}$ and BBI-formulae is defined as follows [5]:

$m \Vdash \top^*$ iff $m = \epsilon$ $m \Vdash P$ iff $P \in Var$ and $m \in v(P)$

$m \Vdash \bot$ iff never $m \Vdash A \lor B$ iff $m \Vdash A$ or $m \Vdash B$

$m \Vdash \top$ iff always $m \Vdash A \land B$ iff $m \Vdash A$ and $m \Vdash B$

$m \Vdash \neg A$ iff $m \nVdash A$ $m \Vdash A \to B$ iff $m \nVdash A$ or $m \Vdash B$

$m \Vdash A * B$ iff $\exists a, b.(a, b \triangleright m$ and $a \Vdash A$ and $b \Vdash B)$

$m \Vdash A {-\!\!*}\, B$ iff $\forall a, b.((m, a \triangleright b$ and $a \Vdash A)$ implies $b \Vdash B)$

A formula A is true at $m \in \mathcal{M}$ if $m \Vdash A$ and is *valid* if $m \Vdash A$ for every $m \in \mathcal{M}$ in every model $((\mathcal{M}, \triangleright, \epsilon), v)$.

3 The Labelled Sequent Calculus for BBI

The inference rules of LS_{BBI} are shown in Figure 1, where p is an atomic formula, A, B are formulae, w, x, y, z are in the set $LVar$ of label variables, and ϵ is the label constant. We define a mapping $\rho : \{\epsilon\} \cup LVar \to \mathcal{M}$ from labels to worlds. We overload the notation in an obvious way so that ϵ is the empty world in the semantics and the label constant, while $\triangleright$ is the ternary relation in the semantics and in the calculus. Therefore, we require that $\forall \rho.\rho(\epsilon) = \epsilon$.

A sequent $\Gamma \vdash \Delta$ consists of a semi-colon separated multiset Γ of relational atoms and labelled formulae and a semi-colon separated multiset Δ of labelled formulae. Note that relational atoms can appear only in the left-hand-side Γ.

A labelled formula $w : A$ means formula A is true in world $\rho(w)$. A relational atom $(x, y \triangleright z)$, which we always write inside parentheses, is interpreted as $\rho(x), \rho(y) \triangleright \rho(z)$ in the semantics. That is, a labelled formula $w : A$ is true iff $\rho(w) \Vdash A$, and a relational atom $(x, y \triangleright z)$ is true iff $\rho(x), \rho(y) \triangleright \rho(z)$ holds.

Definition 1 (Sequent Validity). *A sequent $\Gamma \vdash \Delta$ in LS_{BBI} is valid if for all $(\mathcal{M}, \triangleright, \epsilon)$, v and ρ: if every $w : A \in \Gamma$ and every $(x, y \triangleright z) \in \Gamma$ are true then so is some $w' : B \in \Delta$.*

BBI-validity of a formula A corresponds to the sequent validity of $\vdash x : A$ where x is an arbitrary label. This notion of validity is common for BBI [10,15] and CBI [2], but is stronger than BI-validity [16], where A is only required to be true at the world ϵ in all BI-models. Using labelled sequents, BI-validity informally corresponds to the sequent validity of $\vdash \epsilon : A$. For example, the formula $\top^*$ is BI-valid, but it is not BBI-valid.

In our sequents, the structural connective ";" means additive "and" in the antecedent and means additive "or" in the succedent. Traditional sequents use "," in this role, but our notation is consistent with sequent calculi for the family of Bunched Implication (BI) logics, where ";" is the additive structural connective and "," is the multiplicative structural connective. The "," connective does not appear explicitly in our sequents but is encoded implicitly in the relational atoms.

In each rule, the formula/relational atom shown explicitly in the conclusion is the *principal formula/relational atom.* In the cut rule, the cut-formula is $x : A$.

The semantics of $*$ involves an existential condition, so rules $*L$ and $*R$ incorporate existential and universal quantifiers respectively, conversely for the rules $-\!\!* \ L$ and $-\!\!* \ R$. Therefore, rules $*L$ and $-\!\!* \ R$ create a premise containing new relations, and the labels in the created relation must be fresh (except for the label of the principal formula). Rules $*R$ and $-\!\!* \ L$ create premises using existing relations from the conclusion. Further, in rules A and A_C, the label w must be fresh in the premise, as it represents a new partition of the original world. Contraction admissibility [7] requires the rule A_C, a special case of A with a built-in contraction on $(x, y \triangleright x)$. The rule $\top^* L$ utilises a substitution $[\epsilon/x]$ where $\Gamma[y/x]$ is the result of replacing every occurrence of x in Γ by y.

The *additive rules* ($\bot L$, $\top R$, $\land L$, $\land R$, $\rightarrow L$, $\rightarrow R$) and the *multiplicative rules* ($\top^* L$, $\top^* R$, $*L$, $*R$, $-\!\!* \ L$, $-\!\!* \ R$) respectively deal with the additive/ multiplicative connectives. The *zero-premise rules* are those with no premise (*id*, $\bot L$, $\top R$, $\top^* R$). Figure 2 shows an example derivation in LS_{BBI}.

Note that we start (at the bottom) by labelling the formula with an arbitrary world a. Since provability is preserved by substitutions of labels (Lemma 1), provability of $\vdash a : F$ implies provability of $\vdash w : F$, for any world w. Thus, if a formula is provable, then it is true in every world in every model.

3.1 Soundness and Completeness

Definition 2 (Sequent Falsifiability). *A sequent $\Gamma \vdash \Delta$ is falsifiable if some $(\mathcal{M}, \triangleright, \epsilon)$, v and ρ make every member of Γ true and every member of Δ false.*

Theorem 1 (Soundness). *The labelled sequent calculus LS_{BBI} is sound w.r.t. the non-deterministic monoidal Kripke semantics for BBI.*

For each rule in LS_{BBI}, we show that if the conclusion is falsifiable, then the premise is falsifiable. We prove the completeness of LS_{BBI} by showing that

Identity and Cut:

$$\frac{}{\Gamma; w : P \vdash w : P; \Delta}\ id \qquad \frac{\Gamma \vdash x : A; \Delta \qquad \Gamma'; x : A \vdash \Delta'}{\Gamma; \Gamma' \vdash \Delta; \Delta'}\ cut$$

Logical Rules:

$$\frac{}{\Gamma; w : \bot \vdash \Delta}\ \bot L \qquad \frac{\Gamma[\epsilon/w] \vdash \Delta[\epsilon/w]}{\Gamma; w : \top^* \vdash \Delta}\ \top^* L$$

$$\frac{}{\Gamma \vdash w : \top; \Delta}\ \top R \qquad \frac{}{\Gamma \vdash \epsilon : \top^*; \Delta}\ \top^* R$$

$$\frac{\Gamma; w : A; w : B \vdash \Delta}{\Gamma; w : A \wedge B \vdash \Delta}\ \wedge L \qquad \frac{\Gamma \vdash w : A; \Delta \qquad \Gamma \vdash w : B; \Delta}{\Gamma \vdash w : A \wedge B; \Delta}\ \wedge R$$

$$\frac{\Gamma \vdash w : A; \Delta \qquad \Gamma; w : B \vdash \Delta}{\Gamma; w : A \to B \vdash \Delta}\ \to L \qquad \frac{\Gamma; w : A \vdash w : B; \Delta}{\Gamma \vdash w : A \to B; \Delta}\ \to R$$

$$\frac{(x, y \triangleright z); \Gamma; x : A; y : B \vdash \Delta}{\Gamma; z : A * B \vdash \Delta}\ *L \qquad \frac{(x, z \triangleright y); \Gamma; x : A \vdash y : B; \Delta}{\Gamma \vdash z : A \mathrel{-\!\!*} B; \Delta}\ \mathrel{-\!\!*} R$$

$$\frac{(x, y \triangleright z); \Gamma \vdash x : A; z : A * B; \Delta \qquad (x, y \triangleright z); \Gamma \vdash y : B; z : A * B; \Delta}{(x, y \triangleright z); \Gamma \vdash z : A * B; \Delta}\ *R$$

$$\frac{(x, y \triangleright z); \Gamma; y : A \mathrel{-\!\!*} B \vdash x : A; \Delta \qquad (x, y \triangleright z); \Gamma; y : A \mathrel{-\!\!*} B; z : B \vdash \Delta}{(x, y \triangleright z); \Gamma; y : A \mathrel{-\!\!*} B \vdash \Delta}\ \mathrel{-\!\!*} L$$

Structural Rules:

$$\frac{(y, x \triangleright z); (x, y \triangleright z); \Gamma \vdash \Delta}{(x, y \triangleright z); \Gamma \vdash \Delta}\ E \qquad \frac{(u, w \triangleright z); (y, v \triangleright w); (x, y \triangleright z); (u, v \triangleright x); \Gamma \vdash \Delta}{(x, y \triangleright z); (u, v \triangleright x); \Gamma \vdash \Delta}\ A$$

$$\frac{(x, \epsilon \triangleright x); \Gamma \vdash \Delta}{\Gamma \vdash \Delta}\ U \qquad \frac{(x, w \triangleright x); (y, y \triangleright w); (x, y \triangleright x); \Gamma \vdash \Delta}{(x, y \triangleright x); \Gamma \vdash \Delta}\ A_C$$

$$\frac{(\epsilon, w' \triangleright w'); \Gamma[w'/w] \vdash \Delta[w'/w]}{(\epsilon, w \triangleright w'); \Gamma \vdash \Delta}\ Eq_1 \qquad \frac{(\epsilon, w' \triangleright w'); \Gamma[w'/w] \vdash \Delta[w'/w]}{(\epsilon, w' \triangleright w); \Gamma \vdash \Delta}\ Eq_2$$

Side conditions:

$w \neq \epsilon$ in $\top^* L$, Eq_1 and Eq_2
the labels x and y do not occur in the conclusion in $*L$ and $\mathrel{-\!\!*} R$
the label w does not occur in the conclusion in A and A_C

Fig. 1. The (cut-free) labelled sequent calculus LS_{BBI} for Boolean BI

every derivation of a formula in the Hilbert system for BBI [5] can be mimicked in LS_{BBI}, possibly using cuts. Detailed proofs are available in our arXiv paper [7].

Theorem 2 (Completeness). *The labelled sequent calculus LS_{BBI} is complete w.r.t. the non-deterministic monoidal Kripke semantics for BBI.*

$$\begin{prooftree}
\AxiomC{}
\RightLabel{$\top^* R$}
\UnaryInfC{$(\epsilon, a \triangleright a); (a, \epsilon \triangleright a); a : A \vdash \epsilon : \top^*$}
\AxiomC{}
\RightLabel{id}
\UnaryInfC{$(\epsilon, a \triangleright a); (a, \epsilon \triangleright a); a : A \vdash a : A$}
\RightLabel{$*R$}
\BinaryInfC{$(\epsilon, a \triangleright a); (a, \epsilon \triangleright a); a : A \vdash a : \top^* * A$}
\RightLabel{E}
\UnaryInfC{$(a, \epsilon \triangleright a); a : A \vdash a : \top^* * A$}
\RightLabel{U}
\UnaryInfC{$a : A \vdash a : \top^* * A$}
\RightLabel{$\to R$}
\UnaryInfC{$\vdash a : A \to (\top^* * A)$}
\end{prooftree}$$

Fig. 2. An example derivation in LS_{BBI}

3.2 Cut Elimination

We now state the cut-elimination theorem for our labelled sequent calculus. The general proof outlined here is similar to the cut-elimination proof for labelled systems for modal logic [12], i.e., we start by proving a substitution lemma for labels, followed by proving the invertibility of inference rules, weakening admissibility, and contraction admissibility, before proceeding to the main cut-elimination proof. As there are many case analyses in these proofs, we only outline the important parts here. More details are available in our arXiv paper [7].

Given a derivation Π, its *height* $ht(\Pi)$ is defined as the length of the longest branch in the derivation tree of Π. The substitution lemma shows that provability is preserved under arbitrary substitutions of labels.

Lemma 1 (Substitution). *If Π is an LS_{BBI} derivation for the sequent $\Gamma \vdash \Delta$ then there is an LS_{BBI} derivation Π' of the sequent $\Gamma[y/x] \vdash \Delta[y/x]$ where every occurrence of label x ($x \neq \epsilon$) is replaced by label y, such that $ht(\Pi') \leq ht(\Pi)$.*

The admissibility of weakening and contraction on both formulae and relational atoms follows unsurprisingly from our design of the calculus, as does the invertibility of the inference rules. Note that the rule A_C exists for avoiding contraction on relational atoms when applying the rule A: see [7] for details.

Suppose an application of the *cut* rule has premise derivations Π_1 and Π_2 and a cut-formula $x : A$ of size $|A|$. The *cut height* is $ht(\Pi_1) + ht(\Pi_2)$ and the complexity of such a *cut* rule is $(|A|, ht(\Pi_1) + ht(\Pi_2))$. If there are multiple branches in Π_1, then $ht(\Pi_1)$ shall be the height of the longest branch, similarly for $ht(\Pi_2)$. The strict ordering for both parts of the pair is $>$ on natural numbers.

Theorem 3 (Cut-elimination). *If $\Gamma \vdash \Delta$ is derivable in LS_{BBI}, then it is also derivable in LS_{BBI} without using the cut rule.*

Proof. By induction on the complexity of the proof in LS_{BBI}. We show that each application of *cut* can either be eliminated, or be replaced by one or more *cut* rules of less complexity. The argument for termination is similar to the cut-elimination proof for *G3ip* [13]. We start to eliminate the topmost *cut* first, and repeat this procedure until there is no *cut* in the derivation. We first show that *cut* can be eliminated when the *cut height* is the lowest, i.e., at least one premise is of height 1. Then we show that the *cut height* is reduced in all cases in which the cut formula is not principal in both premises of cut. If the cut formula is principal in both premises, then the *cut* is reduced to one or more *cut*s on smaller formulae or shorter derivations. Since atoms cannot be principal in logical rules, finally we can either reduce all *cut*s to the case where the cut formula is not principal in both premises, or reduce those *cut*s on compound formulae until their *cut height*s are minimal and then eliminate those *cut*s. □

4 Localising Structural Rules

To obtain an effective proof search procedure for LS_{BBI}, we need to restrict the use of structural rules, which in LS_{BBI}, can permute upwards through all rules except for *id*, $\top^* R$, $*R$, and $-\!\!* \; L$. The other logical rules do not rely on relational atoms, so we can apply them whenever possible. This allows us to design a more compact proof system where applications of structural rules are separated into a special entailment relation for relational atoms. We localise the structural rules Eq_1 and Eq_2 first, and then localise the other structural rules.

Let r be an instance of a structural rule where the substitution used in the rule instance is θ: which is the identity substitution except when r is Eq_1 or Eq_2. We can view r (upwards) as a function that takes a set of relational atoms (in the conclusion of the rule) and outputs another set (in the premise). We write $r(\mathcal{G}, \theta)$ for the output relational atoms of an instance of r with substitution θ and with conclusion containing $\mathcal{G}$. Let σ be a sequence of instances of structural rules $[r_1(\mathcal{G}_1, \theta_1); \cdots; r_n(\mathcal{G}_n, \theta_n)]$. Given a set of relational atoms $\mathcal{G}$, the result of the (backward) application of σ to $\mathcal{G}$, denoted by $\mathcal{S}(\mathcal{G}, \sigma)$, is defined as:

$$\mathcal{S}(\mathcal{G},\sigma) = \begin{cases} \mathcal{G} & \text{if} \quad \sigma = [\,] \\ \mathcal{S}(\mathcal{G}\theta \cup r(\mathcal{G}',\theta), \sigma') & \text{if} \quad \mathcal{G}' \subseteq \mathcal{G} \text{ and } \sigma = [r(\mathcal{G}',\theta);\sigma'] \\ \text{undefined} & \qquad\;\; \text{otherwise} \end{cases}$$

Given a $\sigma = [r_1(\mathcal{G}_1, \theta_1); \cdots; r_n(\mathcal{G}_n, \theta_n)]$, we denote with $subst(\sigma)$ the composite substitution $\theta_1 \circ \cdots \circ \theta_n$, where $t(\theta_1 \circ \theta_2)$ means $(t\theta_1)\theta_2$.

Definition 3. *Let $\mathcal{G}$ be a set of relational atoms. The entailment relation $\mathcal{G} \vdash_E u = v$ holds iff there exists a sequence σ of Eq_1 or Eq_2 structural rules such that $\mathcal{S}(\mathcal{G}, \sigma)$ is defined, and $u\theta = v\theta$, where $\theta = subst(\sigma)$.*

The equality entailment does not fully capture the reflexivity, transitivity, and symmetry of equality. Rather, the structural rule E is used when symmetry is required to derive an equality. As a second step, we isolate the rest of the structural rules into a separate entailment relation, as we did with Eq_1 and Eq_2.

Definition 4. *Let $\mathcal{G}$ be a set of relational atoms. The entailment relation $\vdash_R$ has the following two forms:*

1. *$\mathcal{G} \vdash_R (w_1 = w_2)$ holds iff there is a sequence σ of E, U, A, A_C applications so that $\mathcal{S}(\mathcal{G}, \sigma) \vdash_E (w_1 = w_2)$.*
2. *$\mathcal{G} \vdash_R (w_1, w_2 \triangleright w_3)$ holds iff there is a sequence σ of E, U, A, A_C applications so that $(w_1', w_2' \triangleright w_3') \in \mathcal{S}(\mathcal{G}, \sigma)$ and the following hold: $\mathcal{S}(\mathcal{G}, \sigma) \vdash_E (w_1 = w_1')$, $\mathcal{S}(\mathcal{G}, \sigma) \vdash_E (w_2 = w_2')$, and $\mathcal{S}(\mathcal{G}, \sigma) \vdash_E (w_3 = w_3')$.*

$$\frac{\mathcal{G} \vdash_R (w_1 = w_2)}{\mathcal{G}||\Gamma; w_1 : P \vdash w_2 : P; \Delta}\ id \qquad \frac{(\epsilon, w \triangleright \epsilon); \mathcal{G}||\Gamma \vdash \Delta}{\Gamma; w : \top^* \vdash \Delta}\ \top^* L \qquad \frac{\mathcal{G} \vdash_R (w = \epsilon)}{\mathcal{G}||\Gamma \vdash w : \top^*; \Delta}\ \top^* R$$

$$\frac{\mathcal{S}(\mathcal{G}, \sigma)||\Gamma \vdash x : A; w : A * B; \Delta \qquad \mathcal{S}(\mathcal{G}, \sigma)||\Gamma \vdash y : B; w : A * B; \Delta}{\mathcal{G}||\Gamma \vdash w : A * B; \Delta}\ *R^\dagger$$

$$\frac{\mathcal{S}(\mathcal{G}, \sigma)||\Gamma; w : A \mathrel{-\!\!*} B \vdash x : A; \Delta \qquad \mathcal{S}(\mathcal{G}, \sigma)||\Gamma; w : A \mathrel{-\!\!*} B; z : B \vdash \Delta}{\mathcal{G}||\Gamma; w : A \mathrel{-\!\!*} B \vdash \Delta}\ \mathrel{-\!\!*} L^\ddagger$$

$\dagger$: σ is a derivation of $\mathcal{G} \vdash_R (x, y \triangleright w)$ $\qquad$ $\ddagger$: σ is a derivation of $\mathcal{G} \vdash_R (x, w \triangleright z)$

Fig. 3. Changed rules in LS^{sf}_{BBI}

Thus we can move all the structural rules into $\vdash_R$, giving an intermediate system LS^{sf}_{BBI}, with changed logical rules shown in Figure 3. We write $\mathcal{G}||\Gamma \vdash \Delta$ to emphasise that the left hand side of a sequent is partitioned into relational atoms $\mathcal{G}$ and labelled formulae Γ. Note that the entailment $\vdash_R$ is not a premise, but a side condition for the rule to be applicable.

Theorem 4. *A sequent $\Gamma \vdash \Delta$ is derivable in LS_{BBI} iff it is derivable in LS^{sf}_{BBI}.*

5 Mapping Proof Search to Constraint Solving

The intermediate system LS^{sf}_{BBI} is essentially a variant of LS_{BBI} that packages structural rules at certain points in the proof search. We can further separate proof search into two stages: guessing the shape of the derivation tree, and then checking that each entailment $\vdash_R$ can be proved. The latter involves guessing a relational atom to use in the $*R$ or $\mathrel{-\!\!*} L$ rule which also satisfies the equality constraints in the *id* and $\top^* R$ rules. We formalise this via a symbolic proof system where the relational atoms in the rules $*R$, $\mathrel{-\!\!*} L$ are selected lazily via the introduction of *free variables*, that must be instantiated to concrete labels satisfying all the constraints in the derivation.

Free variables help to make the right decisions when applying $*R$ and $\mathrel{-\!\!*} L$ rules. That is, suppose the *id* rule (or analogously the $\top^* R$ rule) requires a free variable $\mathbf{x}$ to be equal to a label w, we can satisfy this by globally assigning w to $\mathbf{x}$. In this way, the search space is reduced, and many applications of structural rules are guided by the result of *id* and $\top^* R$ rules. See Section 6 for an example.

In our symbolic system $FVLS_{BBI}$, free variables are denoted by $\mathbf{x}$, $\mathbf{y}$ and $\mathbf{z}$. We use $\mathbf{u}, \mathbf{v}, \mathbf{w}$ for either labels or free variables, and a, b, c for ordinary labels. A *symbolic sequent* is a sequent possibly with occurrences of free variables in place of labels. We shall sometimes refer to the normal (non-symbolic) sequent as a *ground sequent* to emphasise that it contains no free variables. The symbolic proof system $FVLS_{BBI}$ is given in Figure 4. The rules are mostly similar to LS^{sf}_{BBI}, but lacking the entailment relations $\vdash_R$. Instead, constraints containing new free variables are introduced when applying $*R$ and $\mathrel{-\!\!*} L$ backwards. Notice also that in $FVLS_{BBI}$, the $*R$ and $\mathrel{-\!\!*} L$ rules do not compute the set $\mathcal{S}(\mathcal{G}, \sigma)$.

$$\frac{}{\mathcal{G}||\Gamma; \mathbf{w}_1 : P \vdash \mathbf{w}_2 : P; \Delta}\ {\scriptstyle id} \qquad \frac{}{\mathcal{G}||\Gamma; \mathbf{w} : \bot \vdash \Delta}\ {\scriptstyle \bot L} \qquad \frac{}{\mathcal{G}||\Gamma \vdash \mathbf{w} : \top; \Delta}\ {\scriptstyle \top R}$$

$$\frac{\mathcal{G}; (\epsilon, \mathbf{w} \triangleright \epsilon)||\Gamma \vdash \Delta}{\mathcal{G}||\Gamma; \mathbf{w} : \top^* \vdash \Delta}\ {\scriptstyle \top^* L} \qquad \frac{}{\mathcal{G}||\Gamma \vdash \mathbf{w} : \top^*; \Delta}\ {\scriptstyle \top^* R}$$

$$\frac{\mathcal{G}||\Gamma; \mathbf{w} : A; \mathbf{w} : B \vdash \Delta}{\mathcal{G}||\Gamma; \mathbf{w} : A \wedge B \vdash \Delta}\ {\scriptstyle \wedge L} \qquad \frac{\mathcal{G}||\Gamma \vdash \mathbf{w} : A; \Delta \qquad \mathcal{G}||\Gamma \vdash \mathbf{w} : B; \Delta}{\mathcal{G}||\Gamma \vdash \mathbf{w} : A \wedge B; \Delta}\ {\scriptstyle \wedge R}$$

$$\frac{\mathcal{G}||\Gamma \vdash \mathbf{w} : A; \Delta \qquad \mathcal{G}||\Gamma; \mathbf{w} : B \vdash \Delta}{\mathcal{G}||\Gamma; \mathbf{w} : A \to B \vdash \Delta}\ {\scriptstyle \to L} \qquad \frac{\mathcal{G}||\Gamma; \mathbf{w} : A \vdash \mathbf{w} : B; \Delta}{\mathcal{G}||\Gamma \vdash \mathbf{w} : A \to B; \Delta}\ {\scriptstyle \to R}$$

$$\frac{\mathcal{G}; (a, b \triangleright \mathbf{w})||\Gamma; a : A; b : B \vdash \Delta}{\mathcal{G}||\Gamma; \mathbf{w} : A * B \vdash \Delta}\ {\scriptstyle *L^\dagger} \qquad \frac{\mathcal{G}; (a, \mathbf{w} \triangleright c)||\Gamma; a : A \vdash c : B; \Delta}{\mathcal{G}||\Gamma \vdash \mathbf{w} : A \mathrel{-\!\!*} B; \Delta}\ {\scriptstyle \mathrel{-\!\!*} R^\ddagger}$$

$$\frac{\mathcal{G}||\Gamma \vdash \mathbf{x} : A; \mathbf{w} : A * B; \Delta \qquad \mathcal{G}||\Gamma \vdash \mathbf{y} : B; \mathbf{w} : A * B; \Delta}{\mathcal{G}||\Gamma \vdash \mathbf{w} : A * B; \Delta}\ {\scriptstyle *R^\sharp}$$

$$\frac{\mathcal{G}||\Gamma; \mathbf{w} : A \mathrel{-\!\!*} B \vdash \mathbf{x} : A; \Delta \qquad \mathcal{G}||\Gamma; \mathbf{w} : A \mathrel{-\!\!*} B; \mathbf{y} : B \vdash \Delta}{\mathcal{G}||\Gamma; \mathbf{w} : A \mathrel{-\!\!*} B \vdash \Delta}\ {\scriptstyle \mathrel{-\!\!*} L^\natural}$$

$\dagger$: a, b do not occur in the conclusion of $*L$
$\ddagger$: a, c do not occur in the conclusion of $\mathrel{-\!\!*} R$
$\sharp$: $\mathbf{x}, \mathbf{y}$ do not occur in the conclusion of $*R$
$\natural$: $\mathbf{x}, \mathbf{z}$ do not occur in the conclusion of $\mathrel{-\!\!*} L$

Fig. 4. Labelled sequent calculus $FVLS_{BBI}$ for Boolean BI

So the relational atoms in $FVLS_{BBI}$ are those that are created by $*L$, $\mathrel{-\!\!*} R$, $\top^* L$. We refer to a derivation in $FVLS_{BBI}$ as a *symbolic derivation.*

An *equality constraint* is an expression of the form $\mathcal{G} \vdash^?_R (\mathbf{u} = \mathbf{v})$, and a *relational constraint* is of the form $\mathcal{G} \vdash^?_R (\mathbf{u}, \mathbf{v} \triangleright \mathbf{w})$. Constraints are ranged over by $\mathfrak{c}, \mathfrak{c}', \mathfrak{c}_1, \mathfrak{c}_2$, etc. We write $\mathcal{G}(\mathfrak{c})$ for the left hand side $\mathcal{G}$ of $\mathfrak{c}$. We write $\mathcal{G} \vdash^?_R C$ for either an equality or relational constraint. We write $fv(\mathfrak{c})$ for the set of free variables in $\mathfrak{c}$, and $fv(\mathcal{C})$ for the set of free variables in a set of constraints $\mathcal{C}$.

Definition 5 (Constraint systems). *A* constraint system *is a pair* $(\mathcal{C}, \preceq)$ *of a set of constraints and a well-founded partial order on elements of* $\mathcal{C}$ *satisfying* ***Monotonicity:*** $\mathfrak{c}_1 \preceq \mathfrak{c}_2$ *implies* $\mathcal{G}(\mathfrak{c}_1) \subseteq \mathcal{G}(\mathfrak{c}_2)$. *It is* well-formed *if it also satisfies* ***Unique variable origin:*** $\forall \mathbf{x}$ *in* $\mathcal{C}$*, there exists a unique minimum (w.r.t.* $\preceq$*) constraint* $\mathfrak{c}(\mathbf{x}) = \mathcal{G}_x \vdash^?_R (\mathbf{u}, \mathbf{v} \triangleright \mathbf{w})$ *s.t.* $\mathbf{x}$ *occurs in* $(\mathbf{u}, \mathbf{v} \triangleright \mathbf{w})$*, but not in* $\mathcal{G}_x$*, and* $\mathbf{x}$ *does not occur in any* $\mathfrak{c}'$ *where* $\mathfrak{c}' \preceq \mathfrak{c}(\mathbf{x})$*. Such a* $\mathfrak{c}(\mathbf{x})$ *is the origin of* $\mathbf{x}$.

From now on, we use $\mathfrak{c}(\mathbf{x})$ for the origin (constraint) of $\mathbf{x}$, as defined above. We use $\mathbb{C}$ to range over constraint systems. We write $\mathfrak{c}_i \prec \mathfrak{c}_j$ when $\mathfrak{c}_i \preceq \mathfrak{c}_j$ and $\mathfrak{c}_i \neq \mathfrak{c}_j$. Further, we define a direct successor relation $\lessdot$ as follows: $\mathfrak{c}_i \lessdot \mathfrak{c}_j$ iff $\mathfrak{c}_i \prec \mathfrak{c}_j$ and there does not exist any $\mathfrak{c}_k$ such that $\mathfrak{c}_i \prec \mathfrak{c}_k \prec \mathfrak{c}_j$.

During proof search, associated constraints are generated as follows. Note that the labels for constraints correspond to those in Figure 4.

Definition 6. *To a given symbolic derivation* Π*, we associate a set of constraints* $\mathcal{C}(\Pi)$ *as follows where the lowest rule instance of* Π *is:*

id	$\mathcal{C}(\Pi) = \{\mathcal{G} \vdash^?_R (\mathbf{w}_1 = \mathbf{w}_2)\}$
$\top^* R$	$\mathcal{C}(\Pi) = \{\mathcal{G} \vdash^?_R (\mathbf{w} = \epsilon)\}$
$*R$	$\mathcal{C}(\Pi) = \mathcal{C}(\Pi_1) \cup \mathcal{C}(\Pi_2) \cup \{\mathcal{G} \vdash^?_R (\mathbf{x}, \mathbf{y} \triangleright \mathbf{w})\}$ *where the left premise derivation is* Π_1 *and the right-premise derivation is* Π_2
$-\!\!* \, L$	$\mathcal{C}(\Pi) = \mathcal{C}(\Pi_1) \cup \mathcal{C}(\Pi_2) \cup \{\mathcal{G} \vdash^?_R (\mathbf{x}, \mathbf{w} \triangleright \mathbf{y})\}$ *where the left premise derivation is* Π_1 *and the right-premise derivation is* Π_2
–	*If* Π *ends with any other rule, with premise derivations* $\{\Pi_1, \ldots, \Pi_n\}$*, then* $\mathcal{C}(\Pi) = \mathcal{C}(\Pi_1) \cup \cdots \cup \mathcal{C}(\Pi_n)$.

Each constraint $\mathfrak{c} \in \mathcal{C}(\Pi)$ corresponds to a rule instance $r(\mathfrak{c})$ in Π where $\mathfrak{c}$ is generated. The ordering of the rule applications in the derivation tree of Π then naturally induces a partial order on $\mathcal{C}(\Pi)$. That is, let $\preceq^\Pi$ be an ordering on $\mathcal{C}(\Pi)$ defined via: $\mathfrak{c}_1 \preceq^\Pi \mathfrak{c}_2$ iff $r(\mathfrak{c}_1)$ is applied below $r(\mathfrak{c}_2)$ on the same branch. Obviously $\preceq^\Pi$ is a partial order. The following property of $\mathcal{C}(\Pi)$ is easy to verify.

Lemma 2. *Let* Π *be a symbolic derivation. Then* $(\mathcal{C}(\Pi), \preceq^\Pi)$ *is a constraint system. Moreover, if the root sequent is ground, then* $(\mathcal{C}(\Pi), \preceq^\Pi)$ *is well-formed.*

Given a symbolic derivation Π, let $\mathbb{C}(\Pi)$ be a constraint system $(\mathcal{C}(\Pi), \preceq^\Pi)$ defined as above. From Lemma 2, if $\mathbb{C}(\Pi) \neq \{\ \}$, then there exists a minimum constraint $\mathfrak{c}$, w.r.t. the partial order $\preceq^\Pi$, such that $\mathcal{G}(\mathfrak{c})$ is ground.

We now define the solvability of a constraint system. This requires that (ternary) relational atoms created by the solution must be accumulated across different constraints, in order to guarantee the soundness of $FVLS_{BBI}$. A *free variable substitution* θ is a mapping from free variables to free variables or labels with finite domain. We denote with $dom(\theta)$ the domain of θ. Given θ and a set V of free variables, $\theta \restriction V$ is the substitution obtained from θ by restricting the domain to V as shown below left. Given θ and θ' such that $dom(\theta') \subseteq dom(\theta)$, we define $\theta \setminus \theta'$ as the substitution as shown below right:

$$\mathbf{x}(\theta \restriction V) = \begin{cases} \mathbf{x}\theta \text{ if } \mathbf{x} \in V \\ \mathbf{x} \;\; \text{otherwise.} \end{cases} \qquad \mathbf{x}(\theta \setminus \theta') = \begin{cases} \mathbf{x}\theta \text{ if } \mathbf{x} \notin dom(\theta') \\ \mathbf{x} \;\; \text{otherwise.} \end{cases}$$

Definition 7 (Simple constraints and their solutions). *A constraint* $\mathfrak{c}$ *is* simple *if its left hand side* $\mathcal{G}(\mathfrak{c})$ *contains no free variables. A* solution (θ, σ) *to a simple constraint* $\mathfrak{c}$ *is a substitution* θ *and a sequence* σ *of structural rules s.t.*

1. *If* $\mathfrak{c}$ *is* $\mathcal{G} \vdash^?_R (\mathbf{u} = \mathbf{v})$ *then* σ *is a derivation of* $\mathcal{G} \vdash_R (\mathbf{u}\theta = \mathbf{v}\theta)$.
2. *If* $\mathfrak{c}$ *is* $\mathcal{G} \vdash^?_R (\mathbf{u}, \mathbf{v} \triangleright \mathbf{w})$ *then* σ *is a derivation of* $\mathcal{G} \vdash_R (\mathbf{u}\theta, \mathbf{v}\theta \triangleright \mathbf{w}\theta)$.

The minimum constraints of a well-formed constraint system are simple.

Definition 8 (Restricting a constraint system). *Let* $\mathbb{C} = (\mathcal{C}, \preceq)$ *be a well-formed constraint system, and* $\mathfrak{c}$ *be a minimum (simple) constraint in* $\mathbb{C}$. *Let* (θ, σ) *be a solution to* $\mathfrak{c}$ *and* $\mathcal{G}' = \mathcal{S}(\mathcal{G}(\mathfrak{c}), \sigma)$. *Define a function* f *on constraints:*

$$f(\mathfrak{c}') = \begin{cases} (\mathcal{G}' \cup \mathcal{G}\theta \vdash^?_R C\theta) & \textit{if } \mathfrak{c}' = (\mathcal{G} \vdash^?_R C) \in \mathcal{C} \setminus \{\mathfrak{c}\} \textit{ and } \mathfrak{c} \preceq \mathfrak{c}', \\ \mathfrak{c}' & \textit{otherwise.} \end{cases}$$

The restriction of $\mathbb{C}$ *by* $(\mathfrak{c}, \theta, \sigma)$*, written* $\mathbb{C} \upharpoonright (\mathfrak{c}, \theta, \sigma)$*, is the pair* $(\mathcal{C}', \preceq')$*, where (1)* $\mathcal{C}' = \{f(\mathfrak{c}') \mid \mathfrak{c}' \in \mathcal{C} \setminus \{\mathfrak{c}\}\}$ *and (2)* $f(\mathfrak{c}_1) \preceq' f(\mathfrak{c}_2)$ *iff* $\mathfrak{c}_1 \preceq \mathfrak{c}_2$*.*

Lemma 3. *The pair* $(\mathcal{C}', \preceq') = \mathbb{C} \upharpoonright (\mathfrak{c}, \theta, \sigma)$ *is a well-formed constraint system.*

Definition 9 (Solution to a well-formed constraint system). *Let* $\mathbb{C} = (\{\mathfrak{c}_1, \ldots, \mathfrak{c}_n\}, \preceq)$ *be a well-formed constraint system. A* solution $(\theta, \{\sigma_1, \ldots, \sigma_n\})$ *to* $\mathbb{C}$ *is a substitution and a set of sequences of structural rules, such that:*

If $n = 0$ *then* $(\theta, \{\sigma_1, \ldots, \sigma_n\})$ *is trivially a solution.*
If $n \geq 1$ *then there must exist some minimum (simple) constraint in* $\mathbb{C}$*. For any minimum constraint* $\mathfrak{c}_i$*, let* $\theta_i = \theta \upharpoonright fv(\mathfrak{c}_i)$*, then* (θ_i, σ_i) *is a solution to* $\mathfrak{c}_i$*, and* $(\theta \setminus \theta_i, \{\sigma_1, \ldots, \sigma_n\} \setminus \sigma_i)$ *is a solution to* $\mathbb{C} \upharpoonright (\mathfrak{c}_i, \theta_i, \sigma_i)$*.*

In Definition 9, suppose a constraint system $\mathbb{C} = (\{\mathfrak{c}_1, \cdots, \mathfrak{c}_n\}, \preceq)$ has a solution $(\theta, \{\sigma_1, \cdots, \sigma_n\})$. For each constraint $\mathfrak{c}_i$ in $\mathbb{C}$, let $\mathfrak{c}'_i$ be the simple constraint obtained from $\mathfrak{c}_i$ in the process of restricting $\mathbb{C}$. Then there is a solution (θ_i, σ_i) to $\mathfrak{c}'_i$, where $\theta_i = \theta \upharpoonright fv(\mathfrak{c}'_i)$, and $\sigma_i \in \{\sigma_1, \cdots, \sigma_n\}$.

Theorem 5 (Soundness). *Let* Π *be a symbolic derivation of a ground sequent* $\mathcal{G}||\Gamma \vdash \Delta$*. If* $\mathcal{C}(\Pi)$ *is solvable, then* $\mathcal{G}||\Gamma \vdash \Delta$ *is derivable in* LS_{BBI}^{sf}*.*

The proof [7] uses induction on the height of symbolic derivations. Intuitively, the proof progressively "grounds" a symbolic derivation, root-upwards. At each inductive step we show that grounding the premises corresponds to restricting the constraint system induced by the symbolic derivation.

We prove the completeness of $FVLS_{BBI}$ by showing that for every cut-free derivation Π of a (ground) sequent in LS_{BBI}^{sf}, there is a symbolic derivation Π' of the same sequent such that $\mathcal{C}(\Pi')$ is solvable. Obviously, Π' should have exactly the same rule applications as Π; the only difference is that some relational atoms are omitted in the derivation, but instead are accumulated in the constraint system. Additionally, some (new) labels are replaced with free variables. So the key is to recover the omitted relational atoms in each sequent from the constraint system while solving constraints. The full proof is in our arXiv paper [7].

Theorem 6 (Completeness). *If a sequent has a* LS_{BBI}^{sf} *derivation* Π*, then it has a symbolic derivation* Π' *such that* $\mathcal{C}(\Pi')$ *is solvable.*

6 A Heuristic and Experimental Results

A heuristic. Suppose we want to prove $((a*b)*c) \to (a*(b*c))$. Using $FVLS_{BBI}$, we build a symbolic derivation as in Figure 5 (right associativity for connectives is assumed). The constraints in the derivation are listed below, with the corresponding rules that generate them:

$$\begin{array}{ll}
id_3\text{:} & (a1, a2 \triangleright a0); (a3, a4 \triangleright a1) \vdash_R^? (a2 = \mathbf{x}8) \\
id_2\text{:} & (a1, a2 \triangleright a0); (a3, a4 \triangleright a1) \vdash_R^? (a4 = \mathbf{x}7) \\
*R_2\text{:} & (a1, a2 \triangleright a0); (a3, a4 \triangleright a1) \vdash_R^? (\mathbf{x}7, \mathbf{x}8 \triangleright \mathbf{x}6) \\
id_1\text{:} & (a1, a2 \triangleright a0); (a3, a4 \triangleright a1) \vdash_R^? (a3 = \mathbf{x}5) \\
*R_1\text{:} & (a1, a2 \triangleright a0); (a3, a4 \triangleright a1) \vdash_R^? (\mathbf{x}5, \mathbf{x}6 \triangleright a0).
\end{array}$$

Let $\Gamma_1 := \{a2 : c\ ;\ a3 : a\}$ and $\Gamma_2 := \{a3 : a\ ;\ a4 : b\}$ in

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{a2 : c; a3 : a; a4 : b \vdash \mathbf{x}5 : a}\,id_1 \qquad \dfrac{\dfrac{}{\Gamma_1; a4 : b \vdash \mathbf{x}7 : b}\,id_2 \quad \dfrac{}{a2 : c; \Gamma_2 \vdash \mathbf{x}8 : c}\,id_3}{a2 : c; a3 : a; a4 : b \vdash \mathbf{x}6 : b * c}\,{*R_2}}{a2 : c; a3 : a; a4 : b \vdash a0 : a * b * c}\,{*R_1}}{a1 : a * b; a2 : c \vdash a0 : a * b * c}\,{*L_2}}{a0 : (a * b) * c \vdash a0 : a * b * c}\,{*L_1}}{\vdash a0 : (a * b) * c \to a * b * c}\,{\to R}$$

Fig. 5. A symbolic derivation for $((a * b) * c) \to (a * (b * c))$

From the constraints generated by *id* rules, we already know how $\mathbf{x}5, \mathbf{x}7, \mathbf{x}8$ should be assigned. In the following, we shall write $(a1, a2 \triangleright a0); (a3, a4 \triangleright a1)$ as $\mathcal{G}$, $(a3, \mathbf{x}6 \triangleright a0); (a2, a4 \triangleright \mathbf{x}6)$ as C, $\mathbf{1}$ as the identity substitution, and $\emptyset$ as an empty sequence of rule applications. Now it is much easier to solve the constraint system with the known information. The last constraint can be solved by $([a3/\mathbf{x}5, w/\mathbf{x}6], A((a1, a2 \triangleright a0); (a3, a4 \triangleright a1), \mathbf{1}))$, which generates $(a3, w \triangleright a0); (a2, a4 \triangleright w)$. The resultant constraint system is restricted to the following:

$$\begin{array}{ll}
id_3\text{:} & (a3, w \triangleright a0); (a2, a4 \triangleright w); (a1, a2 \triangleright a0); (a3, a4 \triangleright a1) \vdash^?_R (a2 = \mathbf{x}8) \\
id_2\text{:} & (a3, w \triangleright a0); (a2, a4 \triangleright w); (a1, a2 \triangleright a0); (a3, a4 \triangleright a1) \vdash^?_R (a4 = \mathbf{x}7) \\
*R_2\text{:} & (a3, w \triangleright a0); (a2, a4 \triangleright w); (a1, a2 \triangleright a0); (a3, a4 \triangleright a1) \vdash^?_R (\mathbf{x}7, \mathbf{x}8 \triangleright w) \\
id_1\text{:} & (a3, w \triangleright a0); (a2, a4 \triangleright w); (a1, a2 \triangleright a0); (a3, a4 \triangleright a1) \vdash^?_R (a3 = a3).
\end{array}$$

Now the last constraint is trivially solved by $(\mathbf{1}, \emptyset)$. The second last constraint can be solved by $([a4/\mathbf{x}7, a2/\mathbf{x}8], E((a2, a4 \triangleright w), \mathbf{1}))$, which generates $(a4, a2 \triangleright w)$. The remaining constraints are restricted as below.

$$\begin{array}{l}
id_3\text{: } (a4, a2 \triangleright w); (a3, w \triangleright a0); (a2, a4 \triangleright w); (a1, a2 \triangleright a0); (a3, a4 \triangleright a1) \vdash^?_R (a2 = a2) \\
id_2\text{: } (a4, a2 \triangleright w); (a3, w \triangleright a0); (a2, a4 \triangleright w); (a1, a2 \triangleright a0); (a3, a4 \triangleright a1) \vdash^?_R (a4 = a4)
\end{array}$$

These constraints are trivially solvable by $(\mathbf{1}, \emptyset)$. Therefore the overall solution to the original (first) constraint system is $([a3/\mathbf{x}5, w/\mathbf{x}6, a4/\mathbf{x}7, a2/\mathbf{x}8], A((a1, a2 \triangleright a0); (a3, a4 \triangleright a1), \mathbf{1}) \cdot E((a2, a4 \triangleright w), \mathbf{1}))$, the reader can check that this solution is compliant with our definitions in Section 5.

But there is a simpler way to see that the label w must exist: the two ternary relational atoms in $\mathcal{G}$ manifest that $a0$ can be split into $a2, a3, a4$. This is exactly what C says. For any variant of $\mathcal{G}$ that describes the same splitting of $a0$ as C, the "internal" node $\mathbf{x}6$ can always be assigned to either an existing label or a label generated by the associativity rule. In the example, $\mathbf{x}6$ cannot be matched to any existing label, so we can assign $\mathbf{x}6$ to be a fresh label globally, and add C to the l.h.s. of the successor constraints in the partial order $\prec$. Similarly for any variant of C with the same splitting of $a0$. The next lemma extends this idea.

Lemma 4. *Given constraints $\mathfrak{c}_1 \lessdot \cdots \lessdot \mathfrak{c}_n$ with $\mathcal{G} = \mathcal{G}(\mathfrak{c}_1) = \cdots = \mathcal{G}(\mathfrak{c}_n)$ s.t. the r.h.s. of $\mathfrak{c}_1$ to $\mathfrak{c}_n$ form a binary tree where every internal node is some free variable $\mathbf{x}$ with $\mathfrak{c}_1 \preceq \mathfrak{c}(\mathbf{x})$, and the other nodes are non-ϵ labels: if $\mathcal{G}' \subseteq \mathcal{G}$ and $\mathcal{G}'$ forms a binary tree with the same root and leaves, then $\mathfrak{c}_1, \cdots, \mathfrak{c}_n$ are solvable.*

Table 1. Initial experimental results

Formula	BBeye (opt)	Naive (Vamp)	$FVLS_{BBI}$ Heuristic
$(a \mathrel{-\!\!*} b) \land (\top * (\top^* \land a)) \to b$	d(2) 0	0.003	0.001
$(\top^* \mathrel{-\!\!*} \neg(\neg a * \top^*)) \to a$	d(2) 0	0.003	0.000
$\neg((a \mathrel{-\!\!*} \neg(a * b)) \land ((\neg a \mathrel{-\!\!*} \neg b) \land b))$	d(2) 0	0.004	0.001
$\top^* \to ((a \mathrel{-\!\!*} (b \mathrel{-\!\!*} c)) \mathrel{-\!\!*} ((a * b) \mathrel{-\!\!*} c))$	d(2) 0.015	0.017	0.001
$\top^* \to ((a * (b * c)) \mathrel{-\!\!*} ((a * b) * c))$	d(2) 0.036	0.006	0.000
$\top^* \to ((a * ((b \mathrel{-\!\!*} e) * c)) \mathrel{-\!\!*} ((a * (b \mathrel{-\!\!*} e)) * c))$	d(2) 0.07	0.019	0.001
$\neg((a \mathrel{-\!\!*} \neg(\neg(d \mathrel{-\!\!*} \neg(a * (c * b))) * a)) \land c * (d \land (a * b)))$	d(2) 0.036	0.037	0.001
$\neg((c * (d * e)) \land B)$ where	d(2) 0.016	0.075	0.039
$B := ((a \mathrel{-\!\!*} \neg(\neg(b \mathrel{-\!\!*} \neg(d * (e * c))) * a)) * (b \land (a * \top)))$			
$\neg(C * (d \land (a * (b * e))))$ where	d(3) 96.639	0.089	0.038
C $:= ((a \mathrel{-\!\!*} \neg(\neg(d \mathrel{-\!\!*} \neg((c * e) * (b * a))) * a)) \land c)$			
$(a * (b * (c * d))) \to (d * (c * (b * a)))$	d(2) 0.009	0.048	0.001
$(a * (b * (c * d))) \to (d * (b * (c * a)))$	d(3) 0.03	0.07	0.001
$(a * (b * (c * (d * e)))) \to (e * (d * (a * (b * c))))$	d(3) 1.625	1.912	0.001
$(a * (b * (c * (d * e)))) \to (e * (b * (a * (c * d))))$	d(4) 20.829	0.333	0.001
$\top^* \to (a * ((b \mathrel{-\!\!*} e) * (c * d)) \mathrel{-\!\!*} ((a * d) * (c * (b \mathrel{-\!\!*} e))))$	d(3) 6.258	0.152	0.007

Experimental results. We used a Dell Optiplex 790 desktop with Intel CORE i7 2600 @ 3.4 GHz CPU and 8GB memory as the platform, and tested the following provers on the formulae from Park et al. [15]. (1) BBeye: the OCaml prover from Park et al. based upon nested sequents [15]; (2) Naive (Vamp): translates a BBI formula into a first-order formula using the standard translation, then uses Vampire 2.6 [6] to solve it; (3) $FVLS_{BBI}$ Heuristic: backward proof search in $FVLS_{BBI}$, using the heuristic-based method to solve the set of constraints.

The results are in Table 1. In the BBeye (opt) column, the d() indicates the depth of proof search. The other two columns are for the two methods stated above. We see that naive translation is comparable with BBeye in most cases, but the latter is not stable. When the tested formulae involve more interaction between structural rules, BBeye runs significantly slower. The heuristic method outperforms all other methods in the tested cases.

Nonetheless, our prover is slower than BBeye for formulae that contain many occurrences of the same atomic formulae, giving (id) instances such as:

$$\Gamma; w_1 : P; w_2 : P; \cdots ; w_n : P \vdash \mathbf{x} : P; \Delta$$

We have to choose some w_i to match with $\mathbf{x}$ without knowing which choice satisfies other constraints. In the worst case, we have to try each using backtracking. Multiple branches of this form lead to a combinatorial explosion. Determinising the concrete labels (worlds) for formulae in proof search in LS_{BBI} or BBeye [15] avoids this problem. Further work is needed to solve this in $FVLS_{BBI}$.

Even though we do not claim the completeness of our heuristics method, it appears to be a fast way to solve certain problems. Completeness can be restored by fully implementing LS_{BBI} or $FVLS_{BBI}$. The derivations in LS_{BBI} are generally shorter than those in the Display Calculus or Nested Sequent Calculus for

$$\frac{(a,b\triangleright c);\Gamma[c/d]\vdash\Delta[c/d]}{(a,b\triangleright c);(a,b\triangleright d);\Gamma\vdash\Delta}\;P\qquad\qquad\frac{(a,b\triangleright c);\Gamma\vdash\Delta}{\Gamma\vdash\Delta}\;T$$

$$\frac{(\epsilon,\epsilon\triangleright\epsilon);\Gamma[\epsilon/a][\epsilon/b]\vdash\Delta[\epsilon/a][\epsilon/b]}{(a,b\triangleright\epsilon);\Gamma\vdash\Delta}\;IU\qquad\frac{(a,b\triangleright c);\Gamma[b/d]\vdash\Delta[b/d]}{(a,b\triangleright c);(a,d\triangleright c);\Gamma\vdash\Delta}\;C$$

In T, a, b do occur in the conclusion but c does not
In all substitutions $[y/x]$, $x\neq\epsilon$

Fig. 6. Some auxiliary structural rules

BBI. The reader can verify that most of formulae in Table 1 can even be proved by hand in a reasonable time using our labelled system. The optimisations of the implementation, however, is out of the scope of this paper.

7 Conclusions, Extensions and Further Work

Our main contribution is a labelled sequent calculus for BBI_{ND} that is sound, complete, and enjoys cut-elimination. There are no explicit contraction rules in LS_{BBI} and all structural rules can be restricted so that proof search is entirely driven by logical rules. We further propose a free variable system to restrict the proof search space so that some applications of $*R$, $-\!\!* \, L$ rules can be guided by zero-premise rules. Although we can structure proof search to be more manageable compared to the unrestricted (labelled or display) calculus, the undecidability of BBI implies that there is no terminating proof search strategy for a sound and complete system. The essence of proof search now resides in guessing which relational atom to use in the $*R$ and $-\!\!* \, L$ rules and whether they need to be applied more than once to a formula. Nevertheless, our initial experimental results already raise the hope that a more efficient proof search strategy can be developed based on our calculus.

An immediate task is to find a complete and terminating (if possible) constraint solving strategy. Although we do not have a counter-model construction procedure for our labelled systems, this aspect has been studied by Larchey-Wendling using labelled tableaux [8]. The possibility to adapt his method to BBI_{ND} using our calculus is also a future work.

Another interesting topic is to extend our calculus to handle some semantics other than the non-deterministic monoidal ones. Our design of the structural rules in LS_{BBI} can be generalised as follows. If there is a semantic condition of the form $(w_{11},w_{12}\triangleright w_{13})\wedge\cdots\wedge(w_{i1},w_{i2}\triangleright w_{i3})\Rightarrow(w'_{11},w'_{12}\triangleright w'_{13})\wedge\cdots\wedge(w'_{j1},w'_{j2}\triangleright w'_{j3})\wedge(x_{11}=x_{12})\wedge\cdots\wedge(x_{k1}=x_{k2})$, we create a rule:

$$\frac{(w'_{11},w'_{12}\triangleright w'_{13});\cdots;(w'_{j1},w'_{j2}\triangleright w'_{j3});(w_{11},w_{12}\triangleright w_{13});\cdots;(w_{i1},w_{i2}\triangleright w_{i3});\Gamma\vdash\Delta}{(w_{11},w_{12}\triangleright w_{13});\cdots;(w_{i1},w_{i2}\triangleright w_{i3});\Gamma\vdash\Delta}\;r$$

And apply substitutions $[x_{12}/x_{11}]\cdots[x_{k2}/x_{k1}]$ globally on the premise, where ϵ is not substituted. Many additional features can be added in this way. We summarise the following desirable ones: (1) PD-semantics: the composition of two elements is either the empty set or a singleton, i.e., $(a, b \triangleright c) \land (a, b \triangleright d) \Rightarrow (c = d)$; (2) TD-semantics: the composition of any two elements is always defined as a singleton, i.e., $\forall a, b, \exists c$ s.t. $(a, b \triangleright c)$; (3) indivisible unit: (cf. Section 1) $(a, b \triangleright \epsilon) \Rightarrow (a = \epsilon) \land (b = \epsilon)$; and (4) cancellative: if $w \circ w'$ is defined and $w \circ w' = w \circ w''$, then $w' = w''$, i.e., $(a, b \triangleright c) \land (a, d \triangleright c) \Rightarrow (b = d)$. Note that (2) and (4) are in addition to (1). The above are formalised in rules P, T, IU, C respectively in Figure 6.

The formula $(F * F) \to F$, where $F = \neg(\top \mathrel{-\!\!*} \neg\top^*)$, differentiates BBI_{ND} and BBI_{PD} [10] and is provable using $LS_{BBI} + P$. Using $LS_{BBI} + T$, we can prove $(\neg\top^* \mathrel{-\!\!*} \bot) \to \top^*$, which is valid in BBI_{TD} but not in BBI_{PD} [10], and also $(\top^* \land ((p * q) \mathrel{-\!\!*} \bot)) \to ((p \mathrel{-\!\!*} \bot) \lor (q \mathrel{-\!\!*} \bot))$, which is valid in separation models iff the composition is total [4]. These additional rules preserve cut-elimination.

Oddly, the formula $\neg(\top^* \land A \land (B * \neg(C \mathrel{-\!\!*} (\top^* \to A))))$, which is valid in BBI_{ND}, is very hard to prove in the display calculus and Park et al.'s method. We ran this formula using Park et al.'s prover for a week on a CORE i7 2600 processor, without success. Very short proofs of this formula exist in LS_{BBI} or Larchey-Wendling and Galmiche's labelled tableaux (this formula must also be valid in BBI_{PD}). We are currently investigating this phenomenon.

References

1. Brotherston, J.: A unified display proof theory for bunched logic. ENTCS 265, 197–211 (2010)
2. Brotherston, J., Calcagno, C.: Classical BI: Its semantics and proof theory. LMCS 6(3) (2010)
3. Brotherston, J., Kanovich, M.: Undecidability of propositional separation logic and its neighbours. In: LICS, pp. 130–139 (2010)
4. Brotherston, J., Kanovich, M.: Undecidability of propositional separation logic and its neighbours. Submitted to the Journal of ACM (2013)
5. Galmiche, D., Larchey-Wendling, D.: Expressivity properties of boolean BI through relational models. In: Arun-Kumar, S., Garg, N. (eds.) FSTTCS 2006. LNCS, vol. 4337, pp. 357–368. Springer, Heidelberg (2006)
6. Hoder, K., Voronkov, A.: Comparing unification algorithms in first-order theorem proving. In: Mertsching, B., Hund, M., Aziz, Z. (eds.) KI 2009. LNCS, vol. 5803, pp. 435–443. Springer, Heidelberg (2009)
7. Hóu, Z., Tiu, A., Goré, R.: A labelled sequent calculus for BBI: Proof theory and proof search. arXiv:1302.4783 (2013)
8. Larchey-Wendling, D.: The formal strong completeness of partial monoidal boolean BI. Submitted to Journal of Logic and Computation (2012)
9. Larchey-Wendling, D., Galmiche, D.: Exploring the relation between intuitionistic BI and boolean BI: An unexpected embedding. MSCS 19(3), 435–500 (2009)
10. Larchey-Wendling, D., Galmiche, D.: The undecidability of boolean BI through phase semantics. In: LICS, pp. 140–149 (2010)
11. Larchey-Wendling, D., Galmiche, D.: Non-deterministic phase semantics and the undecidability of boolean BI. ACM TOCL 14(1) (2013)
12. Negri, S.: Proof analysis in modal logic. JPL 34(5-6), 507–544 (2005)
13. Negri, S., von Plato, J.: Structural Proof Theory. CUP (2001)

14. O'Hearn, P.W., Pym, D.J.: The logic of bunched implications. BSL 5(2), 215–244 (1999)
15. Park, J., Seo, J., Park, S.: A theorem prover for boolean BI. In: POPL 2013, pp. 219–232. ACM, New York (2013)
16. Pym, D.J.: The Semantics and Proof Theory of the Logic of Bunched Implications. Applied Logic Series. Kluwer Academic Publishers (2002)
17. Reynolds, J.C.: Separation logic: A logic for shared mutable data structures. In: LICS 2002, pp. 55–74. IEEE Computer Society (2002)

A Refined Tableau Calculus with Controlled Blocking for the Description Logic $\mathcal{SHOI}$

Mohammad Khodadadi, Renate A. Schmidt, and Dmitry Tishkovsky[⋆]

School of Computer Science, The University of Manchester, UK

Abstract The paper presents a tableau calculus with several refinements for reasoning in the description logic $\mathcal{SHOI}$. The calculus uses non-standard rules for dealing with TBox statements. Whereas in existing tableau approaches a fixed rule is used for dealing with TBox statements, we use a dynamically generated set of refined rules. This approach has become practical because reasoners with flexible sets of rules can be generated with the tableau prover generation prototype MetTeL. We also define and investigate variations of the unrestricted blocking mechanism in which equality reasoning is realised by ordered rewriting and the application of the blocking rule is controlled by excluding its application to a fixed, finite set of individual terms. Reasoning with the unique name assumption and excluding ABox individuals from the application of blocking can be seen as two separate instances of the latter. Experiments show the refinements lead to fewer rule applications and improved performance.

1 Introduction

There exist various tableau algorithms for reasoning in description logics [2]. In this paper we present a refinement of the tableau calculus introduced in [12] for the description logic $\mathcal{SHOI}$. Termination is ensured using a rewriting variant of the unrestricted blocking rule [19]. A sufficient condition for termination using unrestricted blocking is the finite model property [19], which $\mathcal{SHOI}$ is known to have [5]. The core tableau rules are in line with a refined tableau calculus obtained in the tableau synthesis framework [18], but, exploiting the tree model property of $\mathcal{SHOI}$, transitive roles are accommodated via propagation rules rather than structural rules.

Labelled tableau approaches allow for a flexible derivation procedure, and are not limited to logics with a form of tree model property. They are common for modal and description logics, hybrid logics and various other non-classical logics, cf. e.g., [6,2,3,4,1].

Different blocking mechanisms have been developed for description logic tableau algorithms. A common point of these mechanisms is essentially that they exploit kinds of the tree model property. They compare maximally expanded label sets of concept expressions through the construction of tree-like models.

[⋆] This research is supported by UK EPSRC research grant EP/H043748/1.

For more expressive logics, for example, logics with role inverse and nominals, back-and-forth traversal of a tree model is required with implicit backtracking using forms of dynamic blocking [9,10]. These blocking techniques provide strong termination results but some care is needed to ensure soundness. In [16,19], it was shown, the description logics $\mathcal{ALBO}$ and $\mathcal{ALBO}^{\mathsf{id}}$, which do not have the tree model property, can be decided using a labelled tableau approach enhanced by the unrestricted blocking mechanism, while many existing blocking mechanisms are not sufficient for description logics without a kind of tree-model property. The unrestricted blocking mechanism ensures weak termination. It is generic and reverts decisions only when needed, namely, when only contradictions were obtained. While many techniques in the tableau calculus presented in this paper have similarities with techniques in existing tableau approaches, there are also significant differences because our tableau calculus is designed to be proof-confluent and as general as possible. We describe a rewriting variant of the unrestricted blocking rule, because equality reasoning is realised by ordered rewriting. In comparison to tableau calculi using essentially standard tableau rules for equality, as for example [3,16,19], rewriting performs fewer inferences. That is because essentially equivalent inference steps on equivalent individuals are avoided. The ordering ensures only the currently smallest individual term in an equivalence class is present in the current node of the tableau derivation.

While being generic the unrestricted blocking rule creates potentially a large number of branching points in the derivation. It is thus important to investigate ways of controlling the application of blocking while preserving soundness, completeness and termination. The number of applications of the blocking rule can be reduced by imposing additional side conditions or adding premises to the rule. In this paper we discuss a general technique of controlling the blocking rule by not applying it to members of an a priori given, finite set of individual terms. This variant of the blocking rule can be utilised for reasoning in domains with the unique name assumption, or where for example it is assumed that some of the given ABox individuals are distinct.

Non-standard in our approach is the use of dynamically generated rules for statements in the TBox rather than using a fixed rule. These dynamic rules are rule refinements obtained in accordance with [20]. Using the MetTeL tool [22] it is easy to generate tableau provers from tableau calculus specifications. This means there is no need to implement a prover for the new tableau calculi manually. This enables us to easily generate a prover for a specific knowledge base based on a calculus with dynamically generated rules. Following this approach we can build specialised provers for various computer applications using ontologies as the information backbone.

The paper is based on the workshop paper [12]. Its main contributions are threefold. First, it presents a labelled tableau calculus for the description logic $\mathcal{SHOI}$ (Section 3). Second, we discuss a general technique of controlling the blocking rule by disabling its application to individual terms from an a priori given, finite set (Section 4). This approach can be utilised for reasoning in domains with the unique name assumption. Third, we use a novel approach for reasoning with respect to TBox statements (Section 5). Rather than using a fixed tableau rule for TBox statements, we dynamically generate rules for each statement. These dynamic rules are optimised by atomic rule refinement described in [20].

The MetTeL tool [22] allows us to automatically generate a prover for a specific knowledge base based on a calculus with dynamically generated rules. In order to evaluate the provers that use the tableau calculi with dynamically generated rules, an experimental comparison between them and provers that use the fixed tableau rule was undertaken (Section 6). Controlled variants of unrestricted blocking are also evaluated. Two repositories of existing ontologies are used as problem sets.

Additionally, we establish the finite model property for $\mathcal{SHOI}$ (Section 2). In [5], the finite model property for $\mathcal{SHOI}$ is obtained from a terminating tableau algorithm. By contrast, our proof of termination in Section 3 takes the reverse route. In the extended version [13] of this paper we provide an alternative proof of the finite model property for $\mathcal{SHOI}$ by a standard filtration argument that does not involve any form of tableau reasoning.

The paper is an extended version of the workshop paper [14]. Due to space limitations all proofs are omitted but can be found in [13].

2 Syntax and Semantics of $\mathcal{SHOI}$

The description logic $\mathcal{SHOI}$ [11,9] extends the description logic $\mathcal{ALC}$ with singleton concepts, role inverse, transitive roles and role inclusion axioms. Its language is defined over disjoint sets of atomic concepts, atomic roles and individuals. The set of individuals is assumed to be finite. C and D denote concepts, A denotes an atomic concept, R and T denote roles, r denotes an atomic role and a and b denote individuals. Concepts and roles are built from atomic concepts, individuals, and atomic roles using the connectives $\{\cdot\}$ (singleton operator), $\neg$, $\sqcup$, and $\exists\cdot.\cdot$ (existential restriction operator), $^-$ (role inverse operator) as defined by these BNFs:

$$C \overset{\mathsf{def}}{=} A \mid \{a\} \mid \neg C \mid C \sqcup C \mid \exists R.C \qquad \text{and} \qquad R \overset{\mathsf{def}}{=} r \mid R^-.$$

The operators $\top, \bot, \sqcap$ and $\forall\cdot.\cdot$ are defined as usual. We assume that $(r^-)^- \overset{\mathsf{def}}{=} r$ in order to simplify the syntax and avoid repetitive occurrences of the role inverse operator. Further, for every atomic role r, $\mathsf{Trans}(r)$ is used to specify that r is transitive. (The predicate Trans is defined on atomic roles only because, in order to specify that r^- is transitive, it is enough to state that r is transitive.)

A knowledge base consists of an ABox $\mathcal{A}$, a TBox $\mathcal{T}$ and an RBox $\mathcal{R}$. A finite number of concept assertions of the form $a : C$ and role assertions of the form $(a, b) : R$ constitute the ABox. The hierarchy between concepts are expressed in the TBox using a finite set of inclusion statements of the form $C \sqsubseteq D$. The RBox is a finite set of transitivity statements $\mathsf{Trans}(r)$ for some atomic roles r and inclusion statements of the form $R \sqsubseteq T$ which are used to express the hierarchy between roles. Normalisation of the RBox is not assumed.

We define the *closure* $\mathcal{R}^+$ of role inclusions in the RBox $\mathcal{R}$ as the smallest RBox that contains $\mathcal{R}$ and satisfies the following two properties: (i) if $Q \sqsubseteq R \in \mathcal{R}^+$ then $Q^- \sqsubseteq R^- \in \mathcal{R}^+$; (ii) if $Q \sqsubseteq R, R \sqsubseteq T \in \mathcal{R}^+$ then $Q \sqsubseteq T \in \mathcal{R}^+$. Given an RBox $\mathcal{R}$, let $\mathcal{R}^*$ denote the RBox $\mathcal{R}^+ \cup \{R \sqsubseteq R \mid R \text{ is a role}\}$.

A *$\mathcal{SHOI}$-model* $\mathcal{I}$ is a tuple $\mathcal{I} \stackrel{\mathsf{def}}{=} (\Delta^\mathcal{I}, \cdot^\mathcal{I})$, where $\Delta^\mathcal{I}$ is a non-empty *domain* of interpretation and $\cdot^\mathcal{I}$ is an *interpretation function* which maps individuals to elements of $\Delta^\mathcal{I}$, atomic concepts to subsets of $\Delta^\mathcal{I}$, and atomic roles to binary relations over $\Delta^\mathcal{I}$. The interpretation function extends inductively to all concept and role expressions as follows.

$$\{a\}^\mathcal{I} \stackrel{\mathsf{def}}{=} \{a^\mathcal{I}\} \quad (\neg C)^\mathcal{I} \stackrel{\mathsf{def}}{=} \Delta^\mathcal{I} \setminus C^\mathcal{I} \quad (C \sqcup D)^\mathcal{I} \stackrel{\mathsf{def}}{=} C^\mathcal{I} \cup D^\mathcal{I}$$
$$(\exists R.C)^\mathcal{I} \stackrel{\mathsf{def}}{=} \{x \mid \exists y \in C^\mathcal{I}\ (x,y) \in R^\mathcal{I}\} \qquad (R^-)^\mathcal{I} \stackrel{\mathsf{def}}{=} \{(x,y) \mid (y,x) \in R^\mathcal{I}\}$$

For any expression or statement E, E is *true (valid)* in the model $\mathcal{I}$ is denoted by $\mathcal{I} \models E$ and is defined as follows.

$$\begin{array}{lll@{\qquad}lll}
\mathcal{I} \models C & \stackrel{\mathsf{def}}{\iff} & C^\mathcal{I} = \Delta^\mathcal{I} & \mathcal{I} \models a : C & \stackrel{\mathsf{def}}{\iff} & a^\mathcal{I} \in C^\mathcal{I} \\
\mathcal{I} \models R \sqsubseteq T & \stackrel{\mathsf{def}}{\iff} & R^\mathcal{I} \subseteq T^\mathcal{I} & \mathcal{I} \models (a,b) : R & \stackrel{\mathsf{def}}{\iff} & (a^\mathcal{I}, b^\mathcal{I}) \in R^\mathcal{I} \\
\mathcal{I} \models C \sqsubseteq D & \stackrel{\mathsf{def}}{\iff} & C^\mathcal{I} \subseteq D^\mathcal{I} & \mathcal{I} \models \mathsf{Trans}(r) & \stackrel{\mathsf{def}}{\iff} & r^\mathcal{I} \text{ is transitive}
\end{array}$$

A concept C is *satisfiable* in a model $\mathcal{I}$ iff $C^\mathcal{I} \neq \emptyset$. A concept is *satisfiable in $\mathcal{I}$ with respect to a knowledge base* if it is satisfiable in $\mathcal{I}$ whenever every statement of the knowledge base is true in $\mathcal{I}$. That is, C is satisfiable with respect to $(\mathcal{A}, \mathcal{T}, \mathcal{R})$ in $\mathcal{I}$ iff $C^\mathcal{I} \neq \emptyset$ provided that $\mathcal{I} \models E$ for every $E \in \mathcal{A} \cup \mathcal{T} \cup \mathcal{R}$.

The termination result in the next section for our tableau calculus for $\mathcal{SHOI}$ relies on the finite model property of the logic.

Theorem 1 (Finite model property of $\mathcal{SHOI}$ [5,13]). *If a concept C is satisfiable with respect to a knowledge base $(\mathcal{A}, \mathcal{T}, \mathcal{R})$ in a $\mathcal{SHOI}$-model then it is satisfiable with respect to $(\mathcal{A}, \mathcal{T}, \mathcal{R})$ in a finite $\mathcal{SHOI}$-model.*

3 Tableau Calculus $\mathit{Tab}_{\mathcal{SHOI}}$

The language of the tableau calculus is an extension of the language of $\mathcal{SHOI}$ with equality formulae and individual terms used as labels. The set of (individual) terms s is defined inductively by the grammar rule $s \stackrel{\mathsf{def}}{=} a \mid f(s,R,C)$, where a denotes any individual, C any concept, R any role, and f is a (fixed) function symbol. Terms which are not ABox individuals can be viewed as being Skolem terms. Formulae in the *tableau language* are ABox assertions over individual terms, and equalities of terms. More precisely, tableau formulae are defined by the grammar rule $E \stackrel{\mathsf{def}}{=} s : C \mid (s,t) : R \mid s \approx t$, where s and t are individual terms, C is a concept and R is a role.

We extend the interpretation of $\mathcal{SHOI}$ to the tableau language as follows. For every $\mathcal{SHOI}$ interpretation $\mathcal{I}$, let the interpretation $f^\mathcal{I}$ in $\mathcal{I}$ of the function f be an arbitrary function that maps triples (x, ρ, χ) with $x \in \Delta^\mathcal{I}$, $\rho \subseteq (\Delta^\mathcal{I})^2$, $\chi \subseteq \Delta^\mathcal{I}$ to elements of $\Delta^\mathcal{I}$. The semantics of tableau formulae is specified by:

$$(f(a,R,C))^{\mathcal{I}} \overset{\text{def}}{=} f^{\mathcal{I}}(a^{\mathcal{I}},R^{\mathcal{I}},C^{\mathcal{I}}), \qquad \mathcal{I} \models s : C \overset{\text{def}}{\Longleftrightarrow} s^{\mathcal{I}} \in C^{\mathcal{I}},$$
$$\mathcal{I} \models s \approx t \overset{\text{def}}{\Longleftrightarrow} s^{\mathcal{I}} = t^{\mathcal{I}}, \qquad \mathcal{I} \models (s,t) : R \overset{\text{def}}{\Longleftrightarrow} (s^{\mathcal{I}},t^{\mathcal{I}}) \in R^{\mathcal{I}}.$$

Since the interpretations of the formulae $s \approx t$, $s : \{t\}$ and $t : \{s\}$ coincide, we refer to them as *equalities*, and to formulae of the form $s : \neg\{t\}$ as *inequalities*.

Having defined the tableau language, next we give a general description of how tableau derivations are constructed and define important notions of tableaux. Let *Tab* denote a tableau calculus comprising of a set of inference rules. A *derivation* or *tableau* for *Tab* is a finitely branching, ordered tree whose nodes are annotated by sets of tableau formulae. Assuming that C is the input concept to be tested for satisfiability with respect to a knowledge base $(\mathcal{A},\mathcal{T},\mathcal{R})$, the root node of the tableau is the set $\{a : C\} \cup \mathcal{A}$, where a denotes a fresh individual and $\mathcal{A}$ is the ABox. Successor nodes are constructed in accordance with a set of inference rules in the calculus. The inference rules have the general form

$$\frac{X_0}{X_1 \mid \ldots \mid X_n} \text{ (side-condition)},$$

where X_0 is the set of premises and the X_i are the sets of conclusions. If $n = 0$, the rule is called *closure rule* and written $X_0/\bot$.

If a rule of the calculus is applicable to a leaf node of the tableau with a matching substitution μ, and it is applied to the leaf node, then the tableau is extended by attaching to the leaf node n child nodes annotated with $N \cup X_i\mu$ for $i = 1, \ldots n$, respectively. In order to avoid redundancies we stipulate that a rule application to a leaf node annotated with N is *redundant* if there is a conclusion set X_i for some $i = 1, \ldots n$ of the rule such that $X_i\mu \subseteq N$, where μ is the matching substitution. This ensures rules are not applied more than once to the same sets of formulae.

A *branch* in the tableau is a maximal path from the root of the tableau to a leaf node. If a closure rule has been applied in a branch then the branch is said to be *closed*. If a branch is not closed, it is called *open*. A tableau is *closed* if all its branches are closed. A branch is *fully expanded* if no more rules are applicable to its leaf node modulo redundancy. We call a tableau *fully expanded* iff all its branches are fully expanded. We denote by $\mathit{Tab}(\mathcal{A},\mathcal{T},\mathcal{R},C)$ a fully expanded tableau constructed using the calculus *Tab* for the input concept C (to be tested for satisfiability) and the knowledge base $(\mathcal{A},\mathcal{T},\mathcal{R})$.

We use equality reasoning for individual terms to achieve termination for the calculus. Equality reasoning can be provided in various ways. One is to supply special tableau rules for reasoning modulo equalities within the branch in a similar way as it is done in [3,16,19]. Another is to use ordered term rewriting. Ordered rewriting is more efficient for handling equal individuals because it allows to reduce the number of tableau formulae in the current branch. Since all individual terms in any tableau derivation are ground, we are dealing with a special case of rewriting, namely, ground rewriting.

In this paper, a *rewrite system* R is a binary relation on the set of all individual terms and consists of rewrite rules which are pairs of individual terms. In order

$$(\bot)\colon \frac{s:\neg C,\ s:C}{\bot}$$

$$(\exists)\colon \frac{s:\exists R.C}{f(s,R,C):C,\ (s,f(s,R,C)):R}$$

$$(\neg\exists)\colon \frac{s:\neg\exists T.C,\ (s,t):R}{t:\neg C}\ (R\sqsubseteq T\in\mathcal{R}^*)$$

$$(\neg\exists^-)\colon \frac{s:\neg\exists T^-.C,\ (t,s):R}{t:\neg C}\ (R\sqsubseteq T\in\mathcal{R}^*)$$

$$(\mathsf{tr})\colon \frac{s:\neg\exists T.C,\ (s,t):R}{t:\neg\exists R.C}\ (R\sqsubseteq T\in\mathcal{R}^*,\ \mathsf{Trans}(R)\in\mathcal{R})$$

$$(\mathsf{tr}^-)\colon \frac{s:\neg\exists T^-.C,\ (t,s):R}{t:\neg\exists R^-.C}\ (R\sqsubseteq T\in\mathcal{R}^*,\ \mathsf{Trans}(R)\in\mathcal{R})$$

$$(\mathsf{TBox})\colon \frac{s:\{s\}}{s:(\neg C\sqcup D)}\ (C\sqsubseteq D\in\mathcal{T})$$

$$(\mathsf{RBox})\colon \frac{(s,t):R}{(s,t):T}\ (R\sqsubseteq T\in\mathcal{R}^+)$$

$$(\neg\neg)\colon \frac{s:\neg\neg C}{s:C}$$

$$(\sqcup)\colon \frac{s:C\sqcup D}{s:C\mid s:D}$$

$$(\neg\sqcup)\colon \frac{s:\neg(C\sqcup D)}{s:\neg C,\ s:\neg D}$$

$$(^-)\colon \frac{(s,t):R^-}{(t,s):R}$$

$$(\mathsf{id}_1)\colon \frac{s:C}{s:\{s\}}$$

$$(\mathsf{id}_2)\colon \frac{s:\neg\{t\}}{t:\{t\}}$$

$$(\mathsf{id}_3)\colon \frac{(s,t):R}{s:\{s\},\ t:\{t\}}$$

$$(\approx)\colon \frac{s:\{t\}}{s\approx t}\ (s\neq t)$$

Fig. 1. The tableau calculus $Tab_{\mathcal{SHOI}}$

to handle equalities, we orient each equality formula appearing in the current branch according to a special, strict partial ordering $\succ$ on individual terms. We denote by $s \rightarrow t$ a rewrite rule (s,t) in which $s \succ t$. Thus, if an equality formula $s \approx t$ appears in a node of a branch then either $s \rightarrow t$ or $t \rightarrow s$ is added as a rewrite rule to the rewrite system of the branch.

Our tableau calculus $Tab_{\mathcal{SHOI}}$ for the description logic $\mathcal{SHOI}$ is given in Figure 1. The $(\bot)$ rule is the closure rule. The $(\neg\neg)$ rule removes occurrences of double negation on concepts. The $(\sqcup)$ and $(\neg\sqcup)$ rules are standard rules for handling concept disjunctions. Given a tableau formula $s : \exists R.C$, the $(\exists)$ rule introduces Skolem term $f(s,R,C)$, as an R-successor of s (instead of introducing a fresh individual as might be done in other presentations). Using Skolem terms has many advantages that outweigh drawbacks and perceived inconveniences. In our setting, Skolem terms provide a convenient technical device to keep track of the order in which witnesses for existential quantification were introduced and record dependency on other witnesses. Such dependencies are then used to stop redundant rule applications when combined with term rewriting. In systems not using Skolem terms this information is typically captured by an ordering on individual constants. In addition, in combination with blocking there is no need to redo inference steps with existential extent that have already been performed or resurrect phantom concepts. That is because when rewriting happening on terms, we also rewrite their dependent Skolem terms and consequently some applications of the $(\exists)$ rule become redundant. For example, rewriting of the term $f(i,R,C)$ to i causes terms such as $f(f(i,R,C),R,C)$ and $f(f(f(i,R,C),R,C),R,C)$, which may appear in formulae in a branch, to be rewritten to i. There is also no need for status variables to keep track of whether individual constants are active or phantom in the deduction process.

The $(\neg\exists)$ rule is equivalent to the standard rule for universally restricted concepts. The $(\neg\exists^-)$ rule allows the backward propagation of concepts along inverted links. The $(^{-})$ rule inverts a given link. The (tr) rule propagates negated existential concept restriction along a transitive link, while the (tr^{-}) rule does the same for inverse occurrences of transitive roles.

Equalities of the form $s : \{s\}$ are tautologies, which are used in our calculus as domain predicates for keeping track of the terms that have been introduced to a branch. This is achieved with the three rules (id_1), (id_2) and (id_3).

The $(\approx)$ rule is a special rule adding, what we call, a *rewrite trigger* $s \approx t$ to the branch. Let $\succ$ be any reduction ordering on the set of individuals in the branch. The addition of any tableau formula $s \approx t$ to a set N of formulae, which annotates a leaf tableau node, immediately triggers the following rewrite process. Suppose that $s \succ t$ (the case $t \succ s$ is symmetrical). Then, $s \to t$ is added to a rewrite system R associated with the current tableau branch. The tableau is extended by attaching one child node to the current leaf node. The child node is annotated by the set N' obtained by rewriting all the tableau formulae in N with respect to the rewrite system R. In particular, this means that, in N' every term s is replaced by a term u such that $s \overset{*}{\to} u$ with respect to R.

For each concept inclusion $C \sqsubseteq D$ of the TBox, the (TBox) rule propagates the concept $\neg C \sqcup D$ to every label occurring on the branch. The (RBox) rule propagates a link of a role into its super role according to the closure $\mathcal{R}^{+}$ of the given RBox $\mathcal{R}$. In Section 5 we replace the (TBox) rule by dynamically generated rules.

It is not difficult to see that each rule of $\mathit{Tab}_{\mathcal{SHOI}}$ preserves satisfiability. Consequently we can state:

Theorem 2 (Soundness). *The tableau calculus* $\mathit{Tab}_{\mathcal{SHOI}}$ *is sound for* $\mathcal{SHOI}$. *That is, if a concept* C *is satisfiable with respect to the knowledge base* $(\mathcal{A}, \mathcal{T}, \mathcal{R})$ *then any fully expanded* $\mathit{Tab}_{\mathcal{SHOI}}$*-tableau for* $(\mathcal{A}, \mathcal{T}, \mathcal{R}, C)$ *has an open branch.*

A tableau calculus *Tab* is *complete* iff for every knowledge base $(\mathcal{A}, \mathcal{T}, \mathcal{R})$ and every concept C if C is unsatisfiable with respect to $(\mathcal{A}, \mathcal{T}, \mathcal{R})$ then there is a closed tableau $\mathit{Tab}(\mathcal{A}, \mathcal{T}, \mathcal{R}, C)$. In order to prove completeness of $\mathit{Tab}_{\mathcal{SHOI}}$, we prove its constructive completeness, which implies completeness. A tableau calculus *Tab* is *constructively complete* if for every open branch in any fully expanded tableau $\mathit{Tab}(\mathcal{A}, \mathcal{T}, \mathcal{R}, C)$ there is a model that validates the knowledge base $(\mathcal{A}, \mathcal{T}, \mathcal{R})$ and satisfies C.

Theorem 3 (Completeness). $\mathit{Tab}_{\mathcal{SHOI}}$ *is a (constructively) complete tableau calculus for the description logic* $\mathcal{SHOI}$.

A form of blocking or loop-checking is necessary in order to ensure termination. We achieve termination by incorporating a variation of the *unrestricted blocking* mechanism described in [16] into the tableau calculus. In [16] equality reasoning is realised by tableau equality rules, whereas in this paper ordered rewriting is used. We therefore adapt the unrestricted blocking rule from [16] as follows:

$$\text{(ub)}: \frac{s : \{s\},\ t : \{t\}}{s \approx t \mid s : \neg\{t\}}\ (s \neq t).$$

In order to achieve termination the following condition must hold.

Termination Condition: In every open branch there is some node from which point onward before any application of the ($\exists$) rule, all possible applications of the (ub) rule have been performed.

The (ub) rule is applicable to any pair of distinct individual terms that are used as labels in the current leaf node. When it is applied, two tableau successor nodes are created. In the left node, $s \approx t$ acts as a trigger which induces rewriting modulo derived equalities. In the right node, $s : \neg\{t\}$ indicates that s and t are not equal. The blocking is reversible, because, when no models can be found in the left branch, reversion is performed through standard backtracking.

Let $\mathit{Tab}_{\mathcal{SHOI}}(\mathrm{ub})$ be the calculus consisting of all the rules of $\mathit{Tab}_{\mathcal{SHOI}}$ and the (ub) rule. Since, the (ub) rule is sound, and $\mathit{Tab}_{\mathcal{SHOI}}$ is sound and (constructively) complete (Theorem 3), we get:

Theorem 4. $\mathit{Tab}_{\mathcal{SHOI}}(\mathrm{ub})$ *is a sound and (constructively) complete for* $\mathcal{SHOI}$*.*

Based on [17,19] it can be shown that adding the rewriting version of unrestricted blocking to a sound and constructively complete, ground semantic tableau calculus ensures termination, if the logic has the finite model property. A tableau calculus *Tab* is *(weakly) terminating* iff for any finite set N, every closed tableau $\mathit{Tab}(N)$ is finite and every open tableau $\mathit{Tab}(N)$ has a finite open branch [18]. A procedure based on a tableau calculus is *fair* if any inference that is possible is performed eventually [19].

Theorem 5 (Termination). *Any fair procedure based on the tableau calculus* $\mathit{Tab}_{\mathcal{SHOI}}(\mathrm{ub})$ *is terminating for satisfiability in* $\mathcal{SHOI}$*.*

As branch selection fairness is particularly important, this provides a weak termination result and means that in an implementation breadth-first search or the more efficient depth-first iterative deepening search gives a decision procedure. Mainstream description logic tableau algorithms with less eager blocking conditions are strongly terminating. We expect to be able to show termination for algorithms based on $\mathit{Tab}_{\mathcal{SHOI}}(\mathrm{ub})$ using depth-first left-to-right search as well.

Theorem 6 (Decidability). *Any fair procedure based on the tableau calculus* $\mathit{Tab}_{\mathcal{SHOI}}(\mathrm{ub})$ *and satisfying the termination condition is a decision procedure for* $\mathcal{SHOI}$ *and its sublogics.*

4 Controlling the Application of Blocking Using ($\mathbf{ub_{noS}}$)

The (ub) rule may potentially create a large number of branching points in the derivation, as it is applicable to all pairs of individual terms in the branch. The situation is worse if the knowledge base contains a large number of individuals and $\exists$-expressions. Also if the input concept is unsatisfiable with respect to the knowledge base then no blocking inference steps are needed. However not blocking is not an option, as it is not known in advance if a problem is unsatisfiable or not. Examples show without blocking, it is not possible to avoid infinite branches. It is thus important to find ways of controlling the application of blocking without loosing termination. We may reduce the number of applications of the (ub) rule, and reduce the search space by imposing appropriate side conditions on the application of the blocking rule. Ideal are side-conditions, and additional premises, that maximise the chance of constructing a finite model without the need for backtracking. It is however not possible to know which identification of individual terms will aid the discovery of a finite model quickly. It is clear that systematic approaches for selecting individual terms to identify are needed, and different approaches display different performances.

The following theorem holds for arbitrary restrictions of the (ub) rule.

Theorem 7 (Soundness and completeness). *The (ub) rule constrained by any additional premises or side-conditions is sound.* Tab$_{\mathcal{SHOI}}$ *extended with such a constrained rule is thus sound and constructively complete for* $\mathcal{SHOI}$*.*

In this section we introduce a general technique for controlling the application of the (ub) rule. One possible way of controlling the (ub) rule is to find individual terms whose identification is known not to be essential for termination. It could also be that the domain of application dictates that certain individuals cannot be equal. For example, a subset of the ABox individuals may be assumed to be uniquely named.

Let us assume it is possible to specify a *finite* set S of individual terms which we want to exclude from blocking or know their blocking is not essential. Consider the following variation of the (ub) rule.

$$(\mathsf{ub}_{\mathsf{noS}})\colon \frac{s : \{s\},\ t : \{t\}}{s \approx t \mid s : \neg\{t\}}(t \notin S, s \neq t)$$

In contrast to the unrestricted blocking rule, the rule is applicable to a pair of distinct terms s and t if *at least one* of them does not belong to S. In other words, the rule is not applied to two terms if *both* belong to S. This means the rule is not symmetric with respect to s and t, but this is not essential. One can however consider a symmetric variant of the rule where s and t are both required to be outside of S. Although the application of the symmetric rule is even more restricted, Theorem 8 below remains true.

Let $Tab_{\mathcal{SHOI}}(\mathrm{ub}_{\mathrm{noS}})$ be the calculus consisting of all the rules of $Tab_{\mathcal{SHOI}}$ and the $(\mathrm{ub}_{\mathrm{noS}})$ rule.

Theorem 8. *Let S be a finite set of individual terms. Then* Tab$_{\mathcal{SHOI}}(\mathrm{ub}_{\mathrm{noS}})$ *is sound, complete and terminating for* $\mathcal{SHOI}$*.*

Replacing the (ub) rule with the $(\mathrm{ub}_{\mathrm{noS}})$ rule, the calculus remains sound and complete, since the $(\mathrm{ub}_{\mathrm{noS}})$ rule is a sound rule. However, preservation of termination needs to be formally proved. This can be done by showing that there exists a *finite open* branch for any satisfiable concept C when constructing the complete tableau using $Tab_{\mathcal{SHOI}}(\mathrm{ub}_{\mathrm{noS}})$. Since the existence of an open branch is ensured by soundness, we just need to show there is a *finite* open branch. This can be shown by constructing a finite, fully expanded and open branch, with the use of a *model branch* built for the given concept using $Tab_{\mathcal{SHOI}}(\text{ub})$. Guided by the model branch, a finite fully expanded branch for the same concept is constructed by $Tab_{\mathcal{SHOI}}(\text{ub}_{\text{noS}})$. During the construction, an association function is used to limit the possible selection of branches to the ones that mimic the model branch. The association function is formed using the instances of the blocking rule which are no longer applicable. The complete proof of a generic variant of this theorem for arbitrary description logics is presented in [13].

Different variations of the (ub_{noS}) rule can be introduced based on how S is chosen. Possible criteria for choosing members of S are syntactic criteria, for example, individual terms that are not used as labels for any $\exists$-expressions.

Also, the (ub_{noS}) rule can be used for reasoning modulo an implicit unique name assumption for a finite subset of the individuals. This is expected to be more efficient than adding explicit inequality assertions to the input set to ensure unique name assumption, which may cause a drop in performance by increasing the overhead for premise selection. Let S_i be a finite set of individual terms which are assumed to be uniquely named. For each set S_i, an instance of the (ub_{noS}) rule should be introduced. An ontology which contains national identification numbers of people as well as student identification numbers, is a good example for this case. None of the national identification numbers (represented by individuals) should be identifiable, equally no student identification numbers should refer to the same person. But a national identification number and a student identification number can refer to the same person.

In our setting, ABox individuals are not excluded from being blocked as in many description logic tableau systems and the blocking rule is applicable to the pairs of ABox individuals. So, we may form a set S using all the ABox individuals. Then, similar to [8], no terms from S are identified which were not created during the derivation. For this case individuals in S need to be specified to be smallest with respect to the reduction ordering $\prec$ and this instance of the (ub_{noS}) rule needs to be used.

$$(\mathsf{ub}_{\mathsf{noABox}}){:}\ \frac{s:\{s\},\ t:\{t\}}{s\approx t \mid s:\neg\{t\}}\ (t \text{ is not an ABox individual },\ s\neq t)$$

5 Refined Tableau Calculus

In this section we refine the calculus $Tab_{\mathcal{SHOI}}$ presented in Section 3. The idea of the refinement is that the (TBox) rule is replaced by dynamically generated and refined tableau rules.

In the first step, all the atomic concepts in the TBox $\mathcal{T}$ are equi-satisfiably replaced by constant concepts and the parametric (TBox) rule is represented as a set of tableau rules for each $C \sqsubseteq D \in \mathcal{T}$. That is, rather than one rule schema for all statements, a set of rules, one for each statement, is present in the calculus. The following rule is generated for each statement $C \sqsubseteq D$ from the TBox $\mathcal{T}$.

$$\frac{s:\{s\}}{s:\neg C \mid s:D}$$

Subsequently, this rule is transformed to an equivalent rule where the disjunctive normal forms of the negation normal forms of $\neg C$ and D are split into branches of the rule. For example, for the TBox statement $\mathsf{Horse} \sqcap \mathsf{Baby} \sqsubseteq \mathsf{Foal}$, the following rule is obtained.

$$\frac{s : \{s\}}{s : \neg\mathsf{Horse} \mid s : \neg\mathsf{Baby} \mid s : \mathsf{Foal}}$$

Notice that all the atomic concepts in the generated rules are constants and they can only match with themselves. The benefit of such a replacement of the (TBox) rule by a set of rules is the possibility of refining the rules. This allows to reduce the branching factor of the rules, while preserving soundness and (constructive) completeness, by using *atomic rule refinement* introduced in [20]. Atomic rule refinement is a special case of general rule refinement which was introduced in [18]. Under the atomic rule refinement, all conclusions of a rule that are of the form $s : \neg A$, where A is an atomic concept or a singleton, are moved to the premise of the rule as $s : A$. For example, the rule for the TBox statement $\mathsf{Horse} \sqcap \mathsf{Baby} \sqsubseteq \mathsf{Foal}$ is refined to the following rule.

$$\frac{s : \mathsf{Horse},\ s : \mathsf{Baby}}{s : \mathsf{Foal}}$$

In the second step, we apply atomic rule refinement to all the rules obtained from the TBox statements. Consequently there are fewer branches in the conclusions and additional premises are added that limit the application of the rules. (Similar refinements on instances of the (RBox) rule are possible for more expressive logics with negated role assertions.)

Let $Tab^{\mathsf{dyn},\mathcal{T}}_{\mathcal{SHOI}}(\mathrm{ub})$ denote the calculus which consists of the refined generated tableau rules from the TBox $\mathcal{T}$ and all rules of $Tab_{\mathcal{SHOI}}$ except the (TBox) rule. That is, for each statement $C \sqsubseteq D \in \mathcal{T}$ a corresponding tableau rule is generated and refined according to atomic rule refinement. Soundness and completeness of $Tab^{\mathsf{dyn},\mathcal{T}}_{\mathcal{SHOI}}(\mathrm{ub})$ is a direct consequence of the results in [20].

Theorem 9. $Tab^{\mathsf{dyn},\mathcal{T}}_{\mathcal{SHOI}}(\mathrm{ub})$ *is sound, constructively complete and terminating tableau calculus for reasoning in* $\mathcal{SHOI}$ *with respect to a knowledge base* $(\mathcal{A}, \mathcal{T}, \mathcal{R})$ *with a fixed TBox* $\mathcal{T}$.

6 Implementation and Experimental Results

In order to analyse the practical benefit of atomic rule refinement and the $(\mathrm{ub}_{\mathrm{noS}})$ rule, two experiments were designed. MetTeL version 2.0-487 was used to generate provers based on variants of $Tab_{\mathcal{SHOI}}(\mathrm{ub})$ and $Tab^{\mathsf{dyn},\mathcal{T}}_{\mathcal{SHOI}}(\mathrm{ub})$ for various ontologies. MetTeL generates Java code for a tableau prover from the specification of the syntax of a logic and the specification of a tableau calculus.[1]

[1] More information about how to generate a tableau prover using MetTeL is available in [21].

By default, the tableau provers generated with MetTeL use a depth-first left-to-right search strategy. While specifying the specification of tableau calculus, appropriate rule priorities were assigned to ensure the fairness of the expansion strategy and hence guarantee termination. The generated provers were used with no modification in this experiment. For simplicity of implementation, instead of the propagation rules (tr) and (tr^{-}) standard transitivity rules were used in the calculi. We indicate the variations by the superscript $^{+}$.

In order to embrace an extensive range of problems with varying input sizes and expressivity, the experiment used the TONES ontology repository [23] and the corpus of OWL DL ontologies from [15]. The complete repositories of 874 ontologies were downloaded. A translator using the OWL API [7] was developed to prepare appropriate input for MetTeL. Each ontology was converted into three forms with the translator. The first form provided input to Fact++ [24] which was then used to validate the translation and outputs of the provers. The second form were translations of the ontology so that we could check its consistency with a prover generated by MetTeL using $Tab^{+}_{\mathcal{SHOI}}(\mathrm{ub})$ as the tableau specification. The third form was used in two ways. First, it was used to produce a tableau specification for $Tab^{+,\mathsf{dyn},\mathcal{T}}_{\mathcal{SHOI}}(\mathrm{ub})$ containing the dynamic rules generated from the ontology. Second, the remaining ontology axioms were translated so that the prover generated using the specification of $Tab^{+,\mathsf{dyn},\mathcal{T}}_{\mathcal{SHOI}}(\mathrm{ub})$ could check its consistency. Inputs prepared for both provers were then used with a results file from Fact++ to produce additional problem sets. One of the results files produced by Fact++ contains the class hierarchy of the ontology. For a randomly picked subsumption relation $C \sqsubseteq D$ in the hierarchy and a fresh individual s, $s : C$ and $s : D$ were added to the input file to form an additional satisfiable input, and respectively $s : C$ and $s : \neg D$ were added to form an additional unsatisfiable input. This experiment was aimed at evaluating the effect on reasoning performance when using $Tab^{+,\mathsf{dyn},\mathcal{T}}_{\mathcal{SHOI}}(\mathrm{ub})$ in comparison to $Tab^{+}_{\mathcal{SHOI}}(\mathrm{ub})$. This means we checked the consistency of the input but omitted checking satisfiability of all concepts and calculating concept hierarchies.

The developed translator successfully translated 628 ontologies and each prover was executed on 2480 inputs with a timeout of 100 seconds. The comparison was done by measuring the execution time of the prover. The results of the comparison of $Tab^{+}_{\mathcal{SHOI}}(\mathrm{ub})$ and $Tab^{+,\mathsf{dyn},\mathcal{T}}_{\mathcal{SHOI}}(\mathrm{ub})$ are presented in Table 1. For the set of results *with timeout*, when a prover did not return any answer within 100 seconds, 100 seconds were used in the calculation of the average in time. While for the set of results *without timeout*, if one of the provers under comparison required more than 100 seconds, that input is not included in the results. The results show that the generated provers based on the refined tableau calculus were faster for unsatisfiable inputs. Inspection showed this was mainly a consequence of having additional closure rules. These closure rules were refinements of dynamically generated rules from TBox statements where all the conclusions have been turned into premises in a rule. The scatter plot on the left of Figure 2 gives a more differentiated picture of the performance. On average we observed a 22% drop in memory use for satisfiable inputs and a 74% drop for

Table 1. Average run times in seconds for $Tab^{+}_{\mathcal{SHOI}}(\text{ub})$ and $Tab^{+,\mathsf{dyn},\mathcal{T}}_{\mathcal{SHOI}}(\text{ub})$

	With timeout			Without timeout		
Input	count	$Tab^{+}_{\mathcal{SHOI}}(\text{ub})$	$Tab^{+,\mathsf{dyn},\mathcal{T}}_{\mathcal{SHOI}}(\text{ub})$	count	$Tab^{+}_{\mathcal{SHOI}}(\text{ub})$	$Tab^{+,\mathsf{dyn},\mathcal{T}}_{\mathcal{SHOI}}(\text{ub})$
Ontology consistency	628	**27.627**	43.094	346	**0.951**	**1.049**
Satisfiable inputs	924	**60.847**	65.999	180	13.447	**0.869**
Unsatisfiable inputs	928	21.521	**3.643**	760	5.053	**1.841**

Table 2. Average run times in seconds for $Tab^{+,\mathsf{dyn},\mathcal{T}}_{\mathcal{SHOI}}(\text{ub})$ and $Tab^{+,\mathsf{dyn},\mathcal{T}}_{\mathcal{SHOI}}(\text{ub}_{\text{noABox}})$

	With timeout			Without timeout		
Input	count	(ub)	$(\text{ub}_{\text{noABox}})$	count	(ub)	$(\text{ub}_{\text{noABox}})$
Ontology consistency	628	43.094	**35.893**	346	**1.049**	**1.025**
Satisfiable inputs	924	65.999	**56.300**	180	0.869	**0.661**
Unsatisfiable inputs	928	**3.643**	**3.635**	760	**1.841**	**1.832**

unsatisfiable inputs when using $Tab^{+,\mathsf{dyn},\mathcal{T}}_{\mathcal{SHOI}}(\text{ub})$ in comparison to $Tab^{+}_{\mathcal{SHOI}}(\text{ub})$. As expected, the performance of the systems were not comparable with FACT++.

Moreover, an experiment to compare the performance of $Tab^{+,\mathsf{dyn},\mathcal{T}}_{\mathcal{SHOI}}(\text{ub})$ and $Tab^{+,\mathsf{dyn},\mathcal{T}}_{\mathcal{SHOI}}(\text{ub}_{\text{noABox}})$, using the same inputs as before, was designed. Since it is not yet possible to express rules such as the (ub_{noS}) rule in the METTEL rule specification language, we generated a prover for the tableau calculus $Tab^{+,\mathsf{dyn},\mathcal{T}}_{\mathcal{SHOI}}$ without any blocking mechanism. Then, code implementing the $(\text{ub}_{\text{noABox}})$ rule

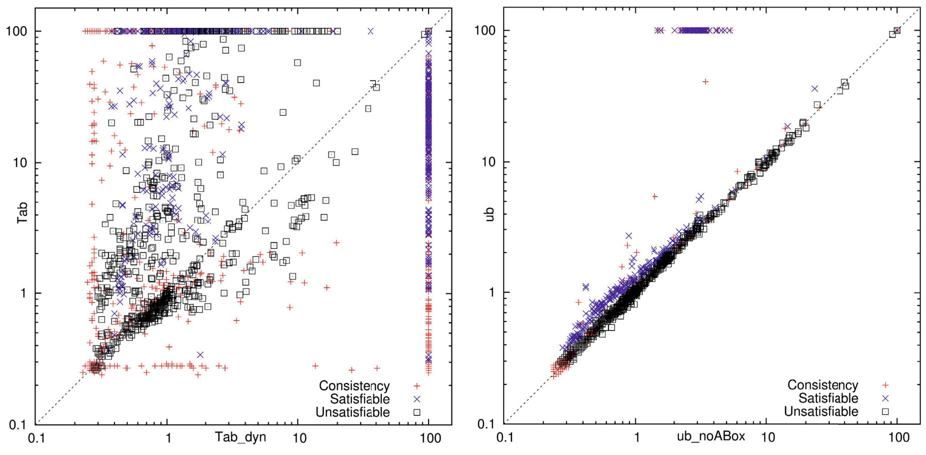


Fig. 2. Scatter plots of $Tab^{+}_{\mathcal{SHOI}}(\text{ub})$ vs. $Tab^{+,\mathsf{dyn},\mathcal{T}}_{\mathcal{SHOI}}(\text{ub})$ and $Tab^{+,\mathsf{dyn},\mathcal{T}}_{\mathcal{SHOI}}(\text{ub})$ vs. $Tab^{+,\mathsf{dyn},\mathcal{T}}_{\mathcal{SHOI}}(\text{ub}_{\text{noABox}})$

was manually added to the generated JAVA code. In order to have a fair comparison, the prover for the (ub) rule was also created by manually adding code implementing the (ub) rule. The results of the comparison are presented in Table 2 and on the right in Figure 2.

The experimental results show using the ($\mathrm{ub}_{\mathrm{noABox}}$) rule had a small benefit in most cases, but there was a group of satisfiable problems not solved using the (ub) rule within the timeout that could be solved under 10 seconds when using the ($\mathrm{ub}_{\mathrm{noABox}}$) rule. A closer analysis of some of the problems suggested this was because they implicitly force the unique name assumption for a large number of ABox individuals.

7 Concluding Remarks

A tableau decision procedure for the description logic $\mathcal{SHOI}$ was presented in this paper. A refined version of the tableau calculus in [12] was presented which uses dynamically generated tableau rules when reasoning with respect to a knowledge base. Following a rule refinement technique in [20] the generated tableau rules were refined leading to a smaller search space. We investigated a controlled variant of the unrestricted blocking rule not applied to members of an a priori defined, finite set. This variant can be utilised for scenarios such as reasoning under unique name assumption.

A comparison was done between the provers generated using the tableau calculus with dynamically generated tableau rules, and a prover with the fixed rule for dealing with TBox statements. The results showed the former is more optimised especially for unsatisfiable inputs. The analysis of the reduction in the branching points and complexity is left as future work.

Other future plans include studying the relationship between properties of a logic and minimally required blocking criteria. That is, expressing side conditions that can be used to control the unrestricted blocking rule to be applied as little as possible. This should be done without endangering termination. Expressing existing blocking mechanisms as variants of unrestricted blocking mechanism, is also one of our future plans. Using these results, we hope to be able to provide uniform explanations and implementations of blocking mechanisms in tableau provers.

References

1. Alenda, R., Olivetti, N., Schwind, C., Tishkovsky, D.: Tableau calculi for $\mathcal{CSL}$ over minspaces. In: Dawar, A., Veith, H. (eds.) CSL 2010. LNCS, vol. 6247, pp. 52–66. Springer, Heidelberg (2010)
2. Baader, F., Sattler, U.: An overview of tableau algorithms for description logics. Studia Logica 69(1), 5–40 (2001)
3. Bolander, T., Blackburn, P.: Termination for hybrid tableaus. J. Logic Comput. 17(3), 517–554 (2007)
4. Cialdea Mayer, M., Cerrito, S.: Nominal substitution at work with the global and converse modalities. In: Proc. AiML-8, pp. 57–74. College Publ. (2010)

5. Duc, C.L., Lamolle, M.: Decidability of description logics with transitive closure of roles in concept and role inclusion axioms. In: Proc. DL 2010. CEUR Workshop Proceedings, vol. 573 (2010)
6. Fitting, M.: Proof methods for modal and intuitionistic logics. Kluwer (1983)
7. Horridge, M., Bechhofer, S.: The OWL API: A Java API for OWL ontologies. Semantic Web 2(1), 11–21 (2011)
8. Horrocks, I., Kutz, O., Sattler, U.: The even more irresistible SROIQ. In: Proc. KR 2006, pp. 57–67. AAAI Press (2006)
9. Horrocks, I., Sattler, U.: A description logic with transitive and inverse roles and role hierarchies. J. Logic Comput. 9(3), 385–410 (1999)
10. Horrocks, I., Sattler, U.: A tableau decision procedure for SHOIQ. J. Automat. Reasoning 39(3), 249–276 (2007)
11. Horrocks, I., Sattler, U., Tobies, S.: Practical reasoning for expressive description logics. In: Ganzinger, H., McAllester, D., Voronkov, A. (eds.) LPAR 1999. LNCS, vol. 1705, pp. 161–180. Springer, Heidelberg (1999)
12. Khodadadi, M., Schmidt, R.A., Tishkovsky, D.: An abstract tableau calculus for the description logic SHOI using unrestricted blocking and rewriting. In: Proc. DL 2012. CEUR Workshop Proceedings, vol. 846, pp. 224–234 (2012)
13. Khodadadi, M., Schmidt, R.A., Tishkovsky, D.: A refined tableau calculus with controlled blocking for the description logic SHOI (2013), `http://www.mettel-prover.org/papers/controlled.pdf`
14. Khodadadi, M., Schmidt, R.A., Tishkovsky, D.: A refined tableau calculus with controlled blocking for the description logic SHOI. To Appear in Proc. DL 2013. CEUR Workshop Proceedings (2013)
15. Matentzoglu, N., Bail, S., Parsia, B.: A corpus of OWL DL ontologies. To Appear in Proc. DL 2013. CEUR Workshop Proceedings (2013)
16. Schmidt, R.A., Tishkovsky, D.: Using tableau to decide expressive description logics with role negation. In: Aberer, K., et al. (eds.) ISWC/ASWC 2007. LNCS, vol. 4825, pp. 438–451. Springer, Heidelberg (2007)
17. Schmidt, R.A., Tishkovsky, D.: A general tableau method for deciding description logics, modal logics and related first-order fragments. In: Armando, A., Baumgartner, P., Dowek, G. (eds.) IJCAR 2008. LNCS (LNAI), vol. 5195, pp. 194–209. Springer, Heidelberg (2008)
18. Schmidt, R.A., Tishkovsky, D.: Automated synthesis of tableau calculi. Logical Methods in Comput. Sci. 7(2), 1–32 (2011)
19. Schmidt, R.A., Tishkovsky, D.: Using tableau to decide description logics with full role negation and identity. arXiv e-Print, abs/1208.1476 (2012)
20. Tishkovsky, D., Schmidt, R.A.: Refinement in the tableau synthesis framework. arXiv e-Print, abs/1305.3131 (2013)
21. Tishkovsky, D., Schmidt, R.A., Khodadadi, M.: Mettel2: Towards a tableau prover generation platform. In: Proc. PAAR 2012. EasyChair Proceedings (2012)
22. Tishkovsky, D., Schmidt, R.A., Khodadadi, M.: The tableau prover generator MetTeL2. In: del Cerro, L.F., Herzig, A., Mengin, J. (eds.) JELIA 2012. LNCS, vol. 7519, pp. 492–495. Springer, Heidelberg (2012)
23. TONES. The tones ontology repository (March 5, 2013)
24. Tsarkov, D., Horrocks, I.: FaCT++ description logic reasoner: System description. In: Furbach, U., Shankar, N. (eds.) IJCAR 2006. LNCS (LNAI), vol. 4130, pp. 292–297. Springer, Heidelberg (2006)

Prefixed Tableau Systems for Logic of Proofs and Provability

Hidenori Kurokawa

Kobe University, Department of Information Science
hidenori.kurokawa@gmail.com

Abstract. In this paper, we introduce prefixed tableau systems for logics combining Artemov's logic of proofs, which is introduced in order to explore combinatorial structure of proofs, and the logic of provability (strong provability), which has been studied as a logic of formal provability (provability and truth) in arithmetic for decades. Such joint logics have already been studied, but no cut-free tableau systems for these logics have been available in the literature so far. We show the admissibility of cut for these systems via semantic completeness for cut-free prefixed tableau systems for these logics.

Keywords: prefixed tableau system, logic of proofs, provability logic.

1 Introduction

Gödel-Löb logic (GL) has been the fundamental modal logic in the area of provability logic (see, [5]). On the other hand, the logic of proofs (LP, aka justification logic) was introduced (in [2]) as an explicit modal logic to study the structure of 'proofs' at the level of propositional logic. Yet another logic has been studied to capture the notion of "being provable in PA and true" (called "strong provability"). The modal logic for strong provability is known as Grz [5].

Logics that combine GL (Grz) and LP have already been introduced, and their arithmetic interpretations ([15], [3], [13]) have been studied. Fitting-style semantics for these logics have also been studied under the names GLA and GrzA ([13], [14]). By combining GL and LP, we can observe how the notion of formal provability in PA and the notion of proofs in PA interact. A good illustration of this is one axiom in GLA, i.e., $\neg t : \varphi \to \Box\neg t : \varphi$ (we call this "mixed negative introspection"). This formula is of interest, not only because it is a kind of negative introspection, which is an analogue of the axiom $\neg\Box\varphi \to \Box\neg\Box\varphi$ in S5 but also because this statement is valid in arithmetic interpretations. More generally, GLA may be of interest because we can express some formula, e.g., $\neg t : \Box\bot$, whose arithmetical interpretation may be interesting and discuss their provability in a modal propositional logic.

In this paper, we introduce prefixed tableau systems for both GrzA and GLA and show cut-admissibility of these systems. (Note that the only proof systems of these logics currently available are Hilbert-style systems.) We claim that cut-free systems for these logics are desirable. This is because a cut-free proof may give a more perspicuous view about the meaning of a proven sentence in the way that so-called "proof-theoretic semantics" talks about the meaning of a sentence.

We use prefixed tableau systems primarily because the mixed negative introspection in these logics makes it difficult to formulate cut-free destructive tableau systems for the logics. In addition to this reason which is analogous to S5, there are other reasons why semantically proving cut-admissibility of the prefixed tableau systems for these logics is significantly less trivial than one might initially expect. First, we need to satisfy both the closure conditions for an item in the semantics called an"evidence function" in order to handle proof terms in LP and the frame condition for GL (Grz). This apparently requires us to once use an infinitary construction of maximal consistent set and to somehow make some feature of it "finite." Second, for GLA, a new rule called "reflection rule" also needs a special care.[1]

2 Hilbert-Style Systems and Kripke-Fitting Models

The language of GLA ($\mathcal{L}_{GLA}$) can be specified as follows:

1. The class of proof terms (Trm) in $\mathcal{L}_{GLA}$ is specified as $t := x|a|!t|t_1 \cdot t_2|t_1 + t_2$

In proof-terms, x is a proof variable, a is a proof constant, and we have proof operators: !(proof-checker), $\cdot$ (proof application), + (proof sum).[2]

2. The class of formulas (Fmla) in $\mathcal{L}_{GLA}$ is specified as follows:

$A := p_i|\bot|\neg A_1|A_1 \to A_2|A_1 \wedge A_2|A_1 \vee A_2|t{:}A|\Box A$

Hilbert style system of GLA is given as follows.

I. Axioms: 0) Axioms of Propositional Logic
1) Axioms of LP: 1. $t{:}(\varphi \to \psi) \to (s{:}\varphi \to t \cdot s{:}\psi)$
2. $t{:}\varphi \to \varphi$ 3. $t{:}\varphi \to !t{:}t{:}\varphi$ 4. $t{:}\varphi \to t + s{:}\varphi$, $s{:}\varphi \to t + s{:}\varphi$

2) Axioms of GL: 1. $\Box(\varphi \to \psi) \to (\Box\varphi \to \Box\psi)$ 2. $\Box\varphi \to \Box\Box\varphi$ 3. $\Box(\Box\varphi \to \varphi) \to \Box\varphi$

3) Connecting Axioms: 1. $t{:}\varphi \to \Box\varphi$ 2.$\neg t{:}\varphi \to \Box\neg t{:}\varphi$ 3. $t{:}\Box\varphi \to \varphi$

II. Rules of Inference: 1. Modus Ponens 2. $\dfrac{\varphi}{\Box\varphi}$ 3. Reflection Rule $\dfrac{\Box\varphi}{\varphi}$

4. Axiom necessitation $c{:}A$, for $A \in \mathcal{CS}(c)$, where $\mathcal{CS}$ is a constant specification (c.s.), i.e., a function which maps a proof constant to the set of axioms in GLA.

Note: (1) For a reason to be explained later, we define a subsystem of GLA ("wGLA") by removing the connecting axiom 3. $t{:}\Box\varphi \to \varphi$ and reflection rule from GLA.

[1] For modal characteristic rules for GL and Grz, we use extant rules presented in [9].
[2] Note that there are countably many proof variables and proof constants in the language.

(2) We could formulate $\mathsf{GLA}_{\emptyset}$ (GLA with the empty constant specification) and define $\mathsf{GLA}_{\mathcal{CS}}$ to be GLA with a specific constant specification. However, in this paper, we do not discuss topics in which subtle issues about constant specifications matter. Thus, we have integrated the full constant specification to the system GLA.

We now define Kripke-Fitting semantics for wGLA.[3] Let a quadruple (K, R, R^e, r) be a frame, where K is non-empty set. Let R be binary relation over K satisfying the following conditions : (1) R is a strict partial order; (2) R has a single root r; (3) R has no infinite ascending chain. R^e is a reflexive, symmetric and transitive relation. We also require $R \subseteq R^e$.[4] Let $\mathcal{E}$ be an evidence function (a function which assigns to a state and a term an "evidence" that is a set of formulas): $K \times Trm \longrightarrow \mathcal{P}(Fmla)$ that satisfies the following properties ($\mathcal{CS}$ is a constant specification for wGLA):

1. $uR^e v$ implies $\mathcal{E}(u,t) \subseteq \mathcal{E}(v,t)$ (Monotonicity)[5];
2. $F \to G \in \mathcal{E}(u,t)$ and $F \in \mathcal{E}(u,s)$ implies $G \in \mathcal{E}(u, t\cdot s)$;
3. $F \in \mathcal{E}(u,t)$ implies $t{:}\ F \in \mathcal{E}(u, !t)$; 4. $\mathcal{E}(u,s) \cup \mathcal{E}(u,t) \subseteq \mathcal{E}(u, s+t)$;
5. $\mathcal{CS}(c) \subseteq \mathcal{E}(u,c)$.

We define a wGLA model as a sextuple $(K, R, R^e, r, \mathcal{E}, \Vdash)$, where r is the root node of the relation R and $\Vdash$ is a forcing relation such that

1. $\Vdash$ commutes with Booleans at each state; for all $u \in K$, $u \nVdash \bot$.
2. $u \Vdash \Box\varphi$ iff for every $v \in K$, s.t. uRv, $v \Vdash \varphi$
3. $u \Vdash t{:}\varphi$ iff $\varphi \in \mathcal{E}(u,t)$ and for every $v \in K$, s.t. $uR^e v$, $v \Vdash \varphi$.
4. $A \in \mathcal{CS}(c)$ implies $\mathcal{K}, u \Vdash c{:}A$ for every $u \in K$.

We have one additional condition for a model of GLA (w.r.t. a formula φ).

Root Soundness: Let r be the root of a tree wGLA model for φ. Then consider the following condition : $r \Vdash Rf(Sb(\varphi))$ for a given formula φ, where $Rf(Sb(\varphi)) = \{\Box A \to A | \Box A \in Sb(\varphi)\}$. ($Sb(\varphi)$ stands for the set of subformulas of φ and Rf means "root formulas.") We call a model that satisfies this condition "a φ-sound GLA model" with $\mathcal{CS}$, usually suppressing $\mathcal{CS}$, for a specific formula φ.

Note: Only wGLA axioms and rules are sound with respect to the class of wGLA. The axiom $t : \Box\varphi \to \varphi$ and reflection rule $\Box\varphi/\varphi$ are sound only with respect to $\Box\varphi$-sound GLA models. This terminology seems to be inevitable, since $\Box\varphi$-soundness depends on a particular formula $\Box\varphi$.

Also, φ is said to be *valid in a model* $\mathcal{K}$ ($\mathcal{K} \Vdash \varphi$), when for any $u \in K$, $u \Vdash \varphi$.

[3] "$\Longrightarrow$", "$\forall$" and "$\exists$" are abbreviations of "if ..., then ...", "for all" and "there exists."

[4] R^e could be treated as a universal relation ([13]). But for a technical reason, we use R^e.

[5] By symmetry of $uR^e v$, 1 implies $\mathcal{E}(u,t) = \mathcal{E}(v,t)$. We call this *stability.*

We now move on to GrzA. The language of GrzA can be specified in a manner similar to that of GLA. The only difference is that we have $\boxdot A$ instead of $\Box A$.

A Hilbert style system of GrzA is as follows.

I. Axioms: 0) Axioms of Propositional Logic 1) Axioms of LP: same as GLA
2) Axioms of Grz: 1. $\boxdot(\varphi \to \psi) \to (\boxdot\varphi \to \boxdot\psi)$
2. $\boxdot\varphi \to \varphi$ 3. $\boxdot\varphi \to \boxdot\boxdot\varphi$ 4. $\boxdot(\boxdot(\varphi \to \boxdot\varphi) \to \varphi) \to \varphi$

3) Connecting Axioms: 1. $t{:}\varphi \to \boxdot\varphi$ 2. $\neg t{:}\varphi \to \boxdot\neg t{:}\varphi$

II. Rules of Inference: 1. Modus Ponens 2. Necessitation $\dfrac{\varphi}{\boxdot\varphi}$

3. Axiom necessitation $c{:}A$ for $A \in \mathcal{CS}(c)$ ($\mathcal{CS}$ is a c.s. for GrzA).

A Kripke-Fitting model $\mathcal{K}^s(= (K^s, R^s, R^e, r, \mathcal{E}, \Vdash))$ for GrzA is given as follows.

1. R^s is a non-strict partial order (with a single root) that has no strictly ascending infinite chain. (Note the redundancy of the root soundness.)

2. Other conditions for Kripke-Fitting models for GLA and GrzA are the same.

3 Prefixed Tableau System for GrzA

For the sake of expository convenience, we first introduce a prefixed tableaux system for GrzA called "TGrzA" and we prove soundness and completeness of TGrzA.

Here is the prefixed tableau system TGrzA. We follow Fitting's terminology for basic notions in the prefixed tableau system (e.g., [6]). $\sigma T\varphi$ or $\sigma F\varphi$ is called "a prefixed signed formula," and $T\varphi$ or $F\varphi$ is called "a signed formula," where T and F are intended to mean "true" and "false" respectively, and a prefix σ is a sequence of positive integers, intended to represent a possible world in a Kripke-Fitting model. A branch is closed if (1) $\sigma T\varphi$ and $\sigma F\varphi$ are on it or (2) some specific closure condition is satisfied in a particular tableau system (e.g., see $\bot$-rule or $\mathcal{CS}$ rules below). A tableau is closed if all branches of it are closed. We say that a formula φ has a tableau proof in TGrzA if $1F\varphi$ has a closed tableau in TGrzA. We say σ is *used* on a branch if a prefix that has σ as its (not necessarily proper) initial segment has already occurred on the branch, and we say σ is new, otherwise.

Classical propositional rules α-rule, β-rule and $\bot$-rule (see [6]).

α-rule: $\dfrac{\sigma T\varphi \wedge \psi}{\begin{array}{c}\sigma T\varphi\\ \sigma T\psi\end{array}}$ $\dfrac{\sigma F\varphi \vee \psi}{\begin{array}{c}\sigma F\varphi\\ \sigma F\psi\end{array}}$ $\dfrac{\sigma F\varphi \to \psi}{\begin{array}{c}\sigma T\varphi\\ \sigma F\psi\end{array}}$ $\dfrac{\sigma T\neg\varphi}{\sigma F\varphi}$ $\dfrac{\sigma F\neg\varphi}{\sigma T\varphi}$

β-rule: $\dfrac{\sigma T\varphi \vee \psi}{\sigma T\varphi \mid \sigma T\psi}$ $\dfrac{\sigma F\varphi \wedge \psi}{\sigma F\varphi \mid \sigma F\psi}$ $\dfrac{\sigma T\varphi \to \psi}{\sigma F\varphi \mid \sigma T\psi}$

$\bot$-rule: A branch is closed if $\sigma T\bot$ occurs on it.

Modal Rules: ν-rules: $\nu_K \dfrac{\sigma T\boxdot\varphi}{\sigma.nT\varphi}$ $\nu_4 \dfrac{\sigma T\boxdot\varphi}{\sigma.nT\boxdot\varphi}$ ($\sigma.n$ is used for ν_K, ν_4.) $\nu_T \dfrac{\sigma T\boxdot\varphi}{\sigma T\varphi}$

π-rule for Grz: $\dfrac{\sigma F \boxdot \varphi}{\begin{array}{c}\sigma.nF\varphi\\ \sigma.nT \boxdot (\varphi \to \boxdot\varphi)\end{array}}$ ($\sigma.n$ is new.)

Rules for LP: (ν-rules for explicit proofs)

EK $\dfrac{\sigma Tt{:}\varphi}{\sigma.nT\varphi}$ ($\sigma.n$ is used.) ET $\dfrac{\sigma Tt{:}\varphi}{\sigma T\varphi}$ E4 $\dfrac{\sigma Tt{:}\varphi}{\sigma.nTt{:}\varphi}$ ($\sigma.n$ is used.)

E4r $\dfrac{\sigma.nTt{:}\varphi}{\sigma Tt{:}\varphi}$ EF $\dfrac{\sigma Ft{:}\varphi}{\sigma.nFt{:}\varphi}$ ($\sigma.n$ is used.) EFr $\dfrac{\sigma.nFt{:}\varphi}{\sigma Ft{:}\varphi}$

Operational Rules on F's : $\cdot$-rule $\dfrac{\sigma F(t\cdot s){:}\varphi}{\sigma Ft{:}\psi \to \varphi | \sigma Fs{:}\psi}$

!-rule $\dfrac{\sigma F!t{:}t{:}\varphi}{\sigma Ft{:}\varphi}$ +-rule $\dfrac{\sigma F(s+t){:}\varphi}{\sigma Ft{:}\varphi}$ $\dfrac{\sigma F(t+s){:}\varphi}{\sigma Ft{:}\varphi}$

Constant Specification ($\mathcal{CS}$) Rules: a branch is closed if it has $\sigma Fc{:}A$ ($A \in \mathcal{CS}(c)$).

We give a few further definitions. We write $\sigma \leq \sigma'$ if σ is a (not necessarily proper) initial segment of σ'. A signed formula $F\varphi$, $T\varphi$ is *realized* at a possible world u of a model $\mathcal{K}^s$ if 1) the formula is $T\varphi$ and $\mathcal{K}^s, u \Vdash \varphi$ or 2) the formula is $F\varphi$ and $\mathcal{K}^s, u \nVdash \varphi$. Let $\mathcal{N}$ be a GrzA interpretation (partial) function from the set of finite σ-sequences to the underlying set K of a model if, for any $\sigma, \sigma' \in Dom(\mathcal{N})$, (1) $\exists u$ ($\mathcal{N}(\sigma)R^s u$) and $\sigma \leq \sigma' \Longrightarrow \mathcal{N}(\sigma)R^s\mathcal{N}(\sigma')$ and (2) $\exists u$ ($\mathcal{N}(\sigma)R^e u) \Longrightarrow \mathcal{N}(\sigma)R^e\mathcal{N}(\sigma')$.

Let Φ be a signed formula ($T\varphi$ or $F\varphi$). A set S of prefixed signed formula is *satisfiable* if there is a model $\mathcal{K}^s$ and a mapping $\mathcal{N}$ from the prefixes in S to possible worlds in $\mathcal{K}^s$, such that if $\sigma\Phi \in S$, then Φ is realized at $\mathcal{N}(\sigma)$ in $\mathcal{K}^s$. (Similarly for a branch and a tableau.) Now we prove soundness of the prefixed tableau system.

Lemma 1. *Suppose $\mathcal{T}$ is a satisfiable tableau. If any tableau rule for GrzA is applied to $\mathcal{T}$, then the resulting tableau is still satisfiable.*

Proof. Suppose a tableau is GrzA-satisfiable because a branch θ of $\mathcal{T}$ is GrzA-satisfiable, i.e. its members are realized at $\mathcal{N}(\sigma)$ of model $\mathcal{K}^s$. Suppose that a tableau rule for GrzA is applied to the tableau $\mathcal{T}$. The entire proof is divided into two cases. Case 1: Our tableau rule is not applied on the branch θ. Then, θ is still present on the new tableau, which makes θ obviously GrzA-satisfiable.

Case 2: The tableau rule is applied on θ. LP-rules, ν rules for $\boxdot$ in GrzA are similar to the cases of S4LPN in [12]. The proof of the case of π-rule is quite intriguing and not necessarily a routine, but it is excessively long. Thus, we omit it here. ⊠

Theorem 1 (Soundness).
If φ has a prefixed GrzA-tableau proof, then φ is valid in all models for GrzA.

Proof. The standard reductio argument showing the soundness of a tableau system via the above lemma. ⊠

Now we move on to the completeness theorem. Let us fix a formula unprovable in GrzA. We construct a countermodel for the formula. Let us give an outline of a construction of a countermodel for the unprovable formula φ in GrzA.

Step 1. We carry out Lindenbaum-Henkin construction for GrzA.

Step 2. We verify that the constructed canonical (pseudo-)model satisfies the conditions of evidence function and forcing relation (Truth Lemma). However, our canonical (pseudo-)model does not satisfy the frame condition that there is no infinite strictly ascending chain. To satisfy the condition, we take further steps.

Step 3. We define the notion of bounded bisimulation and prove a proposition that for any formula of degree n, n-bisimilar states of the two models are equivalent.

Step 4. By using this, we construct a finite height model from the canonical model $\mathcal{K}^s$ such that the height is less than or equal to the degree of the formula φ. We show that the constructed n-height model is indeed a countermodel for φ.

Step 1: We start discussing Lindenbaum-Henkin construction ([7]). We first give some definitions. A set S of prefixed formulas is GrzA-consistent if no GrzA-tableau for a finite part of S is closed. S is maximally GrzA-consistent if S is GrzA-consistent and no S', s.t. $S \subsetneq S'$ is GrzA-consistent. S is π-complete (or $F \boxdot \varphi$-complete) provided, if $\sigma F \boxdot \varphi \in S$, then for some integer k, $\sigma.kF\varphi \in S$. S omits infinitely many integers if the set of integers that do not appear in prefixes in S is infinite.

Let us now give Lindenbaum-Henkin construction. Enumerate all formulas $\sigma_i \Phi_i$ in the language of GrzA. Then construct S_n (for each $n \in \mathbb{N}$) as follows.

$S_0 = \{1F\varphi\}$;

$$S_{n+1} = \begin{cases} S_n \cup \{\sigma_n \Psi_n\} \text{ if this is consistent and } \Psi_n \text{ is not } F \boxdot \psi_n; \\ S_n \cup \{\sigma_n \pi\} \cup \{\sigma_n.k\pi_0\} \cup \{\sigma_n.kT \boxdot (\psi_n \to \boxdot\psi_n)\} \\ \qquad \text{if } S_n \cup \{\sigma_n \Psi_n\} \text{ is consistent, } \Psi_n = \pi = F \boxdot \psi_n \text{ and } \pi_0 = F\psi_n \\ \qquad \text{and } \sigma_n.k \text{ is new } (= \text{does not occur in } S_n \text{ or in } \sigma_n\pi); \\ S_n \text{ otherwise.} \end{cases}$$

Let $S_\omega = \bigcup_n S_n$. The construction is similar the one for the modal logic K in [7] except that we use GrzA-consistency. We state the following claims and lemmas.

Claim. S_n omits infinitely many integers.
Claim. If $S_n \cup \{\sigma_n \pi\}$ is GrzA-consistent, then so is $S_n \cup \{\sigma_n\pi, \sigma_n.k\pi_0, \sigma_n.kT \boxdot (\psi_n \to \boxdot\psi_n)\}$, provided that $\sigma_n.k$ is new.
Claim. If S_n omits infinitely many integers, there will be a new prefix to be used in constructing S_{n+1}.

Lemma 2. *If $\{1F\varphi\}$ is GrzA-consistent, then S_ω will be maximally GrzA-consistent and π-complete.*

Proof. The argument is similar to the standard one used in Henkin-construction for proving completeness of first-order logic. ⊠

Now we construct a canonical Kripke-Fitting model $\mathcal{K}^s = (K^s, R^s, R^e, r, \mathcal{E}, \Vdash)$ for GrzA based on this maximal GrzA-consistent set. Let K^s be $\{\sigma | \sigma\Phi \in S_\omega\}$. We identify sequences with possible worlds (so the interpretation function will be the identity function). The accessibility relations R^s and R^e, the root node r, forcing relation $\Vdash$ (for atomic p) and evidence function $\mathcal{E}$ are given as follows: 1. $\sigma R^s \sigma'$ iff σ is a (not necessarily proper) initial segment of σ' ($\sigma \leq \sigma'$); 2. $\sigma R^e \sigma'$ iff $1 \leq \sigma$ and $1 \leq \sigma'$; 3. $r = 1$; 4. $\psi \in \mathcal{E}(\sigma, t)$ iff $\sigma F t{:}\psi \notin S_\omega$; 5. $\sigma \Vdash p$ iff $\sigma T p \in S_\omega$; $\sigma \Vdash \bot$ iff $\sigma T \bot \in S_\omega$.

Step 2: We have to verify that the R^s, R^e, $\Vdash$ and $\mathcal{E}$ all satisfy the conditions of a Kripke-Fitting model for GrzA. It is obvious that $\leq$ is reflexive and transitive. We can show that it is anti-symmetric as follows: suppose $\sigma \leq \sigma'$ and $\sigma' \leq \sigma$. By the definition of non-proper initial segment of a sequence of positive integers, this implies the following disjunction ($\sigma < \sigma' \wedge \sigma' < \sigma$) or ($\sigma < \sigma' \wedge \sigma' = \sigma$) or ($\sigma' < \sigma \wedge \sigma = \sigma'$) or ($\sigma = \sigma' \wedge \sigma' = \sigma$). The first three cases imply $\sigma < \sigma$, a contradiction. So, $\sigma = \sigma'$ holds. It follows from the definition that $1 \leq \sigma$ and $1 \leq \sigma'$ is an equivalence relation. Also, obviously, $R^s \subseteq R^e$. In this step, we have to verify the two additional features:

1. $\mathcal{E}$ satisfies the conditions of an evidence function;
2. $\Vdash$ can be extended to the entire language of GrzA (Truth Lemma);

We now state these two items, but the proofs for the next few propositions are omitted, since the proofs are similar to those of the S4LPN in [12].

Proposition 1. *1. $\sigma F t{:}\psi \in S_\omega$ if and only if $\sigma' F t{:}\psi \in S_\omega$ (for any σ, σ' in K^s).*
2. $\sigma T t{:}\psi \in S_\omega$ if and only if $\sigma' T t : \psi \in S_\omega$ (for any σ, σ' in K^s).

Corollary 1. *Let $\mathcal{K}^s$ be the canonical model we have constructed.*
1. For all $\sigma_1, \sigma_2 \in K^s$, $\sigma_1 R^e \sigma_2$ (i.e., $R^e = R^s \times R^s$).
2. For any $\sigma \in K^s$, ($\sigma \Vdash t{:}\varphi$) or for any $\sigma \in K^s$, ($\sigma \nVdash t{:}\varphi$).

The constructed $\mathcal{E}$ satisfies the conditions of an evidence function.

Proposition 2. *The evidence function defined above satisfies the following conditions: (1) monotonicity, (2) closure conditions, (3) constant specification.*

Then we can prove the crucial lemma. (Currently, $\mathcal{N}$ is the identity function.)

Lemma 3 (Truth Lemma).
$\sigma\Psi \in S_\omega \Longrightarrow \Psi$ is realized at σ in $\mathcal{K}^s$.

Proof. Similar to the case of S4LPN ([12]).

Step 3: Let $\mathcal{K}^s = (K^s, R^s, R^e, r, \mathcal{E}, \Vdash)$ be the canonical "model" for GrzA constructed by the Lindenbaum-Henkin construction. But this does not yet satisfy the frame condition for a GrzA model. In order to satisfy the condition that a model of GrzA has no infinite strictly ascending chain, we finitize the "height" of the model by using the technique of bounded bisimulation. Here we give some terminologies (following [4] and [8]). The notion of the *height* of states in $\mathcal{K}^s$

is defined by induction. The only element of height 0 is the root of the model; the states of height $n+1$ are those immediate successors of elements of height n that have not yet been assigned a height smaller than $n+1$. The *height of a model* $\mathcal{K}^s$ is the maximum n such that there is a state of height n in $\mathcal{K}^s$, if such a maximum exists; otherwise the height of $\mathcal{K}^s$ is infinite. We will construct a model such that any strictly ascending chain starting from 1 is finite. Hence, there is no infinite strictly ascending chain in it. Now we introduce the notion of the degree of a modal formula φ.

Definition 1. *We define the degree of a modal formula as follows.*
1. $deg(p)=0$; 2. $deg(\bot)=0$; 3. $deg(\neg\varphi)=deg(\varphi)$;
4. $deg(\varphi * \psi)=max\{deg(\varphi), deg(\psi)\}$, where $* \in \{\vee, \wedge. \rightarrow\}$;
5. $deg(\boxdot\varphi)=deg(\varphi)+1$; 6. $deg(t:\varphi)=deg(\varphi)+1$.

Let us define n-bisimulation (bounded bisimulation) on arbitrary Fitting models $\mathcal{K}=(K,R,R^e,r,\mathcal{E},\Vdash)$. This satisfies minimal conditions as follows: there are no conditions for R but we assume $R\subseteq R^e$, $\mathcal{E}$ is stable w.r.t. R^e. The definition of n-bisimulation here is based on [4]. We added some machinery to handle logic of proofs. Let $\mathcal{K}$ and $\mathcal{K}'$ be such Kripke-Fitting models in the following.

Definition 2. *We call w and w' "n-bisimilar" ($w \simeq_n w'$) if there exists a sequence of binary relations $Z_n \subseteq \cdots \subseteq Z_0$ with the following properties (for $i+1 \le n$): (0) $\mathcal{E}(v,t)=\mathcal{E}(v',t)$ for any $v\in K$ and for any $v'\in K'$; (1) wZ_nw';*
(2) If vZ_0v', then v and v' agree on all propositional letters;
(3) If $vZ_{i+1}v'$ and vRu, then $\exists u'$ ($v'Ru'$ and uZ_iu');
(4) If $vZ_{i+1}v'$ and $v'Ru'$, then $\exists u$ (vRu and uZ_iu');
(5) If $vZ_{i+1}v'$ and vR^eu, then $\exists u'$ ($v'R^eu'$ and uZ_iu');
(6) If $vZ_{i+1}v'$ and $v'R^eu'$, then $\exists u$ (vR^eu and uZ_iu').

The condition (0) is motivated by the stability of evidence function, i.e. uR^ev implies $\mathcal{E}(u,t)=\mathcal{E}(v,t)$.

Definition 3. *We define the notion of a generated subset $U(v)$ of the underlying set K of a model $\mathcal{K}$ from v with respect to R (R^e), respectively, as follows: $U(v)=\{u\in K|\exists j(0\le j \text{ and } v(R)^ju)\}$; $U_e(v)=\{u\in K|\exists j(0\le j \text{ and } v(R^e)^ju)\}$*
Similarly, we define a bounded generated subset $U^n(v)$ from v with respect to R as follows: $U^n(v)=\{u\in K|\exists i(0\le i\le n \text{ and } v(R)^iu)\}$.

Here we only finitize the height of the canonical model $\mathcal{K}^s$ for GrzA (i.e., our model may not be a finite one). To do that, we first prove a few general propositions stating properties of bounded bisimulation. By $w\equiv_n w'$, we mean that for any φ s.t. $deg(\varphi)\le n$, $w\Vdash\varphi$ iff $w'\Vdash\varphi$. We now state a lemma and a proposition.

Lemma 4. *1. For any $w,v\in K$ and $w',v'\in K'$, $w\simeq_{n+1}w'$, wRv, $w'R'v'$, and vZ_nv' implies $v\simeq_n v'$. 2. For any $w,v\in K$ and $w',v'\in K'$, $w\simeq_{n+1}w'$, wR^ev, $w'R^{e\prime}v'$, and vZ_nv' implies $v\simeq_n v'$.*

Proposition 3. *For any $w\in\mathcal{K}$ and $w'\in\mathcal{K}'$, $w\simeq_n w' \Longrightarrow w\equiv_n w'$.*

Proof. For proofs of lemma 4, proposition 3, see the appendix 1, 2, respectively.

Step 4: The countermodel to be constructed has to satisfy the required frame condition for a model of GrzA, i.e. there is no infinite strictly ascending chain. Thus, we construct a finite-height model that is n-bisimilar to the original canonical model $\mathcal{K}^s$ for GrzA. We give a few definitions. We use the notation $\mathcal{K}[r]$ to stand for a model $\mathcal{K}$ with the root node r. Also, let $U^n(r) = \{v \in K^s | \exists i(0 \leq i \leq n$ and $r(R^s)^i v)\}$, taking r in the definition of a bounded generated subset of K^s. (Clearly, $U^n(r) \subseteq K^s$, and $U^n(r) \times U^n(r) \subseteq K^s \times K^s$.) The restriction of $\mathcal{K}^s$ to $U^n(r)$ (we use the notation $\mathcal{K}^s \restriction U^n(r)$) is defined as follows. The underlying set of $\mathcal{K}^s \restriction U^n(r)$ is $K^s \cap U^n(r)$, accessibility relations are restriction of R^s, R^e, i.e. $R^s \cap (U^n(r) \times U^n(r))$ (we use the notation R^{s^*} for this) and $R^e \cap (U^n(r) \times U^n(r))$ (we use the notation R^{e^*} for this), the evidence function is a restriction of the function w.r.t. the domain of the function $\mathcal{E} \restriction U^n(r)$ (we use the notation $\mathcal{E}^*$ for this), and the forcing relation is also a restriction of the first coordinate of the relation, $\Vdash \restriction U^n(r)$ (we use the notation $\Vdash^*$ for this). Hence, $\mathcal{K}^s \restriction U^n(r) = (K^s \cap U^n(r), R^{s^*}, R^{e^*}, r, \mathcal{E}^*, \Vdash^*)$. The construction of n-bisimulation defined above can be applied to the two models $\mathcal{K}^s \restriction U^n(r)$ and $\mathcal{K}^s$.

Proposition 4. $(\mathcal{K}^s \restriction U^n(r), r) \simeq_n (\mathcal{K}^s, r)$ *for each* $n \in \mathbb{N}$.

In order to define n-bisimulation, it suffices to use $n+1$-sequence of relations $Z_n \subseteq Z_{n-1} \subseteq \cdots \subseteq Z_0$ such that the conditions of bounded bisimulation (3), (4), (5) and (6) hold among R^s, R^{s^*}, R^e, R^{e^*} and Z_i for any $0 \leq i \leq n$. We want to show that a rooted model $\mathcal{K}^s[r]$ restricted to $U^n(r)$ is sufficient to show that n-bisimulation always holds between $(\mathcal{K}^s, r)$ and $(\mathcal{K}^s[r] \restriction U^n(r), r)$.

Proof. Proof is by induction on n. We omit the details due to the limitation of space, but they are relatively straightforward.

This suffices to show that for any formula φ s.t. $deg(\varphi) = n$, $\mathcal{K}^s, r \Vdash \varphi$ iff $\mathcal{K}^s[r] \restriction U^n(r), r \Vdash^* \varphi$. The height of the model $\mathcal{K}^s[r] \restriction U^n(r)$ is n. Hence, for a formula φ for which we have constructed the canonical model $\mathcal{K}^s$ s.t. $\mathcal{K}^s, 1 \nVdash \varphi$, we have a model $\mathcal{K}^s[r] \restriction U^n(r)$, such that $\mathcal{K}^s[1] \restriction U^n(1), 1 \nVdash \varphi$, where $r = 1$ and the height of $\mathcal{K}^s[1] \restriction U^n(1)$ is at most $deg(\varphi) = n$. This model may still be an infinite model since there may be some infinite branching in the tree. However, for our purpose, a finite-height model suffices. The following are simple consequences of the foregoing proposition.

Corollary 2. *1.* R^{s^*} *has no strictly ascending infinite chain. 2.* R^{e^*} *is an equivalence relation. Moreover, for any* $\sigma, \sigma' \in K^s \cap U^n(1)$, $\sigma R^{e^*} \sigma'$.

Corollary 3. *For any* ψ, *s.t.* $deg(\psi) = n$, $\mathcal{K}^s[1] \restriction U^n(1), 1 \Vdash^* \psi$ *iff* $\mathcal{K}^s, 1 \Vdash \psi$.

Theorem 2 (Weak Completeness)
If $\{1F\varphi\}$ *has no closed tableau, then there exists a finite-height GrzA model* $\mathcal{K}$, *s.t.* $\mathcal{K} = (K, R, R^e, r, \mathcal{E}, \Vdash)$, *s.t.* $r \nVdash \varphi$.

Proof. We show the contrapositive. Suppose φ is not provable using the prefixed GrzA-tableau rules. Then $\{1F\varphi\}$ is GrzA-consistent, and it omits infinitely many integers. So, we can extend it to a (restricted) maximally GrzA-consistent, π-complete set S_ω by the above construction. We can define a canonical Kripke-Fitting model $\mathcal{K}^s$ out of S_ω. By Truth Lemma, we can show $F\varphi$ is realized at 1 in $\mathcal{K}^s$. $\mathcal{K}^s, 1 \nVdash \varphi$. By finitizing the height of the model using the proposition presented above, this is equivalent to $\mathcal{K}^s[1] \restriction U^n(1), 1 \nVdash^* \varphi$. Hence, there is a finite height Kripke-Fitting model $\mathcal{K}$ and the root r such that $\mathcal{K}, r \nVdash \varphi$. ⊠

4 Prefixed Tableau System for wGLA

Now we go back to GLA. Recall we defined an auxiliary subsystem wGLA of GLA without the axiom 3)-3 $t\colon\Box\varphi \to \varphi$ and Reflexive rule, 3. $\Box\varphi/\varphi$. Both of them are sound with respect to Kripke-Fitting semantics with the appropriate root soundness condition. However, to prove completeness smoothly, we first give an auxiliary prefixed tableau system for wGLA (called "TwGLA"). Here we keep using Fitting's terminology (in [6]), but we write $\sigma < \sigma'$ if σ is a proper initial segment of σ'.

α-rule, β-rules, $\perp$-rule, and the rules for LP are the same as those of TGrzA.

Modal Rules: ν-rules: K $\dfrac{\sigma T\Box\varphi}{\sigma.nT\varphi}$ ($\sigma.n$ is used.) 4 $\dfrac{\sigma T\Box\varphi}{\sigma.nT\Box\varphi}$ ($\sigma.n$ is used.)

The π-rule for GL: $\dfrac{\sigma F\Box\varphi}{\begin{array}{c}\sigma.nF\varphi\\ \sigma.nT\Box\varphi\end{array}}$ ($\sigma.n$ is new.)

$\mathcal{CS}$ Rules: a branch is closed if it has $\sigma Fc\colon A$ (A is an axiom of wGLA and $A \in \mathcal{CS}(c)$.)

An interpretation function $\mathcal{N}$ for wGLA, etc. are defined in the same manner as the one given to GrzA. Note that we use the same convention for a prefix being "used" on a branch as we used in GrzA. We only state the soundness and completeness of TwGLA, since the Lindenbaum-Henkin construction and the argument to ensure that R, R^e, $\Vdash$ and $\mathcal{E}$ all satisfy the conditions of a wGLA are similar to those of GrzA.[6]

Theorem 3 (Soundness for wGLA)
If φ has a prefixed wGLA-tableau proof, then φ is valid in all GLA models.

Theorem 4 (Weak Completeness for wGLA)
If $\{1F\varphi\}$ has no closed tableau, then there exists a GLA Kripke-Fitting model $\mathcal{K}^*$ $(= (K^*, R^*, R^{e^*}, r^*, \mathcal{E}^*, \Vdash^*))$, s.t. $1 \nVdash^* \varphi$.

[6] The two major differences are the following: (1) the definition of S_n in the Lindenbaum-Henkin construction is modified as follows. In the case when $S_n \cup \{\sigma_n \Psi_n\}$ is consistent, $\Psi_n = \pi = F\Box\psi_n$), $\pi_0 = F\Box\psi_n$, and $\sigma_n.k$ is new, we let $S_{n+1} = S_n \cup \{\sigma_n\Psi_n\} \cup \{\sigma_n.k\pi_0\} \cup \{\sigma_n.kT\Box\psi_n\}$. (2) in the definition of the canonical model, we have $\sigma R\sigma'$ iff $\sigma < \sigma'$.

5 Prefixed Tableau System for GLA

Now we extend the completeness theorem from wGLA to the full GLA. We first formulate a prefixed tableau system for GLA, which we call TGLA. It turns out that the axiom 3)-3 $t:\Box\varphi \to \varphi$ and Reflection Rule $\Box\varphi/\varphi$ can be dealt with by adding one tableau rule called "Reflection Rule." We need to add Reflection Rule and the following modification of TwGLA to obtain TGLA.

1. Constant specification has to be modified accordingly. Instead of taking all the axioms for wGLA, we use all the axioms of the full GLA.

2. Reflection Rule: From $\{1F\varphi\}$, $\dfrac{\cancel{1F\varphi}}{\{1F\Box\varphi\}}$ can be derived.

Note : This crossing is to emphasize that $1F\varphi$ is not on the same tableau after an application Reflection Rule. Due to this feature, this can be taken as a kind of destructive rule in the sense of [6]. Although satisfiability is preserved from the premise to the conclusion in this rule, we move from one model $\mathcal{K}$ (in the premise) to another $\mathcal{K}^+$ (in the conclusion). Accordingly, the premise and the conclusion have the same prefix 1, but their interpretations of the prefix 1 are different.

In addition to these, we reformulate the notion of a *tableau proof* in TGLA.

Definition 4. *φ has a tableau proof in TGLA if $\{1F\Box\varphi\}$ has a closed tableau in TGLA.*

An example: Connection axiom $t:\Box\Box\varphi \to \varphi$ is derivable by this rule.[7]

1.	~~$1F\Box(t:\Box\Box\varphi \to \varphi)$~~	
2.	$1F\Box\Box(t:\Box\Box\varphi \to \varphi)$	(Reflection Rule, line 1)
3.	$1.1F\Box(t:\Box\Box\varphi \to \varphi)$; $1.1T\Box\Box(t:\Box\Box\varphi \to \varphi)$	(π-rule, line 2)
4.	$1.1.1F(t:\Box\Box\varphi \to \varphi)$; $1.1.1T\Box(t:\Box\Box\varphi \to \varphi)$	(π-rule, line 3)
5.	$1.1.1Tt:\Box\Box\varphi$	(α-rule, line 4)
6.	$1.1.1F\varphi$	(α-rule, line 4)
7.	$1.1Tt:\Box\Box\varphi$	(E4r, line 5)
8.	$1Tt:\Box\Box\varphi$	(E4r, line 7)
9.	$1T\Box\Box\varphi$	(ET, line 8)
10.	$1.1T\Box\varphi$	(ν-rule for K, line 9)
11.	$1.1.1T\varphi$	(ν-rule for K, line 10)
	$\times$	

We show the soundness of the Reflection Rule with respect to the appropriate root sound Kripke-Fitting semantics. (Soundness of the other rules are already shown.)

Lemma 5. *Suppose $\mathcal{T} = \{1F\varphi\}$ is a tableau that is satisfiable in a φ-sound GLA model. If reflection rule for GLA is applied to $\mathcal{T}$, then the resulting tableau $\{1F\Box\varphi\}$ is still satisfiable in a $\Box\varphi$-sound GLA model.*

Proof. Suppose $\{1F\varphi\}$ is satisfiable in a φ-sound GLA model. We want to show that $\{1F\Box\varphi\}$ is satisfiable in a $\Box\varphi$-sound GLA model. (Note that $\{\Box\psi \to \psi | \Box\psi \in Sb(\varphi)\} \subseteq \{\Box\psi \to \psi | \Box\psi \in Sb(\Box\varphi)\}$. In addition, the only formula from the latter set that is missing from the former is $\Box\varphi \to \varphi$, since $Sb(\Box\varphi) \backslash Sb(\varphi) = \{\Box\varphi\}$.)

[7] To save space, we use ";" to write more than one formula horizontally.

Satisfiability implies that there exists a GLA-interpretation (partial) function $\mathcal{N}$ and a GLA model that is φ-sound, s.t. $\mathcal{K}, \mathcal{N}(1) \nVdash \varphi$ and $\mathcal{N}(1) = r$ $(r \in K)$. To show $1F\Box\varphi$ is satisfiable in a $\Box\varphi$-sound GLA model, we construct another GLA-model that is $\Box\varphi$-sound and an interpretation function $\mathcal{N}^+$ that satisfies $\{1F\Box\varphi\}$, following the argument in [1] that uses a method of gluing a new root node. Let $\mathcal{K}^+$ be a sextuple $(K^+, R^+, R^{e^+}, r^+, \mathcal{E}^+, \Vdash^+)$, s.t.

1. $K^+ = \{r^+\} \cup K$, where r^+ is a new root node of $\mathcal{K}^+$ $(r^+ \notin K)$.
2. $R^+ = \{(r^+, r)\} \cup \{(r^+, y) | (r, y) \in R\} \cup R$
3. $R^{e^+} = \{(r^+, y) | y \in K^+\} \cup R^e$.
4. $\mathcal{E}^+$ is s.t. $\forall u \in K$ $[\mathcal{E}^+(r^+, t) = \mathcal{E}(u, t)$ and $\mathcal{E}^+(u, t) = \mathcal{E}(u, t)]$.[8]
5. The forcing relation $\Vdash^+$ is defined as follows.

5.1. For any propositional variable $p \in Sb(\varphi)$,
if $u = r^+$, then $u \Vdash^+ p$ iff $r \Vdash p$, and if $u \neq r^+$, then $u \Vdash^+ p$ iff $u \Vdash p$.

5.2. At any $u \in K^+$, $\Vdash^+$ commutes with Booleans for any formula in $Sb(\varphi)$.[9]

5.3. For any $\Box\psi \in Sb(\varphi)$, $\forall u \in K^+$, $u \Vdash^+ \Box\psi$ iff $\forall v \in K^+$, $uR^+v \Longrightarrow v \Vdash^+ \psi$

5.4. For any $t{:}\psi \in Sb(\varphi)$, $\forall u \in K^+$, $u \Vdash^+ t{:}\psi$ iff $\psi \in \mathcal{E}^+(u, t)$ and $\forall v \in K^+$ $(uR^{e^+}v \Longrightarrow v \Vdash^+ \psi)$

6. (c.s.) $A \in \mathcal{CS}(c)$ (($\mathcal{CS}$) for GLA) implies $K^+, u \Vdash^+ c{:}A$ for all $u \in K^+$

On this model $\mathcal{K}^+$, we claim the following.

Claim. $\forall\psi \in Sb(\varphi)$ $\forall u \in K^+$ $[(u = r^+ \Longrightarrow u \Vdash^+ \psi$ iff $r \Vdash \psi)$ and $(u \neq r^+ \Longrightarrow u \Vdash^+ \psi$ iff $u \Vdash \psi)]$.

Proof. Induction on the structure of ψ.

Let us redefine a new (extended) interpretation function $\mathcal{N}^+$ over the extended model.[10] Let $\mathcal{N}^+(1) = r^+$ and $\mathcal{N}^+(1.\sigma) = \mathcal{N}(\sigma)$. So, $\mathcal{K}^+, \mathcal{N}^+(1) \nVdash^+ \Box\varphi$. Hence, $F\Box\varphi$ is realized in $\mathcal{N}^+(1)$ in $\mathcal{K}^+$. Thus, $\{1F\Box\varphi\}$ is satisfiable. Also, by the last claim, $\mathcal{K}^+$ is $\Box\varphi$-sound. ⊠(Lemma)

Soundness of GLA can be shown as follows. The proof of soundness is different from that of usual modal logics, due to the φ-soundness of a model.

Theorem 5 (Soundness for GLA)
If φ has a prefixed GLA-tableau proof, then φ is valid in all φ-sound GLA models.

Proof. Suppose φ has a tableau proof in TGLA. Namely, $\{1F\Box\varphi\}$ has a closed tableau TGLA. And suppose that $\Box\varphi$ is invalid. Then, there exists a $\Box\varphi$-sound

[8] The definition of $\mathcal{E}^+$ goes beyond $Sb(\Box\varphi)$ due to its closure conditions. However, in order to prove this lemma, we do not have to go beyond $Sb(\Box\varphi)$ since the way we use $\Vdash^+$ does not go into the term induction, but only formula induction within $Sb(\Box\varphi)$.

[9] Extend $\Vdash$ so that the new forcing relation preserves $\Vdash$'s feature of commuting with Booleans in the new state r^+. The old $\Vdash$ commutes with Booleans in all states in K.

[10] We modify $\mathcal{N}$ so that $1 < \sigma \Longrightarrow \mathcal{N}^+(1)R^+\mathcal{N}^+(\sigma)$ (for $\sigma \neq 1$). We omit the details but $\mathcal{N}^+$ can be shown to be well-defined.

GLA-model, s.t. $\exists u \in K$, s.t. $u \nVdash \Box\varphi$. For u, we have $u = \mathcal{N}(1)$ or $u \neq \mathcal{N}(1)$. In the former case, $\mathcal{N}(1) \nVdash \Box\varphi$. So, $\{1F\Box\varphi\}$ is satisfiable (in a $\Box\varphi$-sound GLA-model). Since all the rules of TGLA preserve satisfiability, any tableau constructed from $\{1F\Box\varphi\}$ is satisfiable. But no satisfiable tableau is closed. Contradiction.

In the latter case, since $\mathcal{N}(1)$ is the root node, $\mathcal{N}(1)Ru$. On the other hand, $u \nVdash \Box\varphi$ implies that there exists $v \in K$, s.t. uRv and $v \nVdash \varphi$.

By transitivity, $\mathcal{N}(1)Ru$ and uRv implies $\mathcal{N}(1)Rv$. $\mathcal{N}(1)Rv$ and $v \nVdash \varphi$ implies $\mathcal{N}(1) \nVdash \Box\varphi$. So, $\{1F\Box\varphi\}$ is satisfiable (in a $\Box\varphi$-sound GLA-model). Since all the rules of TGLA preserve satisfiability, any tableau constructed from $\{1F\Box\varphi\}$ is satisfiable. But no satisfiable tableau is closed. Contradiction.

Hence, $\{1F\Box\varphi\}$ has a closed tableau TGLA $\Longrightarrow \Box\varphi$ is valid in all $\Box\varphi$-sound GLA models. However, by the lemma, $\Box\varphi$ is valid in all $\Box\varphi$-sound GLA models $\Longrightarrow \varphi$ is valid in all φ-sound GLA models. Therefore, $\{1F\Box\varphi\}$ has a closed tableau TGLA $\Longrightarrow \varphi$ is valid in all φ-sound GLA models. ⊠

Next, we show weak completeness of TGLA for a formula φ with respect to φ-sound Kripke-Fitting models. We prove this by taking the following outline.

(1) Assume unprovability of φ, i.e. $\{1F\Box\varphi\}$ has no closed tableau in TGLA.

(2) We prove that $\{1F\Box^{N+1}\varphi\}$ has no closed tableau in TwGLA. (Here N stands for $|\{\Box\psi|\Box\psi \in Sb(\varphi)\}|$.)

(3) By using completeness of TwGLA, we can show that there exists a GLA-model $\mathcal{K}$ such that $\mathcal{K}, r \nVdash \Box^{N+1}\varphi$.

(4) By using an argument occasionally used in the literature of provability logic (in [10]), we transform the model constructed in (3) into a φ-sound GLA-model.

Proposition 5 ((1) ⇒ (2)). *If there is a closed tableau for $1F\Box^n\varphi$ $(n > 1)$ in TwGLA, then $1F\Box\varphi$ has a closed tableau in TGLA.*

Proof. Suppose $1F\Box^n\varphi$ has a closed tableau in TwGLA $(n > 1)$. Pick one. Then apply reflection rule $n - 1$ times on top of it. The resulting tableau must be a closed tableau for $1F\Box\varphi$ in TGLA. ⊠

Now we prove the weak completeness of GLA. (This involves steps from (2) to (4).) Our proof is based on a proof given in [1], which also goes back to [10].

Theorem 6 (Completeness for GLA)

If φ has no tableau proof in TGLA, i.e., $\{1F\Box\varphi\}$ has no closed tableau in TGLA, then there exists a φ-sound GLA Kripke model $\mathcal{K}$ $(= (K, R, R^e, r, \mathcal{E}, \Vdash))$ and there exists $u \in K$, s.t. $\mathcal{K}, u \nVdash \varphi$.

Proof. Suppose $1F\Box\varphi$ has no closed tableau in TGLA. In particular, this implies that there is no closed tableau for this in TwGLA. By completeness of wGLA, we have a wGLA-countermodel for φ. However, this does not guarantee that the model is a φ-sound GLA countermodel for φ. The following argument shows this.

Let $N = |\{\Box\psi|\Box\psi \in Sb(\varphi)\}|$. Proposition 5 (i.e., (1) ⇒ (2)). implies that if $1F\Box^{N+1}\varphi$ has a closed tableau in wGLA, then $1F\Box\varphi$ has a closed tableau in

GLA. So, taking the contrapositive, by the assumption, we obtain the statement $1F\Box^{N+1}\varphi$ has no closed tableau in wGLA. By completeness of wGLA, this implies that there exists a (not necessarily φ-sound) GLA-model $\mathcal{K}$ s.t. $\mathcal{K}, r \nVdash \Box^{N+1}\varphi$. (This corresponds to the step (2) $\Rightarrow$ (3) in the outline.)

Now we show the step (3) $\Rightarrow$ (4) in the outline. $\mathcal{K}, r \nVdash \Box^{N+1}\varphi$ implies that there is a sequence of nodes $r = a_0Ra_1Ra_2\ldots a_NRa_{N+1}$, s.t. $a_i \nVdash \Box^{N+1-i}\varphi$ where $0 \leq i \leq N+1$. (Note that $a_{N+1} \nVdash \varphi$.)

None of the formulas $\Box\psi \to \psi$ ($\Box\psi \in Sb(\varphi)$) can be false at two (or more) different nodes a_i and a_j. Indeed, suppose there is such a ψ_1. Then, $a_i \nVdash \Box\psi_1 \to \psi_1$ and $a_j \nVdash \Box\psi_1 \to \psi_1$, where we assume $a_i < a_j$ w.l.o.g. Then, $a_i \Vdash \Box\psi_1$ and $a_i \nVdash \psi_1$. Also, $a_j \Vdash \Box\psi_1$ and $a_j \nVdash \psi_1$. But this clearly raises a contradiction.

We have N formulas of the form $\Box\psi \to \psi$, each of which is false at most one world in a chain of $N+1$-many world. Then, there exists at least one world a_i in which $\Box\psi \to \psi$ are true at a_i. Namely, $\exists i$ $(0 \leq i \leq N+1)$, s.t. $a_i \Vdash \bigwedge\{\Box\psi \to \psi | \Box\psi \in Sb(\varphi)\}$. We may have either a case a_iRa_{N+1} or a case $a_i = a_{N+1}$. Either way, we take the restriction of $\mathcal{K}$. More precisely, take such a subset $K'(= \{u \in K | a_iRu \text{ or } a_i = u\})$ as the underlying set of the Kripke-Fitting model that we want to construct, take the two accessibility relation as the restrictions of R and R^e by this set accordingly, take the evidence function as the restriction of $\mathcal{E}$ concerning its first coordinate to the above set, take the restriction of forcing relation, and let the new root $r' = a_i$. We call the new model $\mathcal{K}' = (K', R', R^{e\prime}, r', \mathcal{E}', \Vdash')$. Then, by construction, $\mathcal{K}'$ is a φ-sound GLA model. Note that this new root was already a part of a model of wGLA, s.t. $r' \nVdash \Box^{N+1-i}\varphi$. Also, this truncation does not affect the structure of that part of the model which is used to ensure that for some $u \in K'$, $u \nVdash \varphi$ since the restriction is made below a_{N+1}. For $R^{e\prime}$, this is the restriction of R^e to $K' \times K'$, i.e., $R^{e\prime} = R^e \cap (K' \times K')$. It is easy to check that this is still an equivalence relation. So, $R^{e\prime}$ is also taken care of. Therefore, the restriction $\mathcal{K}'$ is a φ-sound GLA model such that for some $v \in K'$, $v \nVdash \varphi$. ⊠

Corollary 4. *Cut is admissible in TGLA.*

Corollary 5. *GLA is a conservative extension of GL.*

6 Discussions

Let us briefly mention that although we have been able to show that cut is admissible in TGLA, unfortunately this does not imply the subformula property (as discussed in [12]). This is due to the presence of the axiom of LP $t{:}(\varphi \to \psi) \to (s{:}\psi \to t \cdot s{:}\psi)$. This can be taken to be a kind of internal modus ponens, which is a counterpart of cut in Hilbert-style systems. G. Jäger [11] introduced a convenient terminology to describe the situation, i.e. "internal cut / external cut." In all the currently available sequent calculi for logics including LP one way or another, we have the admissibility of the external cut but not that of the internal cut (i.e., the foregoing axiom). Whether we can ultimately eliminate the internal cut or not is a challenging problem, and we may need a radically different formulation of LP in order to solve this problem. We leave this issue to our future research.

References

1. Artëmov, S.N.: Logic of proofs. Ann. Pure Appl. Logic 67(1-3), 29–59 (1994)
2. Artemov, S.N.: Explicit provability and constructive semantics. The Bulletin of Symbolic Logic 7(1), 1–36 (2001)
3. Artemov, S.N., Nogina, E.: Logic of knowledge with justifications from the provability perspective. Technical report, CUNY Ph.D. Program in Computer Science Technical Report TR-2004011 (2004)
4. Blackburn, P., de Rijke, M., Venema, Y.: Modal Logic. Cambridge University Press (2001)
5. Boolos, G.: The logic of provability. Cambridge University Press (1992)
6. Fitting, M.: Proof Methods for Modal and Intuitionistic Logic. Reidel Publishing Company (1983)
7. Fitting, M.: Modal proof theory. In: Handbook of Modal Logic. Elsevier, New York (2006)
8. Goranko, V., Otto, M.: Model theory of modal logic. In: Blackburn, P., Wolter, F., van Benthem, J. (eds.) Handbook of Modal Logic, pp. 249–329. Kluwer (2007)
9. Goré, R.: Tableau methods for modal and temporal logics. In: D'Agostino, M., Gabbay, D., Hähnle, R., Posegga, J. (eds.) Handbook of Tableau Methods, pp. 297–396. Kluwer (1999)
10. Guaspari, D., Solovay, R.M.: Rosser sentences. Ann. Math. Logic 16(1), 81–99 (1979)
11. Jäger, G.: Modal fixed point logics. In: Esparza, J., Spanfelner, B., Grumberg, O. (eds.) Logics and Languages for Reliability and Security. NATO Science for Peace and Security Series - D: Information and Communication Security, vol. 25. IOS Press (2010)
12. Kurokawa, H.: Tableaux and hypersequents for justification logics. Ann. Pure Appl. Logic 163(7), 831–853 (2012)
13. Nogina, E.: Epistemic completeness of GLA. The Bulletin of Symbolic Logic 13(3), 407 (2007)
14. Nogina, E.: Logic of strong provability and explicit proofs. In: Proceedings of Logic Colloquium 2008 (2008)
15. Yavorskaya, T.: Logic of proofs and provability. Ann. Pure Appl. Logic 113(1-3) (2001)

Appendix 1: Proof of the Lemma 4

1. Suppose $w \simeq_{n+1} w'$, wRv, $w'R'v'$, and vZ_nv'. Then there exists a sequence $Z_{n+1} \subseteq \cdots \subseteq Z_0$ that satisfies the conditions of $w \simeq_{n+1} w'$. We now construct a new sequence of relations (we call them Z'_i) based on these Z_i that satisfies the condition of $v \simeq_n v'$.

To do that, take the generated subsets $U_e(v) = \{u \in K | \exists j(0 \leq j \text{ and } v(R^e)^j u)\}$ and $U'_e(v') = \{u \in K' | \exists j(0 \leq j \text{ and } v(R^{e\prime})^j u)\}$.[11] ($i = 0$ means that v (or v') itself in the set, respectively.) Let $Z'_i = Z_i \cap (U_e(v) \times U'_e(v'))$ and take the sequence of Z'_i up to n $(0 \leq i \leq n)$, s.t. $Z'_n \subseteq \cdots \subseteq Z'_1 \subseteq Z'_0$. We show that this sequence satisfies all the conditions for $v \simeq_n v'$.

[11] We use R^e, R^e here to take of both (3), (4) and (5), (6). Using R is not enough to show the conditions (5) or (6).

For (0), this condition is the same in $\simeq_{n+1}$ and $\simeq_n$, so it is obvious.

For (1), vZ_nv' is given by assumption, and clearly $(v, v') \in U_e(v) \times U_e(v')$.

For (2), since $Z_0' \subseteq Z_0$ and Z_0 satisfies the condition (2), Z_0' satisfies (2).

For (3), suppose $xZ_{i+1}'x'$ and xRy $(1 \leq i+1 \leq n)$. Since $Z_{i+1}' \subseteq Z_{i+1}$, $xZ_{i+1}x'$ and xRy. By the condition (3) of $w \simeq_{n+1} w'$, $\exists y'(x'R'y'$ and $yZ_iy'))$. Call the state y_1'. xRy and $x'R'y_1'$ implies xR^ey and $x'R^{e\prime}y_1'$. Since $(x, x') \in U_e(v) \times U_e'(v')$ (this follows from $xZ_{i+1}'x'$), $(y, y_1') \in U_e(v) \times U_e'(v')$. Thus, $yZ_i'y_1'$. Therefore, $\exists y'(x'R'y'$ and $yZ_i'y'))$. For (4), (5) and (6), the proof is similar to the case (3). (Case (5) and (6) are even slightly simpler since we directly consider R^e.)

2. The proof of the second statement is essentially the same as above. ⊠

Appendix 2 : Proof of Proposition 3

Proof by induction on n. Base case: $n = 0$. $deg(\varphi) = 0$. Here φ is either a propositional variable and their Boolean combination. Due to the condition (1) and (2) for $w \simeq_0 w'$, i.e. wZ_0w', and if wZw' then $w \Vdash p$ iff $w' \Vdash p$. Boolean cases are straightforward.

Inductive case: We assume the statement for n, namely $\mathcal{K}, w \simeq_n \mathcal{K}', w' \Longrightarrow w \equiv_n w'$, and prove the statement for $n+1$, namely $\mathcal{K}, w \simeq_{n+1} \mathcal{K}', w' \Longrightarrow w \equiv_{n+1} w'$. We have two subcases.

Subcase 1. $\varphi = \Box\psi$ $(deg(\varphi) = deg(\Box\psi) = \mathrm{deg}(\psi) + 1 = n + 1)$. We want to show that $w \simeq_{n+1} w' \Longrightarrow w \nVdash \Box\psi$ iff $w' \nVdash \Box\psi$.

To show this, suppose (A) $w \simeq_{n+1} w'$, and suppose (B) $w \nVdash \Box\psi$, i.e. $\exists v(wRv$ and $v \nVdash \psi)$. Call this state v_1. Then, wRv_1 and $v_1 \nVdash \psi$. By the condition (1) of the assumption (A), $wZ_{n+1}w'$. So, we have $wZ_{n+1}w'$ and wRv_1. By condition (4) of the assumption (A), $\exists v'(w'R'v'$ and $v_1Z_nv')$. Call this v_1'. So $w'R'v_1'$ and $v_1Z_nv_1'$. Note that we now have all the statements in the assumptions of the lemma with respect to particular instances v_1, v_1'. Hence, $v_1 \simeq_n v_1'$.

By IH, $v_1 \simeq_n v_1' \Longrightarrow v_1 \Vdash \psi$ iff $v_1' \Vdash \psi$. Since the antecedent of the claim is already shown above, we have $v_1' \nVdash \psi$. So, $\exists v' \in K(w'R'v'$ and $v' \nVdash \psi)$. Therefore, $w' \nVdash \Box\psi$. The other direction is similar.

Subcase 2. $\varphi = t{:}\psi$ $(deg(\varphi) = deg(t{:}\psi) = deg(\psi) + 1)$.

We want to show that $w \simeq_{n+1} w' \Longrightarrow w \nVdash t : \psi$ iff $w' \nVdash t : \psi$. To show this, suppose (A) $w \simeq_{n+1} w'$, and suppose (B) $w \nVdash t : \psi$, i.e. $\psi \notin \mathcal{E}(w, t)$ or $\exists v \in K(wR^ev$ and $v \nVdash \psi)$. (Call it v_1.)

By the condition (0) of $w \simeq_{n+1} w'$, without depending the number $n+1$, the first disjunct implies $\psi \notin \mathcal{E}'(w', t)$. So, $\psi \notin \mathcal{E}'(w', t)$ or $\exists v' \in K'(wR^{e\prime}v'$ and $v' \nVdash \psi)$ is implied by the first disjunct.

By the condition (1) of the assumption (A), $wZ_{n+1}w'$. Also, the second disjunct implies wR^ev_1. Hence, by the condition (5) of (A), $\exists v' \in K'(v_1Z_nv'$ and $w'R^{e\prime}v')$ (Call it v_1'). So, we have $w'R^{e\prime}v_1'$ and $v_1Z_nv_1'$.

By the above lemma (the statement 2), since we have shown the particular instance of the assumptions of this lemma, we can show $v_1 \simeq_n v'$.

By IH, $v_1 \simeq_n v_1' \Longrightarrow v_1 \nVdash \psi$ iff $v_1' \nVdash \psi$. Hence, $v_1 \nVdash \psi$ iff $v_1' \nVdash \psi$.

Thus, $v_1' \nVdash \psi$. So, we have $\exists v' \in K'(w'R^{e\prime}v'$ and $v' \nVdash \psi)$.

So, either way, $\psi \notin \mathcal{E}'(w', t)$ or $\exists v' \in K'(wR^{e\prime}v'$ and $v' \nVdash \psi)$ is derivable. Therefore, $w' \nVdash t{:}\psi$. The converse is similar. ⊠ (proposition).

Correspondence between Modal Hilbert Axioms and Sequent Rules with an Application to S5[⋆]

Björn Lellmann[1] and Dirk Pattinson[1,2]

[1] Department of Computing, Imperial College London
[2] Research School of Computer Science, The Australian National University

Abstract. Which modal logics can be 'naturally' captured by a sequent system? Clearly, this question hinges on what one believes to be natural, i.e. which format of sequent rules one is willing to accept. This paper studies the relationship between the format of sequent rules and the corresponding syntactical shape of axioms in an equivalent Hilbert-system. We identify three different such formats, the most general of which captures most logics in the S5-cube. The format is based on restricting the context in rule premises and the correspondence is established by translating axioms into rules of our format and vice versa. As an application we show that there is no set of sequent rules of this format which is sound and cut-free complete for S5 and for which cut elimination can be shown by the standard permutation-of-rules argument.

1 Introduction

Syntactical descriptions of modal logics often are given in terms of Hilbert calculi. This allows for a simple, intuitive description of the logic and elegant completeness proofs via canonical models. But Hilbert calculi are not optimal to establish e.g. decidability and interpolation for which Sequent systems are far better suited. This raises the question of a precise correspondence between Hilbert and Sequent calculi, and more generally of a classification of the structural, proof-theoretic machinery such as labels, nested sequents and the format of sequent rules that is necessary to give complete (cut-free) sequent systems for a given modal logic. This clearly depends on the format of sequent rules as every logic L can be captured trivially by the system $\{ \Rightarrow A \mid A \in L\}$ where A ranges over the theorems of the logic.

Here, we consider rules that are close in spirit to the standard sequent rules for modal logics such as K, KT and $\mathsf{S4}$ and restrict ourselves to pure two-sided sequents with the comma as only structural connective. In particular, we disallow labels [13], additional structural connectives as in Display Logic [10] or nested sequents [3,15]. Our motivating question is thus *which modal logics can be captured by pure sequent calculi*, or more precisely, *what format of pure sequent rules is necessary to capture a given modal logic?* The rules considered here introduce precisely one layer of modalities and fix context formulae. Within this format, we

[⋆] Supported by EPSRC-Project EP/H016317/1.

consider three classes of rules that differ in the way context formulae are handled. The most general of these formats, *rules with context restrictions*, captures not only most standard normal logics but also a number of non-normal logics such as conditional logics, probabilistic or coalition logic, including all non-iterative logics, and is amenable to reasonably simple purely syntactical criteria sufficient for cut elimination and the subformula property [12].

Our main contribution is twofold. We establish a correspondence between syntactical shapes of axioms in a Hilbert system and rule formats for logical rules in a sequent calculus (with cut) so that both formalisms axiomatise the same logic. We then use this correspondence to obtain impossibility results. Our first such result merely illustrates the techniques and shows that K4 cannot be axiomatised by shallow sequent rules (with cut, and *a posteriori*, also without cut). We then show that there cannot be a cut-free system of rules with context restrictions for S5 where the rules additionally satisfy a permutability condition that we call *mixed cut closure*. This condition allows to permute cuts between principal and context formulae, and is present in virtually all pure sequent systems for modal logics (e.g. [16] or [9] for the dual case of tableau systems). While this does not show that there cannot be a complete and cut-free pure sequent system for S5, we still obtain a dichotomy result: either, S5 cannot be axiomatised by rules with restrictions and a more general rule format and/or proof theoretic structure is needed, or else the rules of a complete, cut-free system do not allow to permute principal/context cuts (which we consider to be highly unlikely). While clearly more work is needed to establish a complete hierarchy of calculi and rule formats, together with associated impossibility results that we hope to inspire, we are not aware of any other formal impossibility results, or proof-theoretic hierarchy of modal sequent systems to date.

Technically, our methods are purely syntactical and we only require logics to be monotone. Our systems are extensions of the propositional system G3cp of [16] and we allow contraction throughout so that our results implicitly extend to systems where contraction has been absorbed into the modal rules (e.g. S4 given in *op.cit.*). Rules with restrictions have been used previously in [12] which provides a formal translation from non-nested (Hilbert) axioms to rules and the idea for the translation of nested axioms as well as sufficient syntactic criteria for cut elimination. The formal translation for nested axioms, the converse translation and the impossibility results (for which the translation is crucial) are new.

Related Work. Our work is close to [5] in spirit where axioms for substructural logics are translated into (structural) sequent rules and a hierarchy of substructural logics is presented (together with a semantic cut-admissibility proof). For display logic, [10] gives a back-and-forth translation between so-called primitive axioms for normal modal logics and display rules using extra structural connectives and, implicitly, also tense logic. Similarly, [13] uses labels to capture first-order frame conditions over normal modal logics definable by geometric sequents. Finally, [6] translates between paraconsistent logics and sequent calculi in a similar way, using different syntactical formats (and construct cut-free sequent systems via non-deterministic semantics). We are not aware of any translations between modal axioms and logical rules for pure sequent calculi or any formal impossibility results.

2 Preliminaries

We write $\mathbb{N}$ for the set of non-negative integers and $\mathfrak{P}(S)$ for the power set of a set S. We assume familiarity with the standard notions of modal logic as given e.g. in [2,4]. Throughout the paper we fix a set Λ of *modalities* which we take to be unary for expository reasons, and a countable set Var of propositional variables. The set $\mathcal{F}$ of *formulae* is then defined using the modalities $\heartsuit \in \Lambda$, variables $p \in \mathsf{Var}$ and the boolean connectives $\bot, \wedge, \vee, \rightarrow$:

$$\mathcal{F} \ni A ::= p \mid \bot \mid A \vee A \mid A \wedge A \mid A \rightarrow A \mid \heartsuit A$$

As usual we abbreviate $A \rightarrow \bot$ by $\neg A$. Finite sequences $A_1, \ldots, A_n$ of formulae are denoted by $\boldsymbol{A}$. We write $\bigwedge_{i=1}^{n} A_i$ or $\bigwedge \boldsymbol{A}$ for the iterated conjunction $A_1 \wedge \ldots \wedge A_n$ and similarly for iterated disjunctions. The empty conjunction is $\bot \rightarrow \bot$ and similarly, the empty disjunctions is $\bot$. If F is a set of formulae we write $\Lambda(F)$ for the set $\{\heartsuit A \mid \heartsuit \in \Lambda,\ A \in F\}$ and $\mathsf{Prop}(\boldsymbol{p})$ for the set of propositional formulae in the variables $\boldsymbol{p}$. If $\sigma : \mathsf{Var} \rightarrow \mathcal{F}$ is a substitution and A is a formula we write $A\sigma$ for the result of uniformly substituting every variable in A according to σ. For a formula A, $\mathsf{Sf}(A)$ is the set of subformulae of A and $\mathsf{var}(A)$ for the set of variables occurring in A. A formula is *rank-1* if every variable occurs under exactly one modality and *non-iterative* if every variable occurs under at most one modality.

If F is a set of formulae, a *multiset over* F is a map $F \rightarrow \mathbb{N}$ with finite support. The notion of union between multisets extends the set-theoretic union in the obvious way and we write Γ, Δ for the union of the multisets Γ and Δ. The set $\mathcal{S}(F)$ of *sequents over* F is the set of tuples of multisets over F, written as $\Gamma \Rightarrow \Delta$. Application of a substitution and the notion of the set of subformulae extend to multisets and sequents of formulae in the obvious way. We use the system $\mathsf{G3cp}$ from [14,16] with general axioms (see Table 1) for the underlying (classical) propositional logic and denote this system by G. As structural rules, We consider are Cut and $\mathsf{Con} = \{\mathsf{Con_L}, \mathsf{Con_R}\}$ (see Table 1) and we write $\mathsf{G[CutCon]}$ if a result holds for G and any extension with Cut or Con. For $\mathcal{A} \subseteq \mathcal{F}$ we write $\mathsf{G[CutCon]} + \mathcal{A}$ for the sequent system with *groundsequents* $\Rightarrow A$ for $A \in \mathcal{A}$. *Derivability in* $\mathsf{G[CutCon]} + \mathcal{A}$ is defined as derivability in $\mathsf{G[CutCon]}$ from assumptions $\{\Gamma \Rightarrow A\sigma, \Delta \mid A \in \mathcal{A}, \Gamma \Rightarrow \Delta \in \mathcal{S}(\mathcal{F}), \sigma \text{ a substitution}\}$ and denoted by $\vdash_{\mathsf{G[CutCon]}+\mathcal{A}}$. In case $\mathcal{A} = \emptyset$ we also write $\vdash_{\mathsf{G[CutCon]}}$.

We consider modal logics given by a *Hilbert system*, i.e. containing set $\mathcal{A} \subseteq \mathcal{F}$ of *axioms*, all (classical) propositional tautologies, and closed under modus ponens (from $A \rightarrow B$ and A infer B) and uniform substitution (from A infer $A\sigma$) as well as monotonicity (from $A \rightarrow B$ infer $\heartsuit A \rightarrow \heartsuit B$) for all $\heartsuit \in \Lambda$.

Table 1. Rules for the standard systems $\mathsf{G}, \mathcal{R}_{\mathsf{K}}$ and $\mathcal{R}_{\mathsf{S5}}$ and the structural rules

$$\frac{}{\Gamma, A \Rightarrow A, \Delta}\ \mathcal{A} \qquad \frac{}{\Gamma, \bot \Rightarrow \Delta}\ \bot_{\mathsf{L}}$$

$$\frac{\Gamma, A \Rightarrow \Delta \quad \Gamma, B \Rightarrow \Delta}{\Gamma, A \lor B \Rightarrow \Delta}\ \lor_{\mathsf{L}} \qquad \frac{\Gamma, A, B \Rightarrow \Delta}{\Gamma, A \land B \Rightarrow \Delta}\ \land_{\mathsf{L}} \qquad \frac{\Gamma, B \Rightarrow \Delta \quad \Gamma \Rightarrow A, \Delta}{\Gamma, A \to B \Rightarrow \Delta}\ \to_{\mathsf{L}}$$

$$\frac{\Gamma \Rightarrow A, B, \Delta}{\Gamma \Rightarrow A \lor B, \Delta}\ \lor_{\mathsf{R}} \qquad \frac{\Gamma \Rightarrow A, \Delta \quad \Gamma \Rightarrow B, \Delta}{\Gamma \Rightarrow A \land B, \Delta}\ \land_{\mathsf{R}} \qquad \frac{\Gamma, A \Rightarrow B, \Delta}{\Gamma \Rightarrow A \to B, \Delta}\ \to_{\mathsf{R}}$$

$$\frac{A_1, \dots, A_n \Rightarrow B}{\Gamma, \Box A_1, \dots, \Box A_n \Rightarrow \Box B, \Delta}\ R_{\mathsf{K}_n} \qquad \frac{\Gamma, A \Rightarrow \Delta}{\Gamma, \Box A \Rightarrow \Delta}\ R_{\mathsf{T}} \qquad \frac{\Box\Gamma \Rightarrow A, \Box\Delta}{\Sigma, \Box\Gamma \Rightarrow \Box A, \Box\Delta, \Pi}\ R_5$$

$$\frac{\Gamma \Rightarrow A, \Delta \quad \Sigma, A \Rightarrow \Pi}{\Gamma, \Sigma \Rightarrow \Delta, \Pi}\ \mathsf{Cut} \qquad \frac{\Gamma, A, A \Rightarrow \Delta}{\Gamma, A \Rightarrow \Delta}\ \mathsf{Con}_{\mathsf{L}} \qquad \frac{\Gamma \Rightarrow A, A, \Delta}{\Gamma \Rightarrow A, \Delta}\ \mathsf{Con}_{\mathsf{R}}$$

$$\mathsf{G} := \{\mathcal{A}, \bot_{\mathsf{L}}, \lor_{\mathsf{L}}, \lor_{\mathsf{R}}, \land_{\mathsf{L}}, \land_{\mathsf{R}}, \to_{\mathsf{L}}, \to_{\mathsf{R}}\} \qquad \mathcal{R}_{\mathsf{K}} := \{R_{\mathsf{K}_n} \mid n \geq 0\} \qquad \mathcal{R}_{\mathsf{S5}} := \{R_{\mathsf{T}}, R_5\}$$

3 Rules with Restrictions

We briefly recapitulate the notion of a rule with context restrictions from [12].

Definition 1. For a set F of formulae a *context restriction over* F (or *restriction*) is a tuple $\langle F_1; F_2 \rangle \in \mathfrak{P}(F)^2$. For a restriction $\mathcal{C} = \langle F_1; F_2 \rangle$ and a sequent $\Gamma \Rightarrow \Delta$ the *restriction of* $\Gamma \Rightarrow \Delta$ *according to* $\mathcal{C}$ is the sequent $(\Gamma \Rightarrow \Delta)\restriction_{\mathcal{C}} = \Gamma\restriction_{F_1} \Rightarrow \Delta\restriction_{F_2}$ consisting of all those formulae from Γ resp. Δ which are substitution instances of formulae in F_1 resp. F_2. We write $\mathfrak{C}(F)$ for the set of all restrictions over F.

Example 2. Let $\Gamma \Rightarrow \Delta = \Box(A \land B), C \lor D \Rightarrow \Box E, F$. Then:

1. for $\mathcal{C}_\emptyset := \langle \emptyset; \emptyset \rangle$ we have $(\Gamma \Rightarrow \Delta)\restriction_{\mathcal{C}_\emptyset} = \ \Rightarrow$
2. for $\mathcal{C}_{\mathsf{id}} := \langle \{p\}; \{p\} \rangle$ we have $(\Gamma \Rightarrow \Delta)\restriction_{\mathcal{C}_{\mathsf{id}}} = \Gamma \Rightarrow \Delta$
3. for $\mathcal{C}_{\mathsf{S5}} := \langle \{\Box p\}; \{\Box p\} \rangle$ we have $(\Gamma \Rightarrow \Delta)\restriction_{\mathcal{C}_{\mathsf{S5}}} = \Box(A \land B) \Rightarrow \Box E$.

Definition 3. A *rule with context restrictions* (or simply a *rule*) is a tuple $\mathcal{P}/\Sigma \Rightarrow \Pi$ where $\mathcal{P} \subseteq \mathcal{S}(\mathsf{Var}) \times \mathfrak{C}(\mathcal{F})$ is the set of *premisses* with associated context restrictions, and $\Sigma \Rightarrow \Pi \in \mathcal{S}(\Lambda(\mathsf{Var}))$ are the *principal formulae*, such that no variable occurs twice in the principal formulae and every variable occurs in the principal formulae iff it occurs in at least one of the premisses. An *instance* of a rule R is given by a substitution $\sigma : \mathsf{Var} \to \mathcal{F}$ and a *context* $\Gamma \Rightarrow \Delta \in \mathcal{S}(\mathcal{F})$ and is written as

$$\frac{\{\Gamma\restriction_{F_1}, \Theta\sigma \Rightarrow \Delta\restriction_{F_2}, \Upsilon\sigma \mid (\Theta \Rightarrow \Upsilon; \langle F_1; F_2 \rangle) \in \mathcal{P}\}}{\Gamma, \Sigma\sigma \Rightarrow \Delta, \Pi\sigma}\ .$$

We assume that every set of rules is closed under injective renaming of variables (respecting the variable conditions) and for every $\heartsuit \in \Lambda$ includes the *monotonicity rule* $\mathsf{Mon} = \{(p \Rightarrow q; \mathcal{C}_\emptyset)\}/\heartsuit p \Rightarrow \heartsuit q$. Notions concerning derivability in $\mathsf{G}[\mathsf{CutCon}] + \mathcal{A}$ are extended in the obvious way to a set $\mathcal{R}$ of rules using the notation $\vdash_{\mathsf{G}[\mathsf{CutCon}]\mathcal{R}+\mathcal{A}}$. A rule is called *shallow* if the only restrictions occurring in it are $\mathcal{C}_\emptyset$ or $\mathcal{C}_{\mathsf{id}}$. It is *one-step* if only the restriction $\mathcal{C}_\emptyset$ occurs in it.

Example 4. Most standard modal rules fit the format of rules with restrictions:

1. R_{K_n} is the rule with restrictions $\{(p_1, \ldots, p_n \Rightarrow q; \mathcal{C}_\emptyset)\}/\Box p_1, \ldots, \Box p_n \Rightarrow \Box q$
2. The rule R_{T} is the rule with restrictions $\{(p \Rightarrow\ ; \mathcal{C}_{\mathsf{id}})\}/\Box p \Rightarrow$
3. The rule R_{5} is the rule with restrictions $\{(\ \Rightarrow p; \mathcal{C}_{\mathsf{S5}})\}/\ \Rightarrow \Box p$.

The rules R_{T} and R_{K_n} are also shallow rules, rule R_{K_n} also is a one-step rule.

By additionally copying the principal formulae into the premisses in the spirit of G3-systems such as the system G3s for S4 in [16] Contraction can be made admissible in cut-free rule sets [12, Thm. 16].

4 From Axioms to Rules

We now give an extension and formalisation of the translation from axioms in a Hilbert-system to rules in a sequent system from [11,12] which uses the following standard argument.

Lemma 5. *For every set $\mathcal{A}$ of axioms and sequent $\Gamma \Rightarrow \Delta$ we have $\vdash_{\mathsf{GCutCon}+\mathcal{A}} \Gamma \Rightarrow \Delta$ iff $\vdash_{\mathcal{HA}} \bigwedge \Gamma \to \bigvee \Delta$.*

The following equivalence between rules and axioms therefore suffices:

Definition 6. If $\mathcal{R}$ is a set of rules, then a set $\mathcal{A}$ of axioms and a set $\mathcal{R}_\mathcal{A}$ of rules are *equivalent over* $\mathsf{G}\mathcal{R}$ if for every sequent $\Gamma \Rightarrow \Delta$ we have $\vdash_{\mathsf{G}\mathcal{R}\mathsf{CutCon}+\mathcal{A}} \Gamma \Rightarrow \Delta$ iff $\vdash_{\mathsf{G}\mathcal{R}\mathsf{CutCon}\mathcal{R}_\mathcal{A}} \Gamma \Rightarrow \Delta$.

The main idea for the translation is to take a substitution instance of a non-iterative axiom, where the substitution formulae satisfy certain restrictions, and use the techniques from [11] to turn the non-iterative axiom into a sequent rule. The restrictions guarantee that when substituting the formulae in the rule we get a so-called proto rule, i.e. a rule with a fixed number of context formulae that can be turned into a rule under certain conditions. While this class of axioms might seem restrictive at first, we show in Section 5 that it is sufficient to construct every rule with restrictions. In a first step inspired by [5] we consider conjunctive normal forms of formulae where the polarities of subformulae are controlled.

Definition 7. Let C_ℓ, C, C_r be sets of formulae. The sets $\mathcal{F}_\ell(C_\ell, C, C_r)$ and $\mathcal{F}_r(C_\ell, C, C_r)$ of *left resolvable* (resp. *right resolvable*) *formulae for* (C_ℓ, C, C_r) are given by the following grammar with starting variable P_ℓ (resp. P_r):

$$P_\ell ::= P_\ell \lor P_\ell \mid P_\ell \land P_\ell \mid P_r \to P_\ell \mid A_\ell \mid B \mid \bot \quad \text{where } A_\ell \in C_\ell, B \in C$$
$$P_r ::= P_r \lor P_r \mid P_r \land P_r \mid P_\ell \to P_r \mid A_r \mid B \mid \bot \quad \text{where } A_r \in C_r, B \in C\ .$$

Using invertibility of the propositional rules in every calculus containing GCutCon it is then not hard to see that we can decompose a right resolvable formula into clauses resp. sequents in the following way.

Lemma and Definition 8. *If A is a right (resp. left) resolvable formula with context formulae in (C_ℓ, C, C_r), then A (resp. $A \to \bot$) is equivalent to a set $\mathcal{C}_A$ of clauses of the form $\bigwedge_{i=1}^n A_i \to \bigvee_{j=1}^m B_j$ with $n, m \geq 0$ and $A_i \in C_\ell \cup C$ and $B_j \in C_r \cup C$ corresponding to sequents $A_1, \dots, A_n \Rightarrow B_1, \dots, B_m$. We then call $\bigwedge \mathcal{C}_A$ a (C_ℓ, C, C_r)-*normal form *of A.* □

The notions of resolvable formulae allow us now to specify the syntactic requirement for a formula to be expressible as a rule. For conjunctions of such formulae we treat each clause separately.

Definition 9. Let $V \subseteq \mathsf{Var}$ and let $C_\ell, C_r \subseteq \mathcal{F}$ such that for all formulae $A, B \in C_\ell \cup C_r$ we have $\mathsf{var}(A) \cap \mathsf{var}(B) = \emptyset$ and $\mathsf{var}(A) \cap V = \emptyset$. A formula is *translatable with context formulae in (C_ℓ, C_r) and variables in V* if it has the form $\bigwedge_{i=1}^n A_i \to \bigvee_{j=1}^m B_j$ with $A_i \in C_\ell \cup \Lambda(\mathcal{F}_r(C_\ell, C, C_r)) \cup V$ for all $i \leq n$ and $B_j \in C_r \cup \Lambda(\mathcal{F}_\ell(C_\ell, C, C_r)) \cup V$ for all $j \leq m$ and if furthermore every formula in $C_\ell \cup C_r$ occurs at most once on the top level of the formula (i.e. not in the scope of a modality) and occurs on the top level iff it occurs in the scope of a modality. A formula is *translatable* if it is translatable for some sets C_ℓ, C_r of context formulae and variables V.

Intuitively, a formula is translatable with context formulae in (C_ℓ, C_r) and variables in V if it is a substitution instance of a non-iterative formula, where subformulae in C_ℓ (resp. C_r) are responsible for the nesting and behave in a way that they can be turned into context formulae on the left (resp. right) of the corresponding rule. The variables in V are used in the construction of the principal formulae of the rule. A formula might be translatable in more than one way.

Example 10.
1. The axiom $(\mathsf{4})$ $\Box q \to \Box\Box q$ is translatable with context formulae in $(\{\Box q\}, \emptyset)$ and variables in $\emptyset$.
2. The axiom $(\mathsf{5})$ $\neg\Box q \to \Box\neg\Box q$ is translatable with context formulae in $(\emptyset, \{\Box q\})$ or $(\{\neg\Box q\}, \emptyset)$ and variables in $\emptyset$.
3. The axiom (T) $\Box q \to q$ is translatable with context formulae in $(\emptyset, \emptyset)$ and variables in $\{q\}$.

Since such an axiom contains a fixed number of context formulae, in the first step instead of rules we only get so-called proto-rules, i.e. rules with a fixed context.

Definition 11. Given a rule with restrictions $R = \mathcal{P}/\Sigma \Rightarrow \Pi$ a *proto-rule for* R is a tuple $(R; \Gamma \Rightarrow \Delta)$ given by a context $\Gamma \Rightarrow \Delta \in \mathcal{S}(\mathcal{F})$ such that

1. no propositional variable occurs more than once in $\Gamma \Rightarrow \Delta$
2. no propositional variable occurs both in $\Gamma \Rightarrow \Delta$ and R
3. if $\Gamma \Rightarrow \Delta \neq\ \Rightarrow$, then $(\Gamma \Rightarrow \Delta)\restriction_{\mathcal{C}} \neq\ \Rightarrow$ for every restriction $\mathcal{C}$ of R.

We often leave the context implicit and write $\widehat{R}$ for a proto-rule for R. An *instance* of a proto-rule $\widehat{R} = (R; \Gamma \Rightarrow \Delta)$ is given by a substitution $\sigma : \mathsf{Var} \to \mathcal{F}$ and a context $\Theta \Rightarrow \Xi$ where $(\Theta \Rightarrow \Xi)\restriction_{\mathcal{C}} =\ \Rightarrow$ for every restriction $\mathcal{C}$ of R, and is the same as the instance of R with substitution σ and context $\Gamma\sigma, \Theta \Rightarrow \Delta\sigma, \Xi$ according to Definition 3. Derivability using proto-rules is defined as expected.

Informally, the difference between rules and proto-rules is that in proto-rules the premisses *including the context* are fixed up to substitution, while in rules also the *number* of the context formulae in the premisses may vary.

Theorem 12. *Every translatable axiom A is equivalent over* GMonCutCon *to a proto-rule.*

Proof (Sketch). We start with a translatable axiom A, i.e. a formula of the form

$$\bigwedge_{P\in\boldsymbol{P}_\ell} \heartsuit_P P \wedge \bigwedge_{q\in\boldsymbol{q}_\ell} q \wedge \bigwedge_{L\in\boldsymbol{L}} L \to \bigvee_{P\in\boldsymbol{P}_r} \heartsuit_P P \vee \bigvee_{q\in\boldsymbol{q}_r} q \vee \bigvee_{R\in\boldsymbol{R}} R$$

where for some $C_\ell, C_r \subseteq \mathcal{F}$ and $V \subseteq \mathsf{Var}$ satisfying the restrictions of Definition 9 we have $\boldsymbol{P}_\ell \subseteq \mathcal{F}_r(C_\ell, V, C_r), \boldsymbol{P}_r \subseteq \mathcal{F}_\ell(C_\ell, V, C_r)$ and $\boldsymbol{q}_\ell \cup \boldsymbol{q}_r \subseteq V$ and $\boldsymbol{L} \subseteq C_\ell, \boldsymbol{R} \subseteq C_r$. After turning this into the ground sequent $\Rightarrow A$ and resolving propositional logic using GCutCon we replace the formulae P under the modalities with fresh variables t_P and add as premisses all the sequents $t_P \Rightarrow P$ for $P \in \boldsymbol{P}_\ell$ and $P \Rightarrow t_P$ for $P \in \boldsymbol{P}_r$ to get

$$\frac{\{t_P \Rightarrow P \mid P \in \boldsymbol{P}_\ell\} \qquad \{P \Rightarrow t_P \mid t_P \in \boldsymbol{P}_r\}}{\{\heartsuit_P t_P \mid P \in \boldsymbol{P}_\ell\}, \boldsymbol{q}_\ell, \boldsymbol{L} \Rightarrow \{\heartsuit_P t_P \mid P \in \boldsymbol{P}_r\}, \boldsymbol{q}_r, \boldsymbol{R}}\,.$$

Now we replace the sequents $t_P \Rightarrow P$ (resp. $P \Rightarrow t_P$) by the sequents corresponding to the clauses of a (C_ℓ, C, C_r)-normal form of the formulae $t_P \to P$ (resp. $P \to t_P$) according to Lemma 8. The lemma ensures that in this step a context formula ends up on the left (resp. right) hand side of one of these premisses if and only if it is in $\boldsymbol{L}$ (resp. $\boldsymbol{R}$). Now we apply a standard trick and introduce two fresh variables r, s and replace variables $q \in \boldsymbol{q}_\ell$ (resp. $\boldsymbol{q}_r$) with premisses $r \Rightarrow q, s$ (resp. $r, q \Rightarrow s$). Finally, we eliminate the occurrences of the variables $\boldsymbol{q}_\ell \cup \boldsymbol{q}_r$ in the premisses by performing all possible cuts between sequents in the premisses with cut formula $q \in \boldsymbol{q}_\ell \cup \boldsymbol{q}_r$. The resulting proto-rule then is seen to be equivalent to the axiom A, where for the last step and the fact that premisses ensuring $t_P \to P$ instead of $t_P \leftrightarrow P$ suffice we make use of the monotonicity rules and Cut and the techniques of [12, Lemma 9]. $\square$

In order to produce rules instead of proto-rules we note that in presence of the propositional rules and Cut a rule where a formula A occurs in the left component of a context restriction is equivalent to a set of proto-rules with a conjunction $\bigwedge_{i\leq n} A^i$ in place of A, where A^i results from A by renaming the variables to fresh ones, and similarly using disjunction for formulae in the right component. This motivates the next definition.

Definition 13. Let the formula B be translatable with context formulae in $(\{C_1, \dots, C_n\}, \{D_1, \dots, D_m\})$ and variables in V. For $s_1, \dots, s_n, t_1, \dots, t_m \geq 0$ the formula $B_{s_1,\dots,s_n,t_1,\dots,t_m}$ is constructed from B by replacing every occurrence of a formula C_k with $\bigwedge_{i=1}^{s_k} C_k^i$ and every occurrence of a formula D_ℓ with $\bigvee_{i=1}^{t_\ell} D_\ell^i$, where the formulae C_k^i, D_ℓ^i result from C_k (resp. D_ℓ) by injectively renaming its variables $\boldsymbol{p}$ to fresh variables $\boldsymbol{p}^i$. Then an *ω-set for B* is a set $\{B_{s_1,\dots,s_n,t_1,\dots,t_m} \mid s_i \geq 0, t_j \geq 0 \text{ for } i \leq n, j \leq m\}$.

Example 14. The set $\{\neg \bigvee_{i=1}^{n} \Box q^i \to \Box\neg \bigvee_{i=1}^{n} \Box q^i \mid n \geq 0\}$ is an ω-set for the axiom (5) $\neg\Box q \to \Box\neg\Box q$.

Intuitively the formulae in an ω-set for B are constructed from B by substituting each context formula with finite (possibly empty) conjunctions resp. disjunctions of copies of this formula with fresh variables. By Theorem 12 each of these axioms translates into a proto-rule and since the context formulae are in the same positions together they are equivalent to a rule with restrictions.

Corollary 15. *Every ω-set for a translatable formula is equivalent over the rule set* $\mathsf{GMonCutCon}$ *to a rule with restrictions.* $\Box$

Example 16. Translating the ω-set $\{\neg \bigvee_{i=1}^{n} \Box q^i \to \Box\neg \bigvee_{i=1}^{n} \Box q^i \mid n \geq 0\}$ for the axiom (5) $\neg\Box q \to \Box\neg\Box q$ yields the rule $\{(\Rightarrow p; \langle\emptyset; \{\Box p\}\rangle)\}/ \Rightarrow \Box p$.

We would like to translate axioms instead of ω-sets of axioms. This is possible if the conjunctions resp. disjunctions can be pushed into the context formulae.

Definition 17. A formula A with free variables $p_1, \ldots, p_n = \boldsymbol{p}$ is *left normal* for a set $\mathcal{R}$ of rules if for every $k \geq 0$ there are formulae $B_1, \ldots, B_n$ such that $\vdash_{\mathsf{GCutCon}\mathcal{R}} \Rightarrow \bigwedge_{i=1}^{k} A^i \leftrightarrow A\sigma_{\boldsymbol{p}}^{\boldsymbol{B}}$ where A^i is the result of injectively renaming the propositional variables $\boldsymbol{p}$ in A to fresh variables $\boldsymbol{p}^i$ and $\sigma_{\boldsymbol{p}}^{\boldsymbol{B}}$ is the substitution given by $\sigma(p_j) = B_j$ and $\sigma(x) = x$ for $x \notin \boldsymbol{p}$. A formula A is *right normal* if $A \to \bot$ is left normal. A context restriction $\langle F_1; F_2\rangle$ is *normal* if every formula in F_1 (resp. F_2) is left (resp. right) normal.

Adding a translatable axiom where all context formulae are left resp. right normal for $\mathcal{R}$ is equivalent over $\mathcal{R}$ to adding its ω-set, and thus we get:

Theorem 18. *Let B be a translatable formula with context formulae in (C_ℓ, C_r) and variables in V such that the formulae in C_ℓ are left normal in $\mathcal{R}$ and those in C_r are right normal for $\mathcal{R}$. Then B is equivalent over* $\mathsf{G}\mathcal{R}\mathsf{CutCon}$ *to a rule with restrictions.* $\Box$

Since variables are both left and right normal for every rule set this immediately yields the translation result for non-iterative and rank-1 axioms from [11].

Corollary 19. *Every non-iterative (resp. rank-1) axiom is equivalent over the rule set* $\mathsf{GMonCutCon}$ *to a set of shallow (resp. one-step) rules.* $\Box$

Example 20. 1. The context formula $\Box q$ is left normal for $\mathcal{R}_{\mathsf{K}}$ and thus translating the axiom (4) $\Box q \to \Box\Box q$ using Theorem 18 yields the well-known rule $R_4 = \{(\Rightarrow p; \langle\{\Box p\}; \emptyset\rangle)\}/ \Rightarrow \Box p$.
2. Similarly, translating the axiom (T) yields the standard rule R_{T} from Table 1.
3. By propositional reasoning and the axioms of K adding both axioms (4) and (5) is equivalent to adding the set $\{\bigwedge_{i=1}^{n} \Box q_\ell^i \wedge \neg \bigvee_{j=1}^{m} \Box q_r^j \to \Box(\bigwedge_{i=1}^{n} \Box q_\ell^i \wedge \neg \bigvee_{j=1}^{m} \Box q_r^j) \mid m, n \geq 0\}$, which is an ω-set for the axiom $\Box q_\ell \wedge \neg\Box q_r \to \Box(\Box q_\ell \wedge \neg\Box q_r)$ under translatability with context formulae in $(\{\Box q_\ell\}, \{\Box q_r\})$. By Corollary 15 this set translates into the standard rule R_5 from Table 1.

5 From Rules to Axioms

The results of the previous section raise the question whether the format of ω-sets for axioms is really necessary. It turns out that the format is both necessary and sufficient in the sense that an axiom can be translated into a rule with restrictions if and only if adding the axiom is equivalent to adding an ω-set. We show this by translating rules with restrictions back into ω-sets of axioms. The first step is to bring the premisses of the rules into a normal form.

Lemma and Definition 21. *Every rule with restrictions is equivalent to a rule in* standard form *over* $\mathsf{GMonCutCon}$*, i.e. a to rule with restrictions* $\mathcal{P}/\Sigma \Rightarrow \Pi$ *where*

1. *if* $(\Rightarrow p; \mathcal{C}_\emptyset) \in \mathcal{P}$ *then there is no premiss* $(\Gamma, p \Rightarrow \Delta; \mathcal{C}) \in \mathcal{P}$ *and no premiss* $(\Rightarrow p; \mathcal{C}) \in \mathcal{P}$ *with* $\mathcal{C} \neq \mathcal{C}_\emptyset$
2. *for all* $q_1, \dots, q_n, p \in \mathsf{Var}$*: if* $(q_1, \dots, q_n \Rightarrow p; \mathcal{C}) \in \mathcal{P}$ *and* $(\Rightarrow q_i; \mathcal{C}_i) \in \mathcal{P}$ *for all* $i \leq n$*, then* $(\Rightarrow p; \mathcal{C} \cup \bigcup_{i=1}^{n} \mathcal{C}_i) \in \mathcal{P}$*.*
3. *no variable occurs both on the left hand side of a premiss and on the right hand side of a (possibly different) premiss.*

Proof (Sketch). For the first claim if there is are premisses $(\Rightarrow p; \mathcal{C}_\emptyset)$ and $(\Gamma, p \Rightarrow \Delta; \mathcal{C})$ in $\mathcal{P}$, due to the presence of Cut we may replace the latter with $(\Gamma \Rightarrow \Delta; \mathcal{C})$. Also, premisses $(\Rightarrow p; \mathcal{C})$ with $\mathcal{C} \neq \mathcal{C}_\emptyset$ are derived by (admissible) Weakening from $(\Rightarrow p; \mathcal{C}_\emptyset)$ and thus can be omitted. For the second claim we simply add the missing premisses, which in the presence of Cut yields an equivalent rule. For the last claim we use the fact that all our rule sets include Mon and replace a rule by the cut between this rule and Mon, see [12, Lemma 9]. $\square$

For the rest of this section we assume w.l.o.g. that all rules are in standard form. Again instead of translating rules directly we first work with sets of proto-rules. Given a proto-rule we now turn its premisses and conclusion into formulae.

Definition 22. Let $R = \mathcal{P}/\Sigma \Rightarrow \Pi$ be a rule and $\widehat{R}$ a proto-rule for R given by the context $\Gamma \Rightarrow \Delta$. The formulae $\mathsf{Prem}_{\widehat{R}}$ and $\mathsf{Concl}_{\widehat{R}}$ are defined by

$$\begin{aligned}\mathsf{Prem}_{\widehat{R}} &= \bigwedge\nolimits_{(\Theta \Rightarrow \Xi; \langle F_1; F_2 \rangle) \in \mathcal{P}} (\bigwedge \Gamma{\restriction_{F_1}} \wedge \bigwedge \Theta \to \bigvee \Xi \vee \bigvee \Delta{\restriction_{F_2}})\\ \mathsf{Concl}_{\widehat{R}} &= \bigwedge \Gamma \wedge \bigwedge \Sigma \to \bigvee \Pi \vee \bigvee \Delta\end{aligned}$$

Then by propositional reasoning it is clear that the premisses of a proto-rule $\widehat{R}$ (resp. its conclusion) are derivable if and only if the sequent $\Rightarrow \mathsf{Prem}_{\widehat{R}}$ (resp. $\Rightarrow \mathsf{Concl}_{\widehat{R}}$) is derivable. To turn these formulae into an axiom we make use of the notion of a projective formula, see e.g. [8].

Definition 23. A formula $A \in \mathcal{F}(\Lambda)$ is *projective* if there is a substitution $\sigma : \mathsf{Var} \to \mathcal{F}(\Lambda)$ such that $\vdash_{\mathsf{GMonCutCon}} \Rightarrow A\sigma$; and for all $p \in \mathsf{var}(A)$ we have $\vdash_{\mathsf{GMonCutCon}} A \Rightarrow p \leftrightarrow p\sigma$. Such a substitution *witnesses projectivity of* A.

Given a proto-rule $\widehat{R}$ once we have a substitution witnessing the projectivity of the formula $\mathsf{Prem}_{\widehat{R}}$ we are done using the following Lemma.

Lemma 24. *If $\widehat{R}$ is a proto-rule and σ a substitution witnessing projectivity of $\mathsf{Prem}_{\widehat{R}}$, then the axiom $\mathsf{Concl}_{\widehat{R}}\sigma$ is equivalent to $\widehat{R}$ over every rule set $\mathcal{R}$.*

Proof. In a first step an induction on the complexity of the formula B shows that if σ witnesses projectivity of a formula A, then for every formula B we have $\vdash_{\mathsf{GMonCutCon}} A \Rightarrow B \leftrightarrow B\sigma$. To see that the proto-rule $\widehat{R}$ is derivable using the axiom assume that we have derivations of its premisses. Then by propositional logic we also have $\vdash_{\mathsf{G}\mathcal{R}\mathsf{CutCon}} \Rightarrow \mathsf{Prem}_{\widehat{R}}$. Thus by projectivity and Cut we get $\vdash_{\mathsf{G}\mathcal{R}\mathsf{CutCon}} \Rightarrow \mathsf{Concl}_{\widehat{R}}\sigma \to \mathsf{Concl}_{\widehat{R}}$ which together with the ground sequent $\Rightarrow \mathsf{Concl}_{\widehat{R}}\sigma$ yields $\Rightarrow \mathsf{Concl}_{\widehat{R}}$. Now resolving the propositional connectives using $\mathsf{GCutCon}$ yields the conclusion of R. For the other direction by projectivity we have $\vdash_{\mathsf{G}\mathcal{R}\mathsf{CutCon}} \Rightarrow \mathsf{Prem}_{\widehat{R}}\sigma$, and resolving the propositional connectives and applying the rule R and propositional rules yields $\vdash_{\mathsf{G}\mathcal{R}\mathsf{CutCon}R} \Rightarrow \mathsf{Concl}_{\widehat{R}}\sigma$. □

Using standard techniques [8] we can always construct such a substitution:

Definition 25 (θ). Let $R = \mathcal{P}/\Sigma \Rightarrow \Pi$ be a rule in standard form and $\widehat{R}$ a proto-rule for R given by $\Gamma \Rightarrow \Delta$. Define the substitution $\theta_{\widehat{R}}$ by

$$\theta_{\widehat{R}}(p) := \begin{cases} \top & : \ (\Rightarrow p; \mathcal{C}_\emptyset) \in \mathcal{P} \\ \mathsf{Prem}_{\widehat{R}} \to p & : \ (\Theta \Rightarrow p, \Xi; \mathcal{C}) \in \mathcal{P} \text{ for some } \mathcal{C} \neq \mathcal{C}_\emptyset \text{ and } \Theta \Rightarrow \Xi \\ \mathsf{Prem}_{\widehat{R}} \wedge p & : \ (\Theta, p \Rightarrow \Xi; \mathcal{C}) \in \mathcal{P} \text{ for some } \Theta \Rightarrow \Xi \\ p & : \ \text{otherwise.} \end{cases}$$

Lemma 26. *$\theta_{\widehat{R}}$ witnesses projectivity of $\mathsf{Prem}_{\widehat{R}}$ if the latter is satisfiable.*

The proof is by standard propositional reasoning and gives the desired translation when combined with Lemma 24.

Theorem 27. *Every proto-rule is equivalent to a translatable axiom.*

Proof. By Lemmata 24 and 26 we get equivalence of $\mathsf{Concl}_{\widehat{R}}\theta_{\widehat{R}}$ and $\widehat{R}$. Since by Lemma 21 the rule R was w.l.o.g. in standard form, every variable occurs either only on the left or on the right of premisses and conclusion, which ensures that the axiom $\mathsf{Concl}_{\widehat{R}}\theta_{\widehat{R}}$ is translatable. □

Moreover, since a rule R is equivalent to the set of proto-rules for R we can translate rules into sets of axioms.

Proposition 28. *Every rule R is equivalent to an ω-set for an axiom A_R. If all context restrictions of R are normal, then it is equivalent to a single axiom A_R. If R is a shallow (resp. one-step) rule, then it is equivalent to a non-iterative (resp. rank-1) axiom.*

Proof. It is not hard to see that a rule R is equivalent to the set of proto-rules for it and that the set of translations of these proto-rules is an ω-set. In case all the restrictions are normal, adding the ω-set for the corresponding axiom A_R is equivalent to adding the axiom A_R itself. A close inspection of the translations of shallow and one-step rules together with the fact that all restrictions for such rules are normal yields the last statement. □

In conclusion we thus have the following correspondences between rules and axioms for monotone modalities and normal context formulae resp. restrictions:

rules with context restrictions	$\leftrightsquigarrow$	translatable axioms
shallow rules	$\leftrightsquigarrow$	non-iterative axioms
one-step rules	$\leftrightsquigarrow$	rank-1 axioms

In the first case if not all context formulae are normal the correspondences hold only between rules and ω-sets for translatable axioms.

Example 29. We apply the translation procedure to the rule R_5 from Table 1. Proto-rules for R_5 are given by sequents $\Box r_1, \ldots, \Box r_n \Rightarrow \Box s_1, \ldots, \Box s_m$ with $m, n \geq 0$. Thus we get the formulae $\mathsf{Prem}_{\widehat{R_5}} = \bigwedge_{i=1}^n \Box r_i \to p \vee \bigvee_{j=1}^m \Box s_j$ and $\mathsf{Concl}_{\widehat{R_5}} = \bigwedge_{i=1}^n \Box r_i \to \Box p \vee \bigvee_{j=1}^m \Box s_j$ and the substitution $\theta_{\widehat{R_5}}$ with $\theta_{\widehat{R_5}}(p) = \mathsf{Prem}_{\widehat{R_5}} \to p$ as well as $\theta_{\widehat{R_5}}(r_i) = r_i$ and $\theta_{\widehat{R_5}}(s_j) = s_j$. Thus by Lemmata 24 and 26 the rule R_5 is equivalent to the set of axioms

$$\left\{ \bigwedge_{i=1}^n \Box r_i \to \Box \left((\bigwedge_{i=1}^n \Box r_i \to p \vee \bigvee_{j=1}^m \Box s_j) \to p \right) \vee \bigvee_{j=1}^m \Box s_j \mid m, n \geq 0 \right\},$$

which is an ω-set for the axiom $\Box r \to \Box((\Box r \to p \vee \Box s) \to p) \vee \Box s$ under translatability with context formulae in $(\{\Box r\}, \{\Box s\})$ and variables in $\{p\}$. By propositional reasoning and normality of $\Box$ this ω-set is moreover equivalent to axioms of the form $\Box r \to \Box((\Box r \wedge \neg \bigvee_{j=1}^m \Box s_j) \vee p) \vee \bigvee_{j=1}^m \Box s_j$, and since $\Box$ is monotone we may first replace p by $\top$ and then omit it, yielding axioms $\Box r \to \Box(\Box r \wedge \neg \bigvee_{j=1}^m \Box s_j) \vee \bigvee_{j=1}^m \Box s_j$. Finally, these axioms are equivalent over K to axioms (4) $\Box r \to \Box\Box r$ and $\{\neg \bigvee_{j=1}^m \Box s_j \to \Box \neg \bigvee_{j=1}^m \Box s_j \mid m \geq 0\}$, where the latter is an ω-set for the axiom $(\mathsf{5})$, which over K is equivalent to $(\mathsf{5})$ itself.

Similarly, rule R_{T} translates into the axiom $r \wedge \Box((r \wedge p \to s) \wedge p) \to s$. Again, by arguments as above this axiom can be seen to be equivalent (as an axiom) over K to the standard axiom (T) $\Box A \to A$.

Remark 30. The translations show that Hilbert-style axioms actually correspond to proto-rules instead of rules. Thus from this perspective it would be more natural to consider sequent systems with proto-rules instead of rules.

6 Application: Limitative Results for K4 and S5

The correspondence results of the previous sections open up new possibilities for investigations into the expressive power of sequent rules of a certain format by investigating the limits of the corresponding class of axioms. We exemplify this by establishing two results: the logic K4 cannot be captured by shallow rules (not even with cut), and $\mathsf{S5}$ cannot be captured in a cut-free way by rule with context restrictions with an additional permutation property. Note that the situation for both logics is different: $\mathsf{K4}$ is not captured by any set of shallow rules, with or without cut. For $\mathsf{S5}$ we have exhibited a set of rules that, together with cut, is sound and complete for $\mathsf{S5}$. Here, we show that there cannot be a cut-free complete set of rules which allows permutations of principal/context cuts into the premisses of the first rule. While the result for $\mathsf{K4}$ is intuitively obvious (but still non-trivial to establish formally), our result on $\mathsf{S5}$ can be interpreted to say that rules with restrictions for $\mathsf{S5}$ must be rather exotic, as they do not satisfy permutability which is present in virtually all cut-free rule sets. While this also implies that 'standard' proofs of cut-elimination that rely on propagating cut upwards fail, our considerations are independent of any such proof and just use permutability of rules. For this section we drop the assumption that the rule sets include the rule Mon.

Theorem 31. *There is no set $\mathcal{R}$ of shallow or one-step rules such that $\mathsf{G}\mathcal{R}\mathsf{CutCon}$ is sound and complete for $\mathsf{K4}$.*

Proof. If there was such a set $\mathcal{R}$ of rules, then also $\mathsf{G}\mathcal{R}\mathsf{MonCutCon}$ would be sound and complete for $\mathsf{K4}$. Then by Proposition 28 we would have a non-iterative axiomatisation $\mathcal{A}$ of $\mathsf{K4}$. Now consider the two frames defined by $\mathfrak{F}_1 = (\{a,b,c\},\{(a,b),(b,c),(a,c)\})$ and $\mathfrak{F}_2 = (\{1,2,3,4\},\{(1,2),(2,3),(1,4)\})$. Clearly $\mathfrak{F}_1$ is transitive while $\mathfrak{F}_2$ is not. But given a non-iterative formula which is satisfied in one of the two frames we can find a world of the other frame and an assignment such that the formula is satisfied. That is, precisely the same non-iterative formulae are valid on both frames. On the other hand it is well-known that $\mathsf{K4}$ is the logic of transitive Kripke-frames which is modally definable by the formula $(4)\Box p \to \Box\Box p$. But then (4) must be derivable in $\mathcal{HA}$ and would therefore be valid in $\mathfrak{F}_2$, a contradiction since $\mathfrak{F}_2$ is not transitive. $\Box$

Along these lines it is also possible to show that shallow rules are not expressive enough to characterise symmetric frames or that one-step rules cannot capture reflexivity, thus establishing that the inclusions between the different formats are proper. The situation becomes more interesting if we want to show limitative results for rules with restrictions in general. It is well known that the properties mentioned above together characterise the logic $\mathsf{S5}$, and there is a set of rules which together with the cut rule captures this logic. On the other hand every known cut-free sequent system for $\mathsf{S5}$ seems to involve some extensions of the rule format to facilitate a cut admissibility proof (see e.g. [15] for an overview). This seems to suggest that there is no set of rules with restrictions which is sound and cut-free complete for $\mathsf{S5}$. As argued above, our translation implicitly involves the cut rule which is problematic for proving results about cut-free systems per se. On the other hand assuming a certain permutability of rules gives us more information about the rule set. The first step is to show that every rule set which is sound and cut-free complete for $\mathsf{S5}$ must include certain rules.

Lemma 32. *If $\mathcal{R}$ is a set of modal rules such that $\mathsf{G}\mathcal{R}\mathsf{Con}$ is sound and complete for $\mathsf{S5}$, then there are rules $R_1 = \mathcal{P}_1/\Sigma_1 \Rightarrow \Pi_1$ and $R_2 = \mathcal{P}_2/\Sigma_2 \Rightarrow \Pi_2$ with*

1. *$\Box p \in \Sigma_1$ and $(\ \Rightarrow p)\restriction_{\mathcal{C}} = \ \Rightarrow p$ for a restriction $\mathcal{C}$ of R_1*
2. *$\Box p \in \Sigma_2$ and $\Box q \in \Pi_2$; or $\Box q \in \Pi_2$ and $(\Box p \Rightarrow\ )\restriction_{\mathcal{C}} = \Box p \Rightarrow\ $ for a restriction $\mathcal{C}$ of R_2.*

Proof. For the existence of R_1 we inspect all possible derivations of the $\mathsf{S5}$-valid sequent $\Box p \Rightarrow p$. In the last applied modal rule $\Box p \Rightarrow$ must have been principal due to the rule format. But then $(\Rightarrow p)\restriction_{\mathcal{C}} = \Rightarrow p$ for a restriction $\mathcal{C}$ of this rule since otherwise the sequent $\Box p \Rightarrow$ would be derivable, in contradiction to the fact that $\Box p \Rightarrow \bot$ is not $\mathsf{S5}$-valid.

For the existence of R_2 we consider the possible derivations of the $\mathsf{S5}$-valid sequent $\Box p \Rightarrow \Box(p \vee q)$. If in such a derivation $\Rightarrow \Box(p \vee q)$ is never principal, then since no rule decreases the complexity of a formula and since context formulae cannot change sides, every occurrence of $\Box(p \vee q)$ must come from the weakening context of an axiom. But then essentially the same derivation is used to derive $\Box p \Rightarrow$, a contradiction since $\Box p \Rightarrow \bot$ is not $\mathsf{S5}$-valid. Thus we have a rule where $\Rightarrow \Box(p \vee q)$ is principal. But then for some $\Gamma \Rightarrow \Delta \in \{\Box p \Rightarrow\ ,\ p \Rightarrow\ ,\ \Rightarrow p\}$ we must have $(\Gamma \Rightarrow \Delta)\restriction_{\mathcal{C}} = \Gamma \Rightarrow \Delta$ for some restriction $\mathcal{C}$ of this rule. In the first two cases we are done, and in the last case we can derive $\Rightarrow p \vee \Box(p \vee q)$, again a contradiction since the corresponding formula is not $\mathsf{S5}$-valid. □

The additional permutation property which we demand of the rule sets is suggested by standard cut elimination proofs in the spirit of [7] which rely on a permutation-of-rules argument to transform a derivation with cut into one without. It demands essentially that cuts where the cut formula is principal in the last applied rule in the derivation of one of the premisses and contextual in that of the other premiss can be permuted into the premisses of the latter rule. If the main connective of the cut formula is propositional we also consider an alternative property, G-*invertibility*, which essentially ensures that we can show invertibility of the propositional connectives by permuting applications of the propositional rules below applications of the modal rules. It should be noted that while these properties are motivated by a certain proof technique, they are properties of *rule sets* and as such independent of the proof technique.

Definition 33. A rule set $\mathcal{R}$ is *mixed-cut closed* if whenever $R, Q \in \mathcal{R}$ and R has principal formulae $\Gamma \Rightarrow \Delta, A$ (resp. $A, \Gamma \Rightarrow \Delta$) such that for some restriction $\mathcal{C}_Q$ of Q we have $(\Rightarrow A)\restriction_{\mathcal{C}_Q} = \Rightarrow A$ (resp. $(A \Rightarrow)\restriction_{\mathcal{C}_Q} = A \Rightarrow$), then $(\Gamma \Rightarrow \Delta)\restriction_{\mathcal{C}_Q} = \Gamma \Rightarrow \Delta$ and for every sequent $\Sigma \Rightarrow \Pi$ and restriction $\mathcal{C}_R$ of R we have $(\Sigma \Rightarrow \Pi)\restriction_{\mathcal{C}_R}\restriction_{\mathcal{C}_Q} = (\Sigma \Rightarrow \Pi)\restriction_{\mathcal{C}_R}$. A rule set $\mathcal{R}$ is G-*inverting* if for every restriction $\langle F_0; F_1\rangle$ of a rule in $\mathcal{R}$ and $i \in \{0,1\}$ we have: whenever $A \circ B \in F_i$ with $\circ \in \{\wedge, \vee\}$, then also $A, B \in F_i$ and whenever $A \to B \in F_i$, then also $A \in F_{1-i}$ and $B \in F_i$.

Using the two rules from Lemma 32 together with these properties we obtain a restricted format of the rules in such a rule set.

Lemma 34. *Let $\mathcal{R}$ be a mixed-cut closed set of modal rules such that $\mathsf{G}\mathcal{R}\mathsf{Con}$ is sound and complete for $\mathsf{S5}$ and such that $\mathsf{G}\mathcal{R}$ is mixed-cut closed or $\mathcal{R}$ is G-inverting. Then w.l.o.g. for every restriction $\langle F_1; F_2\rangle$ of a rule in $\mathcal{R}$ we have $F_1 \subseteq \{\Box p, p\}$ and $F_2 \subseteq \{p\}$.*

Proof. Suppose $\mathsf{G}\mathcal{R}$ is mixed-cut closed. If e.g. for a restriction $\langle F_1; F_2\rangle$ of a rule in $\mathcal{R}$ we have $A \wedge B \in F_1$, then by mixed-cut closure of $\mathsf{G}\mathcal{R}$ applied to this rule and the rule $\wedge_\mathsf{R}$ we also have $p \in F_1$ and $p \in F_2$ and thus w.l.o.g. $F_1 = F_2 = \{p\}$. Similarly for the other propositional connectives. If $\Box A \in F_1$, then using mixed-cut closure with R_2 from the Lemma we have $\Box p \in F_1$. Similarly, if $\Box A \in F_2$, then also $p \in F_2$ using R_1 from the Lemma. If on the other hand $\mathcal{R}$ is G-inverting, then in a derivation in $\mathsf{G}\mathcal{R}\mathsf{Con}$ we may permute applications of rules from G below applications of rules from $\mathcal{R}$ in such a way that in the new derivation only variables or modalised formulae occur as contextual formulae of modal rules. Furthermore using mixed-cut closure of $\mathcal{R}$ and the rules R_1, R_2 as above we may replace the modalised formulae in the restrictions by modalised variables. Thus w.l.o.g. we may replace $\mathcal{R}$ by an equivalent set $\mathcal{R}'$ of rules where only formulae of the desired format occur in the restrictions. This preserves mixed-cut closure of $\mathcal{R}$ and G-invertibility. $\Box$

Now we are in a position to use techniques similar to the ones used in proving Theorem 31 to show that the translations of rules in the format specified by the preceding Lemma cannot characterise $\mathsf{S5}$.

Theorem 35. *There is no mixed-cut closed set $\mathcal{R}$ of modal rules such that $\mathsf{G}\mathcal{R}\mathsf{Con}$ is sound and complete for $\mathsf{S5}$ and such that $\mathsf{G}\mathcal{R}$ is mixed-cut closed or $\mathcal{R}$ is G-inverting.*

Proof. Adding the rules Cut and Mon to the rule set $\mathcal{R}$ preserves soundness and completeness. Moreover, bringing the rule set into standard form preserves the restricted format of the rules guaranteed by Lemma 34. The translations of proto-rules for rules of this format have the form

$$q \wedge \Box c \wedge P \wedge \bigwedge_{i=1}^{n} \Box A_i \to \bigvee_{j=1}^{m} \Box B_j \vee r \ ,$$

where P is a conjunction of variables and negated variables, and the A_i and B_j are propositional formulae in the variables of P, the variables q, r and the formula $\Box c$. To see that such axioms are not sufficient to characterise $\mathsf{S5}$ consider the two frames $\mathfrak{F}_1 := (\mathbb{N}, \mathbb{N}^2)$ and $\mathfrak{F}_2 := (\mathbb{N}, \leq)$. Then for every such formula A, world n of $\mathfrak{F}_2$ and valuation $\sigma : \mathbb{N} \to \mathfrak{P}(\mathsf{Var})$ with $\mathfrak{F}_2, n, \sigma \Vdash \neg A$ we construct a valuation τ on $\mathfrak{F}_1$ by setting $\tau(m) := \sigma(m + n)$. Now it is not too difficult to check that $\mathfrak{F}_1, 0, \tau \Vdash \neg A$ as well. Similarly every such formula which is satisfied in $\mathfrak{F}_1$ can also be satisfied in $\mathfrak{F}_2$, and thus the same formulae of this format are valid in the two frames. But $\mathfrak{F}_1$ is an $\mathsf{S5}$-frame, while $\mathfrak{F}_2$ is not. As the class of $\mathsf{S5}$-frames is modally definable [2] the result follows as in Theorem 31. $\Box$

7 Discussion

Our goal in this paper was to establish a taxonomy of rule formats, together with a methodology for obtaining limitative results on the existence of sequent calculi for particular modal logics. The main application was a formal proof of non-existence of cut-free sequent calculus for $\mathsf{S5}$ in the most general format considered satisfying a (reasonably mild) additional permutation property. Since as well as being instrumental for syntactic proofs this property is present in virtually all cut-free systems, this strongly suggests that a cut-free calculus for S5 will require additional machinery in the rule format or a very different, possibly semantic proof of cut admissibility. But even in the latter case, the rule set would need to violate permutability which we consider highly unlikely.

Clearly, these early results offer much potential for refinement, as e.g. rules with restrictions might be considered too restrictive. Other formats, e.g. using context relations [1], capture more logics but often permit trivial cut-free systems with all theorems of a logic as axioms so that further work is needed to map out this hitherto uncharted landscape. Presently, we are considering a more relaxed rule format where context formulae are permitted to change sides, and investigate applications to provability logic.

References

1. Avron, A., Lahav, O.: Kripke semantics for basic sequent systems. In: Brünnler, K., Metcalfe, G. (eds.) TABLEAUX 2011. LNCS (LNAI), vol. 6793, pp. 43–57. Springer, Heidelberg (2011)
2. Blackburn, P., de Rijke, M., Venema, Y.: Modal Logic. Cambridge University Press (2001)
3. Brünnler, K.: Deep sequent systems for modal logic. Arch. Math. Log. 48, 551–577 (2009)
4. Chellas, B.F.: Modal Logic. Cambridge University Press (1980)
5. Ciabattoni, A., Galatos, N., Terui, K.: Algebraic proof theory for substructural logics: Cut-elimination and completions. Ann. Pure Appl. Logic 163, 266–290 (2012)
6. Ciabattoni, A., Lahav, O., Spendier, L., Zamansky, A.: Automated support for the investigation of paraconsistent and other logics. In: Artemov, S., Nerode, A. (eds.) LFCS 2013. LNCS, vol. 7734, pp. 119–133. Springer, Heidelberg (2013)
7. Gentzen, G.: Untersuchungen über das logische Schließen. I. Math. Z. 39(2), 176–210 (1934)
8. Ghilardi, S.: Unification in intuitionistic logic. J. Symb. Log. 64(2), 859–880 (1999)
9. Goré, R.: Cut-free sequent and tableau systems for propositional diodorean modal logics. Studia Logica 53, 433–457 (1994)
10. Kracht, M.: Power and weakness of the modal display calculus. In: Wansing, H. (ed.) Proof Theory of Modal Logic, pp. 93–121. Kluwer (1996)
11. Lellmann, B., Pattinson, D.: Cut elimination for shallow modal logics. In: Brünnler, K., Metcalfe, G. (eds.) TABLEAUX 2011. LNCS (LNAI), vol. 6793, pp. 211–225. Springer, Heidelberg (2011)
12. Lellmann, B., Pattinson, D.: Constructing cut free sequent systems with context restrictions based on classical or intuitionistic logic. In: Lodaya, K. (ed.) ICLA 2013. LNCS (LNAI), vol. 7750, pp. 148–160. Springer, Heidelberg (2013)
13. Negri, S.: Proof analysis in modal logic. J. Philos. Logic 34, 507–544 (2005)
14. Negri, S., von Plato, J.: Structural proof theory. Cambridge University Press (2001)
15. Poggiolesi, F.: Gentzen Calculi for Modal Propositional Logic. Trends in Logic, vol. 32. Springer, Heidelberg (2011)
16. Troelstra, A.S., Schwichtenberg, H.: Basic Proof Theory. Cambridge Tracts in Theoretical Computer Science, 2nd edn., vol. 43. Cambridge University Press (2000)

Schemata of Formulæ in the Theory of Arrays[⋆]

Nicolas Peltier

CNRS - Grenoble Informatics Laboratory
`Nicolas.Peltier@imag.fr`

Abstract. We consider schemata of quantifier-free formulæ, defined using indexed symbols and iterated connectives ranging over intervals (such as $\bigvee_{i=1}^{n} \phi$ or $\bigwedge_{i=1}^{n} \phi$), and interpreted in the theory of arrays (with the usual functions for storing and selecting elements in an array). We first prove that the satisfiability problem is undecidable (it is clearly semi-decidable). We then consider a natural restriction on the considered structures and we prove that it makes the logic decidable by providing a sound, complete and terminating proof procedure.

1 Introduction

In [2] (see also [3]) the logic of *iterated schemata* is defined, which enriches the language of propositional logic with arithmetic parameters, indexed variables and iterated connectives ranging over intervals of natural numbers. This language allows one to formally define families of formulæ depending on arithmetic parameters, such as, e.g., the parameterized formula $(p_0 \wedge \bigwedge_{\mathtt{i}=0}^{\mathtt{n}} p_i \Rightarrow p_{i+1}) \Rightarrow p_{\mathtt{n}+1}$. Decidability and undecidability results have been obtained for the proposed language (according to the form of the arithmetic expressions that occur in the formulæ) and proof procedures have been devised to test the validity of schemata of formulæ. These procedures use the usual decomposition rules of propositional logic, together with rules performing a lazy instantiation of the parameters and loop detection techniques encoding a limited form of mathematical induction. In [5], these results have been extended to some theories beyond propositional logic, including in particular a fragment of Presburger arithmetic. In the present paper, we consider schemata of formulæ interpreted over the theory of arrays, that plays a central role in program verification[1]. The theory of arrays is defined by using two function symbols *store* and *select* encoding respectively the storage and retrieving of an element in an array, and defined by the two following axioms:

$$\forall x, z, v,\ \ \mathit{select}(\mathit{store}(x, z, v), z) \simeq v \tag{1}$$
$$\forall x, z, w, v,\ \ z \simeq w \vee \mathit{select}(\mathit{store}(x, z, v), w) \simeq \mathit{select}(x, w) \tag{2}$$

⋆ This work has been partly funded by the project ASAP of the French *Agence Nationale de la Recherche* (ANR-09-BLAN-0407-01).

1 This theory does not fall in the scope of the results in [5].

These axioms state that if an element v is inserted into an array x at some position z, then the resulting array contains v at position z (first axiom) and contains the same elements as in x elsewhere (second axiom).

This theory allows for instance to model the heap when describing the behavior of a program. It is very well-studied and several procedures have been proposed to test the validity of formulæ modulo this theory and many extensions. For instance, a decision procedure is defined in [9,8] for dealing with formulæ over the theory of arrays with uninterpreted or interpreted elements and indices, which is able to handle a restricted form of quantification over indices. The work described in [10] focuses on arrays with integer indices and devises a method to combine existing decision procedures which makes it possible to handle some important features of arrays such as sortedness or array dimension. We also mention the logic presented in [11], which is devoted to reasoning with arrays of integers (by reduction to the emptiness problem for counter automata).

In the present paper, we consider schemata of formulæ interpreted on the theory of arrays and defined using indexed variables and iterated connectives. These indexed symbols may be used, for instance, to denote the state of the same variable at different times during the running of a program. Consider for instance the following example. If, in an array T, n elements $e_1, \dots, e_n$ are inserted at *distinct* indices $i_1, \dots, i_n$, then it is clear that the result does not depend on the order in which the insertion is performed. A particular instance of this problem can be modeled by constructing the following two sequences T'_n and T_n:

$$\begin{array}{lll} T_0 \simeq T & T'_0 \simeq \mathit{store}(T, i', a) & \bigwedge_{j=0}^{n} i_j \neq i' \\ \bigwedge_{j=0}^{n} T_{j+1} \simeq \mathit{store}(T_j, i_j, e_j) & \bigwedge_{j=0}^{n} T'_{j+1} \simeq \mathit{store}(T'_j, i_j, e_j) & \end{array}$$

Intuitively, T_{n+1} is obtained from T by inserting the elements $e_1, \dots, e_n$ in the cells $i_1, \dots, i_n$ respectively, and T'_{n+1} is obtained by inserting $a, e_1, \dots, e_n$ at $i', i_1, \dots, i_n$. We want to prove that the array obtained by storing a into T_{n+1} in cell i' is identical to T'_{n+1} modulo extensionality, i.e. that $\forall n\ \forall i\ \mathit{select}(\mathit{store}(T_{n+1}, i', a), i) \simeq \mathit{select}(T'_{n+1}, i)$ is a logical consequence of the previous set of axioms. Note that the index variables j and n must be interpreted as natural numbers hence this problem cannot be encoded in first-order logic[2]. By negating the conclusion, we obtain a clause containing a constant interpreted as a natural number. Existing SMT-solvers (see, e.g., `www.SMT-LIB.org`) cannot decide the validity of such formulæ, unless of course n is instantiated to some fixed number[3]. For proving that the formula holds for all values of n, the use of mathematical induction is essential.

The paper is structured as follows. After defining in Section 2 the syntax and the semantics of the considered logic, we prove in Section 3 that the satisfiability problem is undecidable (it is obviously semi-decidable), by reduction to the

[2] It does not fall in the scope of existing procedures for handling combination of first-order logic and Presburger arithmetic (see, e.g., [7]), since it necessarily involves induction.

[3] As far as we are aware, this problem is out of the scope of the known decidable extensions of the theory of arrays.

Post correspondence problem. Remarkably, this result still holds if the syntax of the formula is further restricted by forbidding arrays containing elements of the same sort as their indices (thus discarding terms such as $\mathit{select}(T, \mathit{select}(T, i))$). In Section 4, we impose additional semantic restrictions on the considered interpretations and define a decision procedure for testing the satisfiability of formulæ in this particular class of interpretations. The proof procedure is based on propositional tableaux enriched by specific rules for equality reasoning modulo the theory of arrays, together with new simplification and loop detection rules ensuring termination. The conditions on the interpretations that make possible the definition of the decision procedure are formally defined in Section 4.2. Informally, these conditions can be summarized as follows: two indexed constants a_i and b_j, with $i < j$ encoding array indices cannot be equal, unless there exists a non-indexed constant c such that $a_i = c = b_j$ or a sequence of constants $c^1, \ldots, c^{j-1-i}$ such that $a_i = c^1_{i+1} = \ldots = c^{j-i-1}_{j-1} = b_j$ holds in the considered interpretation. This restriction does not apply in a systematic way on all the symbols occurring in the formula, but only on those encoding the indices of certain arrays, more precisely those on which a *store* operation is performed, and those containing elements of the same sort as their indices. We call the interpretations satisfying this requirement *contiguous*. Going back to the previous example, the requirement holds if we assume for instance (in addition to the previous properties) that the i_j's are pairwise distinct (more generally it is clear that the requirement always holds if constants with distinct indices are interpreted by distinct terms). Some simple examples of application of our work are presented in Section 5, in which we show that our procedure can be employed to check properties of simple programs with loops, where indexed constants a_i are used to denote the value of some variables a at iteration i. Section 6 briefly concludes the paper and provides some lines of future work. Due to space restrictions, some of the proofs are omitted.

2 Schemata of Formulæ

2.1 Syntax

The set of *sort terms* $\mathtt{T}$ is constructed inductively over a finite set of *base sorts* $\mathtt{B}$ using the operator $\rightarrow$: if $\mathtt{s}$ and $\mathtt{s}'$ are two sort terms in $\mathtt{T}$ then the sort term $\mathtt{s} \rightarrow \mathtt{s}'$ denotes the sort of the arrays (or functions) mapping elements of sort $\mathtt{s}$ to elements of sort $\mathtt{s}'$. Let $\mathtt{nat}$ be a special sort symbol, not occurring in $\mathtt{B}$ or $\mathtt{T}$. Let Σ be a set of symbols (denoting either constants of a base sort or arrays). Each symbol is Σ is associated with a unique *profile* that is either a sort term or of the form $\mathtt{nat} \rightarrow \mathtt{s}$ where $\mathtt{s}$ is a sort term. A symbol is called *indexed* iff its profile is of the later form. The set of indexed and non-indexed symbols are denoted by $\Sigma_{\mathtt{nat}}$ and $\Sigma_{\perp}$, respectively.

Let $\mathcal{V}$ be a set of *arithmetic constants*. An *arithmetic expression* of parameter $\mathtt{n} \in \mathcal{V}$ is an expression of the form k or $\mathtt{n} + k$, where $\mathtt{n} \in \mathcal{V}$ and $k \in \mathbb{N}$. As usual, $\mathtt{n} + 0$ is simply written $\mathtt{n}$. The set of arithmetic expressions of parameter $\mathtt{n}$ is denoted by $\mathcal{T}_{\mathbb{N}}(\mathtt{n})$. If $\alpha, \beta \in \mathcal{T}_{\mathbb{N}}(\mathtt{n})$ then we write $\alpha < \beta$ iff the previous relation holds regardless of the value of the parameter $\mathtt{n}$, i.e., iff one of the following conditions hold: α and β are natural numbers and $\alpha < \beta$, α and β are of the form $\mathtt{n} + k$ and $\mathtt{n} + l$ respectively and $k < l$, or α is a natural number, β is of the form $\mathtt{n} + k$ and $\alpha < k$.

The set of terms $\mathcal{T}(\mathtt{s}, \mathtt{n})$ of sort $\mathtt{s} \in \mathtt{T}$ and of parameter $\mathtt{n} \in \mathcal{V}$ is built inductively as follows.

- All non-indexed symbols $u \in \Sigma_{\perp}$ of profile $\mathtt{s}$ are terms in $\mathcal{T}(\mathtt{s}, \mathtt{n})$.
- For all $u \in \Sigma_{\mathtt{nat}}$ of profile $\mathtt{nat} \to \mathtt{s}$ and for all $\alpha \in \mathcal{T}_{\mathbb{N}}(\mathtt{n})$, we have $u_\alpha \in \mathcal{T}(\mathtt{s}, \mathtt{n})$.
- For all sort terms $\mathtt{s}'$, for all $(t, u) \in \mathcal{T}(\mathtt{s}' \to \mathtt{s}, \mathtt{n}) \times \mathcal{T}(\mathtt{s}', \mathtt{n})$, $\mathit{select}(t, u) \in \mathcal{T}(\mathtt{s}, \mathtt{n})$.
- If $\mathtt{s}$ is of the form $\mathtt{s}' \to \mathtt{s}''$, then for all $(t, u, v) \in \mathcal{T}(\mathtt{s}, \mathtt{n}) \times \mathcal{T}(\mathtt{s}', \mathtt{n}) \times \mathcal{T}(\mathtt{s}'', \mathtt{n})$, $\mathit{store}(t, u, v) \in \mathcal{T}(\mathtt{s}, \mathtt{n})$.

The set $\mathcal{T}(\mathtt{n})$ denotes the entire set of terms $\mathcal{T}(\mathtt{n}) \overset{\text{def}}{=} \bigcup_{\mathtt{s} \in \mathtt{T}} \mathcal{T}(\mathtt{s}, \mathtt{n})$. If u is a term of sort $\mathtt{s} \to \mathtt{s}'$, then $\mathtt{s}$ is the *domain* of u and $\mathtt{s}'$ is its *range*. For every arithmetic expression α, we denote by $\Sigma[\alpha]$ the set of terms that are either element of $\Sigma_{\perp}$ or of the form u_α, with $u \in \Sigma_{\mathtt{nat}}$.

The *term depth* $\delta(u)$ and the *index depth* $\iota(u)$ of a term u are inductively defined as follows: $\delta(u_\alpha) \overset{\text{def}}{=} \delta(u) \overset{\text{def}}{=} 1$ if $u \in \Sigma_{\mathtt{nat}} \cup \Sigma_{\perp}$, $\delta(\mathit{select}(t, u)) \overset{\text{def}}{=} \max(\delta(t), \delta(u)) + 1$, $\delta(\mathit{store}(t, u, v) \overset{\text{def}}{=} \max(\delta(t), \delta(u), \delta(v)) + 1$, $\iota(u) \overset{\text{def}}{=} 0$ if $u \in \Sigma_{\perp}$, $\iota(u_{\mathtt{n}+k}) \overset{\text{def}}{=} \iota(u_k) \overset{\text{def}}{=} k + 1$ if $k \in \mathbb{N}$, $\iota(\mathit{select}(t, u)) \overset{\text{def}}{=} \max(\iota(t), \delta(u))$, $\iota(\mathit{store}(t, u, v) \overset{\text{def}}{=} \max(\iota(t), \delta(u), \delta(v))$, For instance, if $s \in \Sigma_{\perp}$ then $\mathit{select}(t_{\mathtt{n}+1}, \mathit{select}(s, u_{\mathtt{n}}))$ is of term depth 3 and of index depth 2.

The set $\mathcal{F}(\mathtt{n})$ of *A-formulæ* of parameter $\mathtt{n}$ is inductively constructed as follows (note that we assume that all formulæ are in negation normal form).

- $\perp, \top \in \mathcal{F}(\mathtt{n})$.
- If $u, v \in \mathcal{T}(\mathtt{s}, \mathtt{n})$ for some sort $\mathtt{s} \in \mathtt{T}$, then $u \simeq v$ and $u \not\simeq v$ are in $\mathcal{F}(\mathtt{n})$.
- If $\phi, \psi \in \mathcal{F}(\mathtt{n})$ then $\phi \wedge \psi, \phi \vee \psi \in \mathcal{F}(\mathtt{n})$.
- If $i \in \mathcal{V}$, $\phi \in \mathcal{F}(\mathtt{i})$, and $k, l \in \mathbb{N}$ then $\bigvee_{\mathtt{i}=k}^{\mathtt{n}+l} \phi$ and $\bigwedge_{\mathtt{i}=k}^{\mathtt{n}+l} \phi$ are in $\mathcal{F}(\mathtt{n})$.

An expression α is an *index* of a formulæ ϕ if it occurs in a term of the form u_α of ϕ. The term depth and index depth of an A-formula ϕ is the maximal term (resp. index) depth of the terms occurring in ϕ. The *propositional depth* $\pi(\phi)$ is defined as follows. $\pi(\perp) \overset{\text{def}}{=} \pi(\top) \overset{\text{def}}{=} \pi(u \simeq v) \overset{\text{def}}{=} \pi(u \not\simeq v) \overset{\text{def}}{=} 1$, $\pi(\phi \star \psi) \overset{\text{def}}{=} 1 + \max(\pi(\phi), \pi(\psi))$ and $\pi(\Pi_{\mathtt{i}=k}^{\mathtt{n}+l} \phi) = 1 + \pi(\phi)$ (with $\star \in \{\vee, \wedge\}$, $\Pi \in \{\bigvee, \bigwedge\}$).

For every expression e, we denote by $e\{\alpha \mapsto \beta\}$ the expression obtained from e by replacing every occurrence of α by β.

2.2 Semantics

The semantics of A-formulæ is defined in a straightforward way. An *interpretation* $\mathcal{I}$ is a function mapping every expression e in $\mathtt{T} \cup \mathcal{V} \cup \{\mathit{store}, \mathit{select}\} \cup \bigcup_{\mathtt{n} \in \mathcal{V}} (\mathcal{T}_{\mathbb{N}}(\mathtt{n}) \cup \mathcal{T}(\mathtt{n}) \cup \mathcal{F}(\mathtt{n}))$ to an object $[e]^{\mathcal{I}}$ such that:

- For every sort $\mathtt{s} \in \mathtt{T}$, $[\mathtt{s}]^{\mathcal{I}}$ is a non-empty set of elements (the *domain* of $\mathtt{s}$).
- For every $\mathtt{n} \in \mathcal{V}$, $[\mathtt{n}]^{\mathcal{I}}$ is a natural number.
- For every symbol u of profile $\mathtt{s} \in \mathtt{T}$, $[u]^{\mathcal{I}}$ is an element of $[\mathtt{s}]^{\mathcal{I}}$.

- For every constant u of profile $\mathtt{nat} \to \mathtt{s}$, $[u]^{\mathcal{I}}$ is a function from $\mathbb{N}$ to $[\mathtt{s}]^{\mathcal{I}}$.
- $[k]^{\mathcal{I}} \stackrel{\text{def}}{=} k$ if $k \in \mathbb{N}$, and $[\mathtt{n}+k]^{\mathcal{I}} \stackrel{\text{def}}{=} [\mathtt{n}]^{\mathcal{I}} + k$ if $\mathtt{n} \in \mathcal{V}, k \in \mathbb{N}$.
- $[\mathit{store}]^{\mathcal{I}}$ is a function that maps every triple $(e, e', f) \in ([\mathtt{s} \to \mathtt{s}']^{\mathcal{I}}, [\mathtt{s}]^{\mathcal{I}}, [\mathtt{s}']^{\mathcal{I}})$ to an element of $[\mathtt{s} \to \mathtt{s}']^{\mathcal{I}}$.
- $[\mathit{select}]^{\mathcal{I}}$ is a function that maps every pair $(e, e') \in ([\mathtt{s} \to \mathtt{s}']^{\mathcal{I}}, [\mathtt{s}]^{\mathcal{I}})$ to an element of $[\mathtt{s}']^{\mathcal{I}}$.
- $\mathcal{I}$ satisfies the axioms of the theory of arrays, i.e., for every $t, u, v, w \in ([\mathtt{s} \to \mathtt{s}']^{\mathcal{I}}, [\mathtt{s}]^{\mathcal{I}}, [\mathtt{s}']^{\mathcal{I}}, [\mathtt{s}]^{\mathcal{I}})$, we have: $[\mathit{select}]^{\mathcal{I}}([\mathit{store}]^{\mathcal{I}}(t,u,v),u) = v$ and $[\mathit{select}]^{\mathcal{I}}([\mathit{store}]^{\mathcal{I}}(t,u,v),w) = [\mathit{select}]^{\mathcal{I}}(t,w)$ if $u \neq w$.
- $[u_\alpha]^{\mathcal{I}} \stackrel{\text{def}}{=} [u]^{\mathcal{I}}([\alpha]^{\mathcal{I}})$, and for every term $f(t_1,\dots,t_n)$, where $f \in \{\mathit{select}, \mathit{store}\}$, $n = 2, 3$, we have $[f(t_1,\dots,t_n)]^{\mathcal{I}} \stackrel{\text{def}}{=} [f]^{\mathcal{I}}([t_1]^{\mathcal{I}},\dots,[t_n]^{\mathcal{I}})$.
- For every A-formula ϕ, $[\phi]^{\mathcal{I}}$ is a truth value in $\{\mathtt{true}, \mathtt{false}\}$, inductively defined as follows.
- $[\top]^{\mathcal{I}} \stackrel{\text{def}}{=} \mathtt{true}$, $[\bot]^{\mathcal{I}} \stackrel{\text{def}}{=} \mathtt{false}$, $[u \simeq v]^{\mathcal{I}} \stackrel{\text{def}}{=} \mathtt{true}$ iff $[u]^{\mathcal{I}} = [v]^{\mathcal{I}}$, $[u \not\simeq v]^{\mathcal{I}} \stackrel{\text{def}}{=} \mathtt{true}$ iff $[u]^{\mathcal{I}} \neq [v]^{\mathcal{I}}$, $[\phi \vee \psi]^{\mathcal{I}} \stackrel{\text{def}}{=} \mathtt{true}$ iff $[\phi]^{\mathcal{I}} = \mathtt{true}$ or $[\psi]^{\mathcal{I}} = \mathtt{true}$, and $[\phi \wedge \psi]^{\mathcal{I}} \stackrel{\text{def}}{=} \mathtt{true}$ iff $[\phi]^{\mathcal{I}} = \mathtt{true}$ and $[\psi]^{\mathcal{I}} = \mathtt{false}$.
- $[\bigvee_{\mathtt{i}=k}^{\mathtt{n}+l} \phi]^{\mathcal{I}} \stackrel{\text{def}}{=} \mathtt{true}$ iff there exists a natural number $m \in [k, [\mathtt{n}]^{\mathcal{I}} + l]$ such that $[\phi]^{\mathcal{I}[\mathtt{i} \leftarrow m]} = \mathtt{true}$, where $\mathcal{I}[\mathtt{i} \leftarrow m]$ denotes the interpretation coinciding with $\mathcal{I}$ except on $\mathtt{i}$, for which we have $[\mathtt{i}]^{\mathcal{I}[\mathtt{i} \leftarrow m]} \stackrel{\text{def}}{=} m$.
- $[\bigwedge_{\mathtt{i}=k}^{\mathtt{n}+l} \phi]^{\mathcal{I}} \stackrel{\text{def}}{=} \mathtt{true}$ iff for all natural numbers $m \in [k, [\mathtt{n}]^{\mathcal{I}} + l]$ we have $[\phi]^{\mathcal{I}[\mathtt{i} \leftarrow m]} = \mathtt{true}$.

An interpretation $\mathcal{I}$ is a *model* of an A-formula ϕ (resp. set of A-formulæ S) if $[\phi]^{\mathcal{I}} = \mathtt{true}$ (resp. if $\forall \phi \in S, [\phi]^{\mathcal{I}} = \mathtt{true}$). This is written $\mathcal{I} \models \phi$ (resp. $\mathcal{I} \models S$). An A-formula or a set of A-formulæ having a model is *satisfiable.*

3 Undecidability Results

The following proposition is an immediate consequence of the previous definition.

Proposition 1. *The satisfiability problem is semi-decidable for A-formulæ.*

Proof. If the value of the parameter is fixed then all iterated formulæ can be replaced by standard disjunctions and conjunctions. The obtained A-formula is then equivalent to a ground formula interpreted on the theory of arrays, for which satisfiability can be tested by usual procedures. Thus it suffices to enumerate all the possible values of the parameter until we get one for which the considered formula is satisfiable.

The next theorem shows that the problem is not decidable.

Theorem 2. *The satisfiability problem is undecidable for A-formulæ, even if the sets of range and domain sorts are disjoint*[4]*.*

[4] The case in which the domain and range of the arrays are allowed to be identical follows immediately from the results in [5] about the undecidability of schemata of equational formulæ in the empty theory.

Proof. We provide a brief informal overview of the proof. It is based on an encoding of the Post correspondence problem. We thus consider two sequences of words $w^{1,1}, \dots, w^{1,n}$ and $w^{2,1}, \dots, w^{2,n}$ and we construct an A-formula ϕ that is satisfiable iff there exists a sequence $I_1, \dots, I_k$ such that $w^{1,I_1}. \cdots .w^{1,I_k} = w^{2,I_1}. \cdots .w^{2,I_k}$ (where "." denotes word concatenation). To this purpose, we use two indexed arrays W^1 and W^2 denoting the words $w^{1,I_1}. \cdots .w^{1,I_k}$ and $w^{2,I_1}. \cdots .w^{2,I_k}$ respectively. More precisely W^i is defined on the domain $\{a_1, \dots, a_{\mathtt{n}}\}$ and *select*(W^i, a_j) contains the character j of the word $w^{i,I_1}. \cdots .w^{i,I_k}$, defined as a pair $(k,l) \in [1,n] \times [1, |w^{i,k}|]$, meaning that this character is the l-th element of the word $w^{i,k}$. It is straightforward to state as an A-formula that W^i indeed represents a word of the form $w^{i,I^i_1}. \cdots .w^{i,I^i_k}$, for some sequence $I^i_1, \dots, I^i_k$: it suffices to check that the successor of any element (k,l) is either of the form $(k, l+1)$ (if l is not the last character in $w^{i,k}$) or of the form $(k',1)$, for some $k' \in [1,n]$ (if l is the last element of $w^{i,k}$). Similarly, it is easy to check that the words denoted by W^1 and W^2 are identical. The difficult point is to check that the two sequences $I^i_1, \dots, I^i_k$ $(i = 1,2)$ are identical. For expressing this property, we take advantage of the expressive power of the theory of arrays: We introduce two indexed constants b^i such that $(b^i_1, \dots, b^i_{\mathtt{n}})$ is *exactly* the sequence of cells corresponding to the beginning of a new word in W^i. To define these sequences, we construct two arrays B^i $(i = 1,2)$ containing `true` exactly at the cells corresponding to the beginning of a new word in W^i (which can be expressed by stating that for every $j = 1, \dots, \mathtt{n}$, *select*(B^i, aj) is `true` iff *select*(W^i, a_j) is of the form $(k,1)$). Then we check that this array B^i is identical to the array obtained by inserting `true` at every cell b^i_j $(1 \leq j \leq \mathtt{n})$, inside an array containing initially `false` at each cell a_j $(1 \leq j \leq \mathtt{n})$. This guarantees that $\{b^i_1, \dots, b^i_{\mathtt{n}}\}$ is indeed the set of cells corresponding to the start of a new word. The most subtle point is to ensure that the order of these cells is the intended one, i.e. that the first new word after b_j indeed starts at b_{j+1} and not, say, at b_{j+2}. This is done by adding axioms "copying" the value of the next element corresponding to the start of a word all along the array W^i in order to check that two contiguous words really start at some cells b^i_j and b^i_{j+1}. Using these sequences b^i_j, it is straightforward to check that the sequences I^i_j are identical, by verifying that *select*(W^1, b_j) is equal to *select*(W^2, b_j), for every $j = 1, \dots, \mathtt{n}$.

4 Decidability Results

The results in Section 3 show that the satisfiability problem is undecidable for A-formulæ. In such a situation, the standard approach consists in focusing on particular syntactic fragments so that decidability and/or complexity results can be obtained. In the present paper, instead of restricting the class of formulæ, we prefer to restrict the class of *interpretations* by imposing additional conditions on the considered structures. These conditions are formalized in Section 4.2.

4.1 Simplifying the Syntax

For technical convenience, we shall assume from now that all the formulæ satisfy some additional syntactic restrictions:

1. The term depth is bounded by 2; and every literal is either an equation or a disequation between constants or indexed constants or of the form $f(\boldsymbol{u}) \simeq v$, with $f \in \{\mathit{select}, \mathit{store}\}$ and $\boldsymbol{u}, v$ are constants or indexed constants.
2. The only arithmetic expressions occurring in the considered formula are 0, $\mathtt{n}$ or $\mathtt{n}+1$, where $\mathtt{n} \in \mathcal{V}$.
3. The formula contains no nested iterations, i.e., for every subformula of the form $\bigvee_{\mathtt{i}=0}^{\mathtt{n}+l} \phi$ or $\bigwedge_{\mathtt{i}=k}^{\mathtt{n}+l} \phi$, ϕ contains no iterated formula.

It is easy to check that these conditions do not reduce the expressive power of the language. First, any arithmetic expression distinct from the parameter $\mathtt{n}$ occurring as the upper bound of an iteration can be removed by unfolding the considered iteration, e.g., $\bigvee_{\mathtt{i}=0}^{\mathtt{n}+2} \phi$ is equivalent to $\bigvee_{\mathtt{i}=0}^{\mathtt{n}} \phi \vee \phi\{\mathtt{i} \mapsto \mathtt{n}+1\} \vee \phi\{\mathtt{i} \mapsto \mathtt{n}+2\}$. It is also easy to get rid of an iteration whose lower bound is a number $k \neq 0$: this can be done by introducing a new atom $p_{\mathtt{i}}^k$ stating that $\mathtt{i}$ is strictly greater than k, and defined by the axioms: $\neg p_0^k \wedge \ldots \wedge \neg p_{k-1}^k \wedge p_k^k \wedge \bigwedge_{\mathtt{i}=0}^{\mathtt{n}} \neg p_{\mathtt{i}}^k \vee p_{\mathtt{i}+1}^k$. Then an iteration $\bigvee_{\mathtt{i}=k}^{\mathtt{n}} \phi$ (resp. $\bigwedge_{\mathtt{i}=k}^{\mathtt{n}} \phi$) can be written $\bigvee_{\mathtt{i}=0}^{\mathtt{n}} p_{\mathtt{i}}^k \wedge \phi$ (resp. $\bigwedge_{\mathtt{i}=0}^{\mathtt{n}} \neg p_{\mathtt{i}}^k \vee \phi$). Condition 3 is also easy to enforce, because any iteration $\Pi_{\mathtt{j}=0}^{\mathtt{i}} \psi$ occurring inside an iteration can be replaced by a new atom $p_{\mathtt{i}}$, while adding the axioms: $\neg p_0 \vee \psi\{\mathtt{i} \mapsto 0\}$ and $\bigwedge_{\mathtt{i}=0}^{\mathtt{n}} \neg p_{\mathtt{i}+1} \vee (\psi\{\mathtt{j} \mapsto \mathtt{i}+1\} \star p_{\mathtt{i}})$ (with $(\Pi, \star) \in \{(\bigvee, \vee), (\bigwedge, \wedge)\}$). Afterward, Condition 1 can be enforced by applying the standard *flattening* operation (see for instance [6]), i.e., every complex subterm u of parameter $\mathtt{n}$ can be (repeatedly) replaced by a new constant $v_{\mathtt{n}}$, while adding the axiom $\bigwedge_{\mathtt{i}=0}^{\mathtt{n}+1} v_{\mathtt{i}} \simeq u\{\mathtt{n} \mapsto \mathtt{i}\}$. This ensures that the term depth is at most 2 and that the symbols *select* and *store* only occur in positive literals (and occur only once in every literal). Condition 2 is ensured in a similar way, by (recursively) replacing any constant $u_{\mathtt{n}+k}$ with $k > 1$ by a constant $u'_{\mathtt{n}+k-1}$, while adding the axiom $\bigwedge_{\mathtt{i}=0}^{\mathtt{n}+1} u'_{\mathtt{i}} \simeq u_{\mathtt{i}+1}$. Due to space restrictions, the formal description of these transformations is omitted, see [3,4] for more details.

4.2 Restricting the Class of Interpretations

We first define a property of the sort terms which depends only on the syntactic form of the formula. Intuitively, a sort $\mathtt{s}$ is called *non-cyclic* if: (i) no storing operation is performed on arrays of domain $\mathtt{s}$; and (ii) no array of sort $\mathtt{s} \to \mathtt{s}$ occurs in the signature, even with array composition (by composing two arrays of sort $\mathtt{s} \to \mathtt{s}'$ and $\mathtt{s}' \to \mathtt{s}$ one gets an array of sort $\mathtt{s} \to \mathtt{s}$). The condition is formalized as follows.

Definition 3. *Let $\prec$ be a (fixed) ordering among sort terms, such that $\mathtt{s}, \mathtt{s}' \prec \mathtt{s} \to \mathtt{s}'$, for all sort terms $\mathtt{s}, \mathtt{s}'$. A sort $\mathtt{s}$ is* non-cyclic *in an A-formula ϕ if the two following conditions hold.*

- *ϕ contains no term of the form store(u, v, w) where v is of sort* $\mathtt{s}$.
- *ϕ contains no term of sort* $\mathtt{s}' \to \mathtt{s}$ *with* $\mathtt{s}' \not\succ \mathtt{s}$.

The second condition of Definition 3 is related to the notion of a *stratified signature* in [1]: if the formula at hand contains no occurrence of *store* and if the signature is stratified then every sort is non-cyclic.

The conditions of Definition 3 are rather restrictive, thus instead of assuming that every sort is non-cyclic, we prefer to allow cyclic sorts and to add further restrictions on their interpretations. These restrictions are formalized in the following definition.

Definition 4. *A sort* $\mathtt{s}$ *is* contiguous *in an interpretation $\mathcal{I}$ iff for every pair of constant symbols u, v of profile* $\mathtt{nat} \to \mathtt{s}$ *and for every $k, l \in \mathbb{N}$ such that $k < l$ and $\mathcal{I} \models u_k \simeq v_l$, one of the following conditions holds:*
- *There exists a constant w of sort* $\mathtt{s}$ *such that $\mathcal{I} \models u_k \simeq w \wedge v_l \simeq w$.*
- *There exists $l - k + 1$ constants $w^1, \ldots, w^{l-k+1}$ such that $w^1 = u$, $w^{l-k+1} = v$ and $\forall i \in [1, l - k], \mathcal{I} \models w^i_{k+i-1} \simeq w^{i+1}_{k+i}$.*

For instance the condition of Definition 4 holds if $\mathtt{s}$ is finite, because in this case one may associate a non-indexed constant symbol w with each element of the domain of $\mathtt{s}$; or if the implication $k+1 < l \Rightarrow u_k \not\simeq v_l$ holds in $\mathcal{I}$, for all constant symbols $u, v \in \Sigma_{\mathtt{nat}}$ and for all $k, l \in \mathbb{N}$. It also holds if the interpretation of the constants of sort $\mathtt{nat} \to \mathtt{s}$ is monotonic, i.e., if there exists an ordering $\leq$ such that $k < l \Rightarrow a_k \leq b_l$ holds (it suffices to add for each sort $\mathtt{s}$ a new constant u such that u_k is interpreted as the maximal term of sort $\mathtt{s}$ of the form v_{k-1}).

Definition 5. *An interpretation $\mathcal{I}$ is* contiguous *if every sort is contiguous in $\mathcal{I}$. It is* quasi-contiguous *iff every cyclic sort is contiguous.*

The following lemma shows that quasi-contiguous and contiguous interpretations are equivalent for satisfiability testing:

Lemma 6. *An A-formula has a quasi-contiguous model iff it has a contiguous model (up to the addition of a finite set of new constant symbols and axioms).*

In the next section, we shall therefore assume that all the considered interpretations are contiguous.

4.3 Proof Procedure

We devise a tableau-based proof procedure deciding the satisfiability of A-formulæ in contiguous (or quasi-contiguous) interpretations. We first briefly review some basic terminology. Tableaux are viewed as trees labeled by sets of A-formulæ. A *branch* is a path from the root to a leaf. An interpretation $\mathcal{I}$ *validates* a tableau if there exists a leaf labeled by some set S such that $\mathcal{I} \models S$. A branch is *closed* if it contains $\bot$. The procedure is defined by rules of the form $\dfrac{H_1, \ldots, H_n}{\mathcal{C}_1 \mid \ldots \mid \mathcal{C}_m}$, meaning that a leaf labeled by a set S can be expanded by m new children, labeled by $S \cup \mathcal{C}_1, \ldots, S \cup \mathcal{C}_m$ respectively, if S contains $H_1, \ldots, H_n$ (up to an instantiation of the meta-variables). The rule only applies if the branch is open and if there is no $i \in [1, m]$ such that S contains all the formulæ in $\mathcal{C}_i$.

Overview of the Proof Procedure. The proof procedure can be informally described as follows.

- First, the usual decomposition rules of propositional logic are applied, together with additional transitivity and paramodulation rules handling the properties of the equality predicate. We also consider generalized decomposition rules unfolding iterated formulæ (e.g., to infer $\phi\{\mathtt{i} \mapsto \mathtt{n}+1\}$ from $\bigwedge_{\mathtt{i}=0}^{\mathtt{n}+1} \phi$). In order to handle store operations, we introduce new atoms of the form $t \simeq_E s$, meaning that t and s coincide on every element, except on those occurring in the set E.
- Then, we apply an inductive rule performing a case analysis on the parameter $\mathtt{n}$, considering separately the two cases $\mathtt{n} = 0$ and $\mathtt{n} > 0$. The rule recursively replaces $\mathtt{n}$ by either 0 or $\mathtt{n}+1$, and thus lazily instantiates the parameter with natural numbers, which enables further applications of the decomposition rules. Of course, the addition of the induction rule makes the calculus non-terminating, since $\mathtt{n}$ can be instantiated indefinitely.
- To avoid non-termination, a loop detection mechanism is added to prune infinite branches. To get rid of such branches, we have to ensure that the depth of the formulæ occurring in the tableau is bounded, so that every infinite branch contains a cycle, i.e., two nodes labeled by the same set of formulæ. It is easy to see that the term depth and propositional depth of the formula cannot increase, thus we only have to ensure that its index depth is bounded. To this aim, we introduce new (satisfiability-preserving) rules allowing to get rid of all formulæ containing an expression $\mathtt{n}+2$. This ensures that only terms indexed by 0, $\mathtt{n}$ or $\mathtt{n}+1$ will remain in the formula before the inductive rule is applied, hence the index depth will never be greater than 3. When trying to eliminate formulæ containing an occurrence of $\mathtt{n}+2$, it turns out that it is sometimes necessary to infer additional properties of the remaining symbols. For instance, if the considered branch contains a formula $\mathit{select}(t, u_{\mathtt{n}+2}) \not\simeq \mathit{select}(s, u_{\mathtt{n}+2})$ then we have to express the fact that t and s disagree at some element $u_{\mathtt{n}+2}$. But we have to express this property without explicitly referring to the term $u_{\mathtt{n}+2}$ (since our goal is to get rid of all such terms), and of course without introducing new symbols (which would prevent termination). This cannot be done in the initial language, thus we need to enrich the syntax by new predicates allowing to express such properties in a convenient way. Of course, we also need to add expansion rules encoding the axioms defining these predicates.

Enriching the Syntax. We therefore enrich the syntax of the language by new constructions. We first consider *set expressions*, denoting sets of individuals, and built using the constructor $\cup$ over a set of basic sets $\emptyset$, $\{u\}$ and $\theta(\alpha)$, where u is a constant or indexed constant and α an arithmetic expression. The sets $\emptyset$ and $\{u\}$ and the constructor $\cup$ are interpreted in a natural way, and $\theta(\alpha)$ is interpreted in any interpretation $\mathcal{I}$ as the set of terms that are distinct from all expressions in $\mathcal{T}^{\mathcal{I}}_{|<\alpha}$, with $\mathcal{T}^{\mathcal{I}}_{|<\alpha} \stackrel{\text{def}}{=} [\Sigma_{\perp}]^{\mathcal{I}} \cup \{[u_k]^{\mathcal{I}} \mid u \in \Sigma_{\mathtt{nat}}, k < [\alpha]^{\mathcal{I}}\}$. Intuitively, $\mathcal{T}^{\mathcal{I}}_{|<\alpha}$ denotes the set of named elements whose index is strictly lower than α. We also consider a predicate symbol $\in$ interpreted as usual as set membership, and a symbol $\simeq_E$ stating that two arrays agree on all elements not occurring in the set denoted by E. For instance, $t \simeq_{\{u,v\}} s$ states that t and s coincide at every element distinct from u and v and $t \simeq_{\theta(\alpha)} s$ states that t and s coincide on every element in $\mathcal{T}^{\mathcal{I}}_{|<\alpha}$. Furthermore, we introduce a predicate $\not\sim_\alpha$ (with $\alpha \in \mathcal{T}_{\mathbb{N}}(\mathtt{n}) \cup \{-1\}$), such that $\mathcal{I} \models t \not\sim_\alpha s$ iff there exists an element $e \in \theta(\alpha+1)$ (in the domain of t, s) such that the interpretations of t and s disagree at e. The advantage of the predicates $\simeq_E$ and $\not\sim_\alpha$ is that they allow us to express properties of arrays without having to refer explicitly to the symbols denoting elements of the domain of these arrays. In particular, the symbol $\simeq_E$ can be used to encode store operations. More precisely, every atom of the form $\mathit{store}(t,u,v) \simeq s$ can be replaced by the conjunction $\mathit{select}(s,u) \simeq v \wedge t \simeq_{\{u\}} s$, stating the fact that s contains v at u and coincides with t at all other elements. Thanks to this transformation (which obviously preserves satisfiability) we can assume that the considered A-formula contains no instance of *store*. Finally, we add new constants denoting all terms $\mathit{select}(t,u) \in \mathcal{T}(\mathtt{n})$, with appropriate axioms, so that the index of the constant denoting a term $\mathit{select}(t,u)$ is the maximal index in t, u (note that these special constants do not themselves occur in $\mathcal{T}(\mathtt{n})$).

We are now in position to formally describe the inference rules defining the proof procedure.

Propositional Decomposition Rules. The rules ($\wedge$-D), ($\vee$-D) and ($\bot$) are the usual decomposition rules of propositional tableaux. Other decomposition rules (similar to those in [2]) are added to handle iterated connectives.

$$(\wedge\text{-D})\ \frac{\phi \wedge \psi}{\phi, \psi} \qquad (\vee\text{-D})\ \frac{\phi \vee \psi}{\phi \quad | \quad \psi} \qquad (\bot)\ \frac{\phi, \neg\phi}{\bot}$$

$$(\bigwedge\text{-D})\ \frac{\bigvee_{\mathtt{i}=0}^{\mathtt{n}+1} \phi}{\bigvee_{\mathtt{i}=0}^{\mathtt{n}} \phi \quad | \quad \phi\{\mathtt{i} \mapsto \mathtt{n}+1\}} \qquad (\bigvee\text{-D})\ \frac{\bigwedge_{\mathtt{i}=0}^{\mathtt{n}+1} \phi}{\phi\{\mathtt{i} \mapsto \mathtt{n}+1\}, \bigwedge_{\mathtt{i}=0}^{\mathtt{n}} \phi}$$

Equality Rules. The next rules encode the usual properties of the equality predicate. The rule (S) encodes the substitutivity property, ($\simeq_E$-R$_1$) and ($\simeq_E$-R$_2$) encode the properties of the predicate $\simeq_E$, whereas (P) allows one to replace a term by an equal constant or indexed constant. Finally, (Ref) is a closure rule encoding reflexivity and (C) encodes the fact that interpretations are contiguous. To this purpose, it suffices to check that all constants of the form $u_{\beta+2}$ occur either in $\Sigma[\beta+1]$ or in $\theta(\beta+2)$.

$$(\mathrm{S})\ \frac{\mathit{select}(t,u) \simeq w, \mathit{select}(s,v) \simeq w'}{t \not\simeq s \quad | \quad u \not\simeq v \quad | \quad w \simeq w'} \qquad (\mathrm{P})\ \frac{u \simeq u', \phi[u]}{\phi[u']} \quad \text{if } \delta(u') = 1$$

and either $u' \in T(\mathtt{n})$ or u does not occur in the scope of *select* or $\simeq_E$ in ϕ.

$$(\simeq_E\text{-R}_2)\ \frac{u \simeq_E v, u' \simeq_{E'} w}{u \not\simeq u' \quad | \quad v \simeq_{E \cup E'} w} \qquad (\simeq_E\text{-R}_1)\ \frac{t \simeq_E s, \mathit{select}(t',u) \simeq v}{u \in E \quad | \quad t \not\simeq t' \quad | \quad \mathit{select}(s,u) \simeq v}$$

$$(\mathrm{Ref})\ \frac{u \not\simeq u}{\bot} \qquad (\mathrm{C})\ \frac{}{u_{\beta+2} \in E \quad | \quad u_{\beta+2} \in \theta(\beta+2)}$$

where E is the set of terms in $\Sigma[\beta+1]$ that are of the same sort as $u_{\beta+2}$.

$\sim$-Rules. The two following rules handle the predicate $\not\sim_\alpha$. The first one encodes a form of transitivity on $\not\sim_\alpha$ and $\simeq_E$. Indeed, if $u \not\sim_\alpha v$ and $w \simeq_E u$ hold, for some set of elements E whose indices are lower or equal to α, then we necessarily have $w \not\sim_\alpha v$. Indeed, by definition, $u \not\sim_\alpha v$ holds if $\mathit{select}(u, e) \neq \mathit{select}(v, e)$ for some element e distinct from every non-indexed term and from every term indexed by a natural number that is lower or equal to α. But such an element e cannot occur in E, thus w coincides with u at e, and $w \not\sim_\alpha v$ necessarily holds.

$$(\sim\text{-E}) \quad \frac{u \not\sim_\alpha v, w \simeq_E u'}{u \not\simeq u' \quad | \quad w \not\sim_\alpha v} \quad \text{if } \alpha \text{ is maximal in the node, } E \text{ is a finite set.}$$

The next rule allows one to derive expressions of the form $t \not\sim_{\alpha-1} s$. Such an expression is derivable if there exists an element e such that $\mathit{select}(t, e) \neq \mathit{select}(s, e)$ and if this element is distinct from every non-indexed term and from every term indexed by a natural number that is strictly lower than α. Thanks to the fact that the interpretations are assumed to be contiguous, we only have to test that e is distinct from all constants of the same sort of e that are non-indexed or indexed by $\alpha - 1$.

$$(\sim\text{-I}) \quad \frac{\mathit{select}(t, u_\alpha) \simeq v, \mathit{select}(s, w) \simeq w'}{u_\alpha \in E \quad | \quad u_\alpha \not\simeq w \quad | \quad v \simeq w' \quad | \quad t \not\sim_{\alpha-1} s, u_\alpha \simeq w, v \not\simeq w'}$$

where E is the set of terms in $\Sigma[\alpha - 1]$ that are of the same sort as u_α.

The next rule in this section allows one to introduce expressions of the form $\theta(\alpha)$. This is done by replacing a constant u occurring in a set E by $\theta(\alpha)$, if u occurs in $\theta(\alpha)$. This last condition is tested in the same way as in the previous rule, by verifying that u_α is distinct from all constants that are non-indexed or indexed by $\alpha - 1$.

$$\theta\text{-}I \quad \frac{t \simeq_{E \cup \{u_\alpha\}} s}{u_\alpha \in E' \quad | \quad t \simeq_{E \cup \theta(\alpha)} s}$$

where E' is the set of terms in $\Sigma[\alpha - 1]$ that are of the same sort as u_α.

The last rules in this subsection express straightforward properties of $\not\sim_\alpha$:

$$(\not\sim\text{-I}) \quad \frac{t \not\sim_{\alpha+1} s}{t \not\sim_\alpha s} \qquad (\sim\text{-}\bot) \quad \frac{t \not\sim_\alpha t}{\bot}$$

$$(\sim\text{-D}) \quad \frac{}{t \not\sim_\beta s \quad | \quad t \sim_\beta s} \qquad (\sim\text{-}T) \quad \frac{t \not\sim_\beta s}{s' \not\sim_\beta t \quad | \quad s' \not\sim_\beta s}$$

if β is the maximal index in the node, t, s, s' are non-indexed constants or constants indexed by an expression lower or equal to β.

$\in$-Rules. The following rules encode the definition of $\in$ and of the sets $\theta(\alpha)$.

$$\in\text{-}E_1 \quad \frac{u \in E \cup \{v\}}{u \in E \quad | \quad u \simeq v} \qquad \in\text{-}E_2 \quad \frac{u \in \emptyset}{\bot}$$

$$\in\text{-}E_3 \quad \frac{u \in \theta(0)}{\bigwedge_{v \in E'} u \not\simeq v} \qquad \in\text{-}E_4 \quad \frac{u \in \theta(\beta + 1)}{\bigwedge_{v \in E''} u \not\simeq v, u \in \theta(\beta)}$$

if E' and E'' denote respectively the terms in $\Sigma_\bot$ and $\Sigma[\beta]$ that are of the same sort as u.

Induction Rule. The following rule instantiates the parameter $\mathtt{n}$ by considering the two cases: $\mathtt{n} = 0$ and $\mathtt{n} > 0$. The later case is handled by replacing $\mathtt{n}$ by $\mathtt{n}+1$.

$$\text{(Ind)} \quad \frac{\Phi}{\Phi[0/\mathtt{n}] \mid \Phi\{\mathtt{n} \mapsto \mathtt{n}+1\}}$$

where $\Phi[0/\mathtt{n}]$ denotes the set of A-formulæ obtained from Φ by replacing $\mathtt{n}$ by 0, and by replacing all iterations $\Pi_{\mathtt{i}=0}^{0}\phi$ (with $\Pi \in \{\bigwedge, \bigvee\}$) by $\phi\{\mathtt{i} \mapsto 0\}$.

Loop Detection. The two following rules aim at preventing divergence, as explained in Section 4.3. The first one simply deletes from a given node all the A-formulæ containing a maximal index α (the rule applies with $\alpha = \mathtt{n}+2$, but also with $\alpha = 0, 1$ after $\mathtt{n}$ has been instantiated). Of course, the rule does not preserve satisfiability in general, but it preserves satisfiability if the considered node is irreducible by all the previous rules, except (Ind). The intuitive justification is that these rules extract all the relevant information from the formulæ and express it using only symbols indexed by expressions that are strictly smaller than α.

For every set of A-formulæ and for every arithmetic expression α, we denote by $\langle S \rangle_\alpha$ the set of A-formulæ obtained from S by removing all formulæ containing an index greater or equal to α.

$$\text{(W)} \quad \frac{S}{\langle S \rangle_\alpha}$$

if α is the maximal index expression in S and either $\alpha = \mathtt{n}+2$ or $\alpha \in \mathbb{N}$ and S contains no occurrence of $\mathtt{n}$.

Note that in contrast to the other rules, S and $\langle S \rangle_\alpha$ denote the *labels* of the parent and child nodes (not subsets of these labels). Finally, the rule (L) closes a node that is subsumed by another node in the proof tree:

$$\text{(L)} \quad \frac{S}{\bot}$$

if there exists a node N' in the tableau, distinct from the current one labeled by a set of formulæ S' such that $S' \subseteq S$ and either N' is a leaf or N' occurs in the same branch as the current node.

The Properties of the Calculus. We denote by StabArray the procedure defined by the previous inference rules. We assume that the rules are applied with the following strategy. The rules (L), (W) and (Ind) are applied with a strictly lower priority than the other rules. The rule (L) is applied only if (W) does not apply, and (Ind) applies when no other rule applies.

Theorem 7. *The calculus* StabArray *is:*

- *terminating: every tableau is finite;*
- *complete (w.r.t. the class of contiguous interpretations): every irreducible open node has a contiguous model;*
- *sound (w.r.t. the class of contiguous interpretations): an A-formula admitting a closed tableau has no contiguous model.*

The number of A-formulæ occurring in the tableau is at most exponential[5] w.r.t. the size of the initial formula. Therefore, it is easy to check that the complexity of StabArray is at most doubly exponential (since the number of nodes in the tableau is bounded by the number of sets of formulæ).

Example 8. We consider the following set of A-formulæ: $\{select(t_{\mathtt{n}+1}, u_{\mathtt{n}+1}) \simeq v, select(s, u_{\mathtt{n}+1}) \simeq w, v \not\simeq w, u_{\mathtt{n}+1} \not\simeq u_{\mathtt{n}}, t_0 \simeq s, \bigwedge_{\mathtt{i}=0}^{\mathtt{n}} t_{\mathtt{i}} \simeq_{\{u_{\mathtt{i}}\}} t_{\mathtt{i}+1}\}$. Note that this set has no contiguous model: since $u_{\mathtt{n}+1} \not\simeq u_{\mathtt{n}}$, $u_{\mathtt{n}+1}$ must be distinct from $u_{\mathtt{n}}, u_{\mathtt{n}-1}, \dots, u_0$, and thus $t_{\mathtt{n}+1}$ necessarily coincides with t_0 (hence with s) on $u_{\mathtt{n}+1}$. We first apply the rule (S) on the first two formulæ, yielding the branches $t_{\mathtt{n}+1} \not\simeq s$, $u_{\mathtt{n}+1} \not\simeq u_{\mathtt{n}+1}$ and $v \simeq w$. The last two branches can be closed immediately by applying the rules (Ref) and ($\bot$), respectively. Then the rule ($\sim$-I) applies, yielding the two branches: $u_{\mathtt{n}+1} \in E$ (where E is the set of terms in $\Sigma[\mathtt{n}]$ of the same sort as $u_{\mathtt{n}+1}$) and $t_{\mathtt{n}+1} \not\sim_{\mathtt{n}} s$ (the other branches can be closed immediately). We have $E = \{u_{\mathtt{n}}\}$, thus the rule $\in$-E_1 applies on the first branch, yielding $u_{\mathtt{n}+1} \simeq u_{\mathtt{n}}$, and the branch can be closed due to the formula $u_{\mathtt{n}+1} \not\simeq u_{\mathtt{n}}$. In the second branch, we can apply the rule (Ind). In the base case, we get $t_1 \not\sim_0 s$ and $t_0 \simeq_{\{u_0\}} t_1$, hence by (P), $s \simeq_{\{u_0\}} t_1$. Then the rule ($\sim$-E) derives $s \not\sim_0 s$ and the branch is closed by ($\sim$-$\bot$). In the inductive case, $\mathtt{n}$ is replaced by $\mathtt{n}+1$, and the formula $t_{\mathtt{n}+1} \simeq_{\{u_{\mathtt{n}+1}\}} t_{\mathtt{n}+2}$ is derived by ($\bigvee$-D). By ($\simeq_E$-R$_1$), we derive $select(t_{\mathtt{n}+1}, u_{\mathtt{n}+2}) \simeq v$ and thus $t_{\mathtt{n}+1} \not\sim_{\mathtt{n}+1} s$ can be obtained as in the previous node. Then the rule (W) applies and removes all formulæ containing $\mathtt{n}+2$. We get the set of formulæ: $\{t_{\mathtt{n}+1} \not\sim_{\mathtt{n}+1} s, v \not\simeq w, t_0 \simeq s, \bigwedge_{\mathtt{i}=0}^{\mathtt{n}} t_{\mathtt{i}} \simeq_{\{u_{\mathtt{i}}\}} t_{\mathtt{i}+1}\}(\star)$. We apply again the rule (Ind). The base case can be closed immediately as before. In the inductive case, we derive the formulæ $t_{\mathtt{n}+2} \not\sim_{\mathtt{n}+2} s$, $t_{\mathtt{n}+1} \simeq_{\{u_{\mathtt{n}+1}\}} t_{\mathtt{n}+2}$, hence by ($\sim$-E) and ($\not\sim$-I) we get $t_{\mathtt{n}+1} \not\sim_{\mathtt{n}+2} s$ and $t_{\mathtt{n}+1} \not\sim_{\mathtt{n}+1} s$. Then the rule (W) applies again, yielding a set of formulæ that is identical to ($\star$). Thus the rule (L) applies, closing the whole tableau (note that some irrelevant rule applications have been omitted for readability).

5 Applications

We provide some simple applications of the proposed procedure. The program in pseudo-code below copies in an array B all the elements occurring in an array A and satisfying some property p.

```
i ← 0
j ← 0
while i ≤ n do
    if $p(A[\mathtt{i}])$ then
        $B[\mathtt{j}] \leftarrow A[\mathtt{i}]$
        j ← j + 1
    end if
    i ← i + 1
end while
```

The behavior of this program can be encoded by the following A-formula. The parameter $\mathtt{n}$ denotes the number of iterations and $\mathtt{i}$ is the iteration rank.

[5] The exponential blow-up stems from the atoms of the form $\simeq_E$, where E is a set of terms of arbitrary cardinality.

The indexed constants $B_{\mathtt{i}}$ and $\mathtt{j}_{\mathtt{i}}$ denote the value of B and $\mathtt{j}$ at time $\mathtt{i}$. Note that since A is not affected and since it is always indexed by $\mathtt{i}$, it is simpler to encode it as an indexed constant rather than as an array.

$$\begin{gathered}\mathtt{j}_0 \simeq 0 \\ \bigwedge_{\mathtt{i}=0}^{\mathtt{n}}(\neg p(A_{\mathtt{i}}) \simeq \mathtt{true} \vee (B_{\mathtt{i}+1} \simeq \mathit{store}(B_{\mathtt{i}}, \mathtt{j}_{\mathtt{i}}, A_{\mathtt{i}}) \wedge \mathtt{j}_{\mathtt{i}+1} \simeq \mathit{select}(\mathit{succ}, \mathtt{j}_{\mathtt{i}}))) \\ \bigwedge_{\mathtt{i}=0}^{\mathtt{n}}(p(A_{\mathtt{i}}) \simeq \mathtt{true} \vee (B_{\mathtt{i}+1} \simeq B_{\mathtt{i}} \wedge \mathtt{j}_{\mathtt{i}+1} \simeq \mathtt{j}_{\mathtt{i}}))\end{gathered}$$

The symbol *succ* denotes the successor function ($\mathit{succ}(x)$ is written $\mathit{select}(\mathit{succ}, x)$ because all functions are encoded as arrays in our framework). By using a straightforward typing algorithm, we can infer that A, B and $\mathtt{j}$ are of sorts $\mathtt{nat} \to \mathtt{s}$, $\mathtt{nat} \to \mathtt{s}' \to \mathtt{s}$ and $\mathtt{nat} \to \mathtt{s}'$, respectively. The constants $\mathtt{true}$, 0, *succ* and p are of sort $\mathtt{bool}$, $\mathtt{s}'$, $\mathtt{s}' \to \mathtt{s}'$ and $\mathtt{s} \to \mathtt{bool}$, respectively. It is clear that the sort $\mathtt{s}$ is non-cyclic. The sort $\mathtt{s}'$ is cyclic, however, it is easy to check that its interpretation is necessarily contiguous, since $\mathtt{j}$ is the only constant of sort $\mathtt{nat} \to \mathtt{s}'$ and since $\mathtt{j}_{\mathtt{i}+1}$ is always obtained from $\mathtt{j}_{\mathtt{i}}$ by applying some increasing function (thus j_l is equal to j_k for $l > k$ only if $j_l = j_{l-1} = \ldots = j_k$).

We can therefore use StabArray to check that all the values stored in the final array (at $\mathtt{j}_1, \ldots, \mathtt{j}_n$) satisfies the property p. It suffices to check that the following A-formula, together with the previous axioms, is unsatisfiable: $C \simeq B_{\mathtt{n}+1} \wedge \bigvee_{\mathtt{i}=0}^{\mathtt{n}} p(\mathit{select}(C, \mathtt{j}_{\mathtt{i}})) \not\simeq \mathtt{true}$. Note that we *cannot* state in our language the fact that the initial array satisfies the property p at all cells, since the logic does not allow universal quantification.

Conversely, we can also check that all elements occurring in the initial array and satisfying the property p occur in the final array: $C \simeq B_{\mathtt{n}+1} \wedge \bigvee_{\mathtt{i}=0}^{\mathtt{n}}(u \simeq A_{\mathtt{i}} \wedge p(u) \simeq \mathtt{true}) \wedge \bigwedge_{\mathtt{i}=0}^{\mathtt{n}} \mathit{select}(C, \mathtt{j}_{\mathtt{i}}) \not\simeq u$

We provide another similar example. The following program interleaves in the same array C the elements occurring in two arrays A and B.

```
$\mathtt{j} \leftarrow 0$
$\mathtt{i} \leftarrow 0$
while $\mathtt{i} \leq \mathtt{n}$ do
    $C[\mathtt{j}] \leftarrow A[\mathtt{i}]$
    $C[\mathtt{j}+1] \leftarrow B[\mathtt{i}]$
    $\mathtt{j} \leftarrow \mathtt{j}+2$
end while
```

The behavior of the program is modeled as follows.

$$\begin{gathered}\bigwedge_{\mathtt{i}=0}^{\mathtt{n}}(C_{\mathtt{i}+1} \simeq \mathit{store}(\mathit{store}(C_{\mathtt{i}}, \mathtt{j}_{\mathtt{i}}, A_{\mathtt{i}}), \mathit{select}(\mathit{succ}, \mathtt{j}_{\mathtt{i}}), B_{\mathtt{i}})) \\ \mathtt{j}_0 \simeq 0 \wedge \bigwedge_{\mathtt{i}=0}^{\mathtt{n}} \mathtt{j}_{\mathtt{i}+1} \simeq \mathit{select}(\mathit{succ}, \mathit{select}(\mathit{succ}, \mathtt{j}_{\mathtt{i}}))\end{gathered}$$

We can check, for instance, that if the array A satisfies some property p, then all pairs of consecutive cells in the final array necessarily contain an element satisfying p:

$$\begin{gathered}D \simeq C_{\mathtt{n}+1} \wedge \bigvee_{\mathtt{i}=0}^{\mathtt{n}}(u \simeq \mathtt{j}_{\mathtt{i}} \vee u \simeq \mathit{select}(\mathit{succ}, \mathtt{j}_{\mathtt{i}})) \wedge \\ p(\mathit{select}(D, u)) \not\simeq \mathtt{true} \wedge p(\mathit{select}(D, \mathit{select}(\mathit{succ}, u))) \not\simeq \mathtt{true} \\ \bigwedge_{\mathtt{i}=0}^{\mathtt{n}} p(A_{\mathtt{i}}) \simeq \mathtt{true}\end{gathered}$$

6 Conclusion

We have shown that the satisfiability problem is undecidable for schemata of formulæ interpreted in the theory of arrays, and we have defined a restricted class of interpretations (called quasi-contiguous) in which satisfiability can be decided in finite time (with a doubly exponential complexity). Future work includes the implementation of our approach. From a more theoretical point of view, it would be interesting to provide a lower bound for the complexity of satisfiability testing for A-formulæ in quasi-contiguous interpretations (only an upper bound is provided in the present paper). We shall also investigate whether the presented results extend to more expressive theories, including for instance a combination of the theory of arrays and Presburger arithmetic, possibly enriched with additional axioms allowing to express general properties of arrays (e.g., to state that an array is sorted or constant on some interval). As evidenced by the examples in Section 5, such properties can sometimes be expressed in our language, using iterated conjunctions, but this is possible only if the considered interval corresponds to a family of constants. Allowing some restricted form of quantification would therefore enhance the expressive power of the language.

An obvious drawback of our approach is that the conditions on the interpretations are of a semantic nature and thus must be checked by the user. We therefore plan to devise syntactic criteria ensuring that the conditions are satisfied (i.e. ensuring that a satisfiable formula has a contiguous model). The examples in Section 5 suggest that this is feasible, at least in simple cases. It would also be interesting to characterize formally the class of programs and properties that can be modeled in our language.

References

1. Abadi, A., Rabinovich, A., Sagiv, M.: Decidable fragments of many-sorted logic. J. Symb. Comput. 45(2), 153–172 (2010)
2. Aravantinos, V., Caferra, R., Peltier, N.: A schemata calculus for propositional logic. In: Giese, M., Waaler, A. (eds.) TABLEAUX 2009. LNCS, vol. 5607, pp. 32–46. Springer, Heidelberg (2009)
3. Aravantinos, V., Caferra, R., Peltier, N.: Decidability and undecidability results for propositional schemata. Journal of Artificial Intelligence Research 40, 599–656 (2011)
4. Aravantinos, V., Echenim, M., Peltier, N.: A resolution calculus for first-order schemata. Fundamenta Informaticae (accepted for publication, to appear, 2013)
5. Aravantinos, V., Peltier, N.: Schemata of SMT-problems. In: Brünnler, K., Metcalfe, G. (eds.) TABLEAUX 2011. LNCS, vol. 6793, pp. 27–42. Springer, Heidelberg (2011)
6. Armando, A., Ranise, S., Rusinowitch, M.: A rewriting approach to satisfiability procedures. Information and Computation 183(2), 140–164 (2003)
7. Baumgartner, P., Fuchs, A., Tinelli, C.: (LIA) - Model Evolution with Linear Integer Arithmetic Constraints. In: Cervesato, I., Veith, H., Voronkov, A. (eds.) LPAR 2008. LNCS (LNAI), vol. 5330, pp. 258–273. Springer, Heidelberg (2008)

8. Bradley, A.R., Manna, Z.: The Calculus of Computation: Decision Procedures with Applications to Verification. Springer-Verlag New York, Inc., Secaucus (2007)
9. Bradley, A.R., Manna, Z., Sipma, H.B.: What's decidable about arrays? In: Emerson, E.A., Namjoshi, K.S. (eds.) VMCAI 2006. LNCS, vol. 3855, pp. 427–442. Springer, Heidelberg (2006)
10. Ghilardi, S., Nicolini, E., Ranise, S., Zucchelli, D.: Decision procedures for extensions of the theory of arrays. Annals of Mathematics and Artificial Intelligence 50, 231–254 (2007)
11. Habermehl, P., Iosif, R., Vojnar, T.: What else is decidable about integer arrays? In: Amadio, R.M. (ed.) FOSSACS 2008. LNCS, vol. 4962, pp. 474–489. Springer, Heidelberg (2008)

TAFA – A Tool for Admissibility in Finite Algebras

Christoph Röthlisberger

Mathematics Institute, University of Bern
Sidlerstrasse 5, 3012 Bern, Switzerland
christoph.roethlisberger@math.unibe.ch

Abstract. Checking whether a quasiequation is admissible in a finitely generated quasivariety is known to be decidable by checking validity in a suitable (finite) free algebra on finitely many generators. Nevertheless this approach is computationally unfeasible since these free algebras can be very big. TAFA is a system providing algebraic tools to search for the smallest set of algebras, according to the standard multiset well-ordering, such that a quasiequation is admissible in the quasivariety if and only if it is valid in this set of algebras.

1 Introduction

A rule is called *admissible* in a logic if it does not produce any new theorems when added to the logic. Admissibility is typically used to establish key properties of systems such as the completeness of a proof system, e.g., by showing that some cut-rule is admissible. Admissible rules may also be used to shorten proofs in a given system for a logic. Admissibility (often alongside equational unification) has been studied for transitive modal logics and intermediate logics in [19,9,10,13,15,7], and various kinds of proof systems to check admissibility in some of these logics have been defined in [11,14,2]. Characterizations of admissibility have also been obtained for certain many-valued logics in [6,17,16,5], but no general approach has until now been developed for the finite setting.

From an algebraic perspective, a quasiequation is *admissible* in a finitely generated quasivariety (which provide algebraic semantics for many-valued logics) if it is valid in a suitable finite free algebra of this quasivariety. This question is decidable and there exist general methods to obtain proof systems for checking validity in finite algebras (see, e.g., [12,1,19]). However, even free algebras on a small number of generators can be very large since the "worst case" grows double-exponentially.

In [18] the theoretical background is given for a general algorithm which outputs a set of algebras that can be used to check admissibility in a more efficient way. The algorithm returns the smallest (according to the standard multiset ordering) set of algebras $\mathcal{K}$ for a given finitely generated quasivariety $\mathcal{Q}$, such that a quasiequation is admissible in $\mathcal{Q}$ if and only if it is valid in all the algebras of $\mathcal{K}$. The main feature of the system TAFA is an implementation of this algorithm as a tool for studying admissibility in a wide range of finite algebras. In particular, rather than generating a tableaux system for a potentially very large free algebra of a specific finite algebra or many-valued logic (which is possible by, e.g., [12]), it suffices to generate a tableaux system for the typically small output algebra(s) of TAFA.

TAFA is also able to solve general algebraic problems like calculating subalgebras, different kinds of morphisms, products, congruences and their lattices and checking properties like being subdirectly irreducible. Please refer to a standard work like [4] for the details of the algebraic definitions used in this paper. TAFA is implemented in Delphi XE2 and is currently only compiled for Windows, but can easily be used on Mac and Linux using an emulator such as Wine. Many ideas concerning the data structures and basic operations are taken from the implementation of the Algebra Workbench [20]. TAFA as well as a copy of [18] are available from `https://sites.google.com/site/admissibility/`.

2 Basic Features of TAFA

In order to use TAFA the user has first either to define the algebras of interest in TAFA or to load some predefined (see *File > Predefined algebras*[1]) or previously stored algebras (from a file). Defining a new algebra (see *File > New algebra*) includes giving it a name, labelling the elements and defining the operations. The user can easily rename, delete or edit algebras, their elements and operations or add some comment by either double clicking the corresponding field of the grid in the main window or using the menu *Edit*.

The main window of TAFA contains a list showing for each algebra its name, cardinality, the names and arities of its operations, and any comments. We say that an entry of a list, e.g., an algebra in the main window, is *selected* if it is highlighted, and *chosen* if the appropriate checkbox is checked.

TAFA can save the chosen algebras as a binary file (*.fab, fast, illegible), as a text file (*.fai, slower, legible) or, if the algebra is a partially ordered set with an operation "meet", to a *.osf file which can be read by the Algebra Workbench to visualize the corresponding Hasse diagram. TAFA loads algebras from fab- or fai-files and is able to copy or remove algebras in the main window (menu *File*).

The algebras are stored as data type TAlgebra within TAFA, which is connected to lists of the type TAlgebraUniverse and TOperationList providing further procedures and objects. Once the algebras of interest are defined in the main window, the basic operations of universal algebra listed below can be performed.

2.1 Homomorphisms

The menu item *Tools > Morphisms* opens a dialogue window, where the user can choose a domain $\mathbf{A}_1$ and a codomain $\mathbf{A}_2$ (of the same language) from the list of defined algebras. It is possible to choose whether to calculate all homomorphisms between $\mathbf{A}_1$ and $\mathbf{A}_2$ or only those that are surjective, injective or bijective.

[1] Navigation through the menus is denoted here by *Menu > Menu item*.

When the button “Calculate” is pressed, TAFA lists the homomorphisms satisfying the chosen criteria. Double-clicking on an entry of the list shows the mappings from elements of $\mathbf{A}_1$ to elements of $\mathbf{A}_2$.

Using the *Tools* menu of this dialogue window it is also possible to add the homomorphic image as a new algebra to the main window or to save the mapping informations to a text file.

2.2 Subalgebras

The menu item *Tools* > *Subalgebras* opens a dialogue window which lists all the subalgebras of the active algebra. The subalgebras are stored as entities of TAlgebraUniverse within this dialogue window to save time (there is no need to build up the operation tables), but it is possible to add the checked subalgebras as new algebras to the main window using the menu *Tools* of the dialogue window. The *Options* menu of the dialogue window offers the possibility to (heuristically) first list the smaller and then the bigger algebras by first calculating the subalgebras generated by zero or one element, storing their sizes and then trying to combine the given generators in such a way that the subalgebras generated are potentially small.

Tools > *Generating subalgebra* opens a dialogue window where the user can choose some elements $a_1, \dots, a_k$ of the active algebra $\mathbf{A}$. TAFA then calculates the unique subalgebra of $\mathbf{A}$ generated by the elements $a_1, \dots, a_k$.

2.3 Products

Having defined algebras $\mathbf{A}_1, \dots, \mathbf{A}_n$ of the same language in the main window of TAFA, the user can calculate the direct product $\mathbf{A}_1 \times \cdots \times \mathbf{A}_n$ using *Tools* > *Direct product*. Specifying some $k \in \mathbb{N}$ with *Tools* > *Direct power*, the direct power $\mathbf{A}^k$ of the selected algebra $\mathbf{A}$ is calculated.

2.4 Congruences

Tools > *Congruences* opens a dialogue window which lists the congruences $\mathrm{Con}(\mathbf{A})$ of the selected algebra $\mathbf{A}$ in the main window. Selecting a congruence in the list shows how the congruence is defined. The dialogue window menu *Tools* lets the user store the congruence lattice, given by $\mathrm{Con}(\mathbf{A})$, as a new algebra (with operations “meet” and “join”) to the main window. It is also possible to quotient the active structure with the selected congruence or to save the congruence informations to a text file.

2.5 Free Algebras

If the set $\mathcal{K} = \{\mathbf{A}_1, \dots, \mathbf{A}_k\}$ of algebras of the same language is chosen in TAFA, the menu item *Tools* > *Free algebra* lets the user specify a natural number $n \in \mathbb{N}$ and TAFA calculates the free algebra of $\mathcal{K}$ on n generators, denoted $\mathbf{F}_{\mathcal{K}}(n)$.

There is also the possibility to search for the *smallest generating free algebra for* $\mathcal{K}$, i.e., the smallest free algebra $\mathbf{F}_{\mathcal{K}}(n)$ of $\mathcal{K}$ such that all $\mathbf{A}$ in $\mathcal{K}$ are homomorphic images of $\mathbf{F}_{\mathcal{K}}(n)$.

3 Validity and Admissibility

Given a set of algebras $\mathcal{K}$ of the same language, the *quasivariety generated by* $\mathcal{K}$, denoted $\mathbb{Q}(\mathcal{K})$, is the smallest class of algebras closed under taking isomorphisms, subalgebras, products and ultraproducts that contains $\mathcal{K}$. Let us assume that $\mathcal{K}$ is a finite set of finite algebras, i.e., $\mathbb{Q}(\mathcal{K})$ is *finitely generated.* $\mathbf{Tm}$ denotes the term algebra over countably infinitely many generators.

We say that a quasiequation $\Sigma \Rightarrow \varphi \approx \psi$, i.e., a set (possibly empty) of equations Σ implying a single equation $\varphi \approx \psi$, is *valid* in the quasivariety $\mathcal{Q}$, written $\Sigma \models_{\mathcal{Q}} \varphi \approx \psi$, if for every $\mathbf{A} \in \mathcal{Q}$ and every homomorphism $h\colon \mathbf{Tm} \to \mathbf{A}$, whenever $h(\varphi') = h(\psi')$ for all $\varphi' \approx \psi' \in \Sigma$, also $h(\varphi) = h(\psi)$.

The quasiequation $\Sigma \Rightarrow \varphi \approx \psi$ is called *admissible* in the quasivariety $\mathcal{Q}$ if for every homomorphism $\sigma\colon \mathbf{Tm} \to \mathbf{Tm}$:

$$\emptyset \models_{\mathcal{Q}} \sigma(\varphi') \approx \sigma(\psi') \text{ for all } \varphi' \approx \psi' \in \Sigma \quad \text{implies} \quad \emptyset \models_{\mathcal{Q}} \sigma(\varphi) \approx \sigma(\psi).$$

It is well known (see, e.g., [19,17,18]) that the quasiequations admissible in $\mathcal{Q} = \mathbb{Q}(\mathcal{K})$ are the quasiequations valid in $\mathbf{F}_{\mathcal{Q}}(n)$ where $n = \max\{|A| : \mathbf{A} \in \mathcal{K}\}$. Hence checking whether a given quasiequation is admissible in $\mathcal{Q}$ is decidable, since $\mathbf{F}_{\mathcal{Q}}(n)$ is finite if $\mathcal{Q}$ is a finitely generated quasivariety (see [3]). We are now interested in the "smallest" set of algebras generating the quasivariety $\mathbb{Q}(\mathbf{F}_{\mathcal{Q}}(n))$, i.e., the smallest set of *admissibility algebras*, to make checking validity faster (since the corresponding proof system will be smaller). One possibility is to make use of the Derschowitz-Manna multiset ordering $\leq_m$ defined in [8] to compare multisets of cardinalities of sets of algebras. We say that a set of finite algebras $\{\mathbf{A}_1, \ldots, \mathbf{A}_n\}$ is a *minimal generating set* for the quasivariety $\mathbb{Q}(\mathbf{A}_1, \ldots, \mathbf{A}_n)$ if for every set of finite algebras $\{\mathbf{B}_1, \ldots, \mathbf{B}_k\}$:

$$\mathbb{Q}(\mathbf{A}_1, \ldots, \mathbf{A}_n) = \mathbb{Q}(\mathbf{B}_1, \ldots, \mathbf{B}_k) \text{ implies } [|A_1|, \ldots, |A_n|] \leq_m [|B_1|, \ldots, |B_k|].$$

To obtain a minimal generating set for $\mathbb{Q}(\mathbf{F}_{\mathbb{Q}(\mathcal{K})}(n))$, we have implemented the algorithm MinGenSet($\mathcal{K}$) (see [18] for details) which returns the (unique up to isomorphism) minimal generating set for the quasivariety $\mathbb{Q}(\mathcal{K})$ generated by a finite set $\mathcal{K}$ of finite algebras.

The algorithm MinGenSet calculates the congruence lattice of the input algebras, which takes exponential time with respect to the size of the input. Therefore we have implemented the algorithm AdmAlgs (see Figure 1), which combines the decomposing of MinGenSet with the fact that a subalgebra $\mathbf{B}$ of $\mathbf{F}_{\mathbb{Q}(\mathcal{K})}(n)$ generates the same quasivariety as $\mathbf{F}_{\mathbb{Q}(\mathcal{K})}(n)$, i.e., $\mathbb{Q}(\mathbf{B}) = \mathbb{Q}(\mathbf{F}_{\mathbb{Q}(\mathcal{K})}(n))$, if every algebra $\mathbf{A}$ of $\mathcal{K}$ is a homomorphic image of $\mathbf{B}$. The idea is to reduce the sizes of the generating algebras (taking subalgebras of the free algebras) while making sure that the algebras are "complex" enough to generate the given quasivariety (checking homomorphisms), and then to remove redundancy using MinGenSet. The algorithm Free($\mathbf{A}, \mathcal{D}$) used in AdmAlgs calculates the smallest free algebra of $\mathcal{D}$ which is a prehomomorphic image of the algebra $\mathbf{A}$ by increasing the number of generators one a time while searching for surjective homomorphisms onto $\mathbf{A}$. The algorithm SubPreHom($\mathbf{A}, \mathbf{B}$) on the other hand returns a proper subalgebra of $\mathbf{B}$ which has $\mathbf{A}$ as a homomorphic image. If there is no such algebra, $\mathbf{B}$ is returned.

```
function AdmAlgs($\mathcal{K}$)
    declare $\mathcal{A}, \mathcal{D}$ : set
    declare $\mathbf{B}, \mathbf{B}'$ : algebra
    $\mathcal{D} \leftarrow$ MinGenSet($\mathcal{K}$)
    $\mathcal{A} \leftarrow \emptyset$
    for all $\mathbf{A} \in \mathcal{D}$ do
        $\mathbf{B} \leftarrow$ Free($\mathbf{A}, \mathcal{D}$)
        $\mathbf{B}' \leftarrow$ SubPreHom($\mathbf{A}, \mathbf{B}$)
        while $\mathbf{B}' \neq \mathbf{B}$ do
            $\mathbf{B} \leftarrow \mathbf{B}'$
            $\mathbf{B}' \leftarrow$ SubPreHom($\mathbf{A}, \mathbf{B}$)
        end while
        add $\mathbf{B}$ to $\mathcal{A}$
    end for
    return MinGenSet($\mathcal{A}$)
end function
```

Fig. 1. Given a finite set $\mathcal{K}$ of finite algebras, return the minimal generating set of the quasivariety $\mathbb{Q}(\mathbf{F}_{\mathbb{Q}(\mathcal{K})}(\omega))$

Given a set $\mathcal{K}$ of algebras chosen in TAFA, the user selects the appropriate free algebra or lets the program find the smallest generating free algebra for $\mathcal{K}$ with *Tools > Admissibility algebra*. The menu *Options* of the dialogue window for calculating admissibility algebras then lets the user choose whether to search admissibility algebras from smaller to larger or with the usual algorithm of searching subalgebras (which is independent of the sizes). Although the latter is much quicker for small algebras, there are some cases where the heuristic method performs faster. Once the admissibility algebra is stored as a new algebra in the main window, the user can run MinGenSet (from the menu *Tools*) to obtain the unique smallest set of admissibility algebras for $\mathcal{K}$.

4 A Case Study: 3-Element Groupoids

In [18], admissibility was studied using TAFA for various algebras and logics with up to five elements and as many operations, e.g., for the Wajsberg algebra corresponding to the 3-valued Łukasiewicz logic, for Kleene and De Morgan lattices and algebras and for Stone algebras.

In order to have a range of "arbitrary" algebras to study we have also considered the groupoids with three elements, i.e., the 3-element algebras with a binary operation. There are 3330 different groupoids up to isomorphism (out of $3^9 = 19683$ in total) for which TAFA calculates the smallest generating free algebra. Figure 2 illustrates the distribution of the cardinalities of these free algebras. The number of generators is not always the same to produce a free algebra of a given cardinality and there are even 16 cases where three generators are needed.

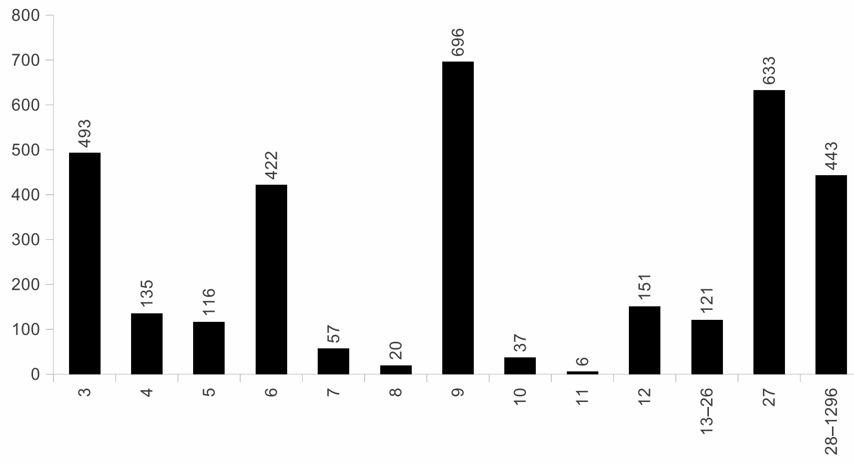


Fig. 2. Cardinality of free algebras (x-axis) and number of groupoids (y-axis)

The main goal was to calculate the smallest set of admissibility algebras for all 3-element groupoids $\mathbf{G}$, namely the results of MinGenSet($\mathbf{F_G}(3)$). For free algebras with less than 25 elements we performed MinGenSet directly, for the larger cases we used AdmAlgs. The admissibility algebras all have fewer than 10 elements. Figure 3 lists the multisets of cardinalities of the minimal generating sets and how many times they occur.

An algebra is called *structurally complete* if the sets of valid and admissible quasiequations coincide for this algebra, and *almost structurally complete*, if these sets coincide for quasiequations with unifiable premises (see [18]). These completeness-checks are accessible in TAFA via the menu *Check*. Performing this check to the groupoids confirmed that 654 of the investigated algebras are not structurally complete, whereas 254 of them are almost structurally complete. The remaining 2676 groupoids are structurally complete.

Cards. of MinGenSet($\mathbf{F_G}(3)$)	2,2	2	3	4	5	6	8	9
Number of groupoids	16	9	2661	90	108	398	18	30

Fig. 3. Cardinalities of the minimal generating sets

Acknowledgements. I would like to thank George Metcalfe for his helpful comments and suggestions and Markus Sprenger for providing the source code of the Algebra Workbench. The author acknowledges support from Swiss National Science Foundation Grant 20002_129507.

References

1. Baaz, M., Fermüller, C.G., Salzer, G.: Automated deduction for many-valued logics. In: Handbook of Automated Reasoning, ch. 20, vol. II, pp. 1355–1402. Elsevier (2001)
2. Babenyshev, S., Rybakov, V., Schmidt, R.A., Tishkovsky, D.: A tableau method for checking rule admissibility in S4. In: Proc. M4M 2009. ENTCS, vol. 262, pp. 17–32 (2010)
3. Birkhoff, G.: On the structure of abstract algebras. Proc. Camb. Philos. Soc. 31, 433–454 (1935)
4. Burris, S., Sankappanavar, H.P.: A Course in Universal Algebra. Graduate Texts in Mathematics, vol. 78. Springer, New York (1981)
5. Cabrer, L.M., Metcalfe, G.: Admissibility via natural dualities. Manuscript
6. Cintula, P., Metcalfe, G.: Structural completeness in fuzzy logics. Notre Dame J. Form. Log. 50(2), 153–183 (2009)
7. Cintula, P., Metcalfe, G.: Admissible rules in the implication-negation fragment of intuitionistic logic. Ann. Pure Appl. Logic 162(10), 162–171 (2010)
8. Dershowitz, N., Manna, Z.: Proving termination with multiset orderings. Commun. ACM 22, 465–476 (1979)
9. Ghilardi, S.: Unification in intuitionistic logic. J. Symbolic Logic 64(2), 859–880 (1999)
10. Ghilardi, S.: Best solving modal equations. Ann. Pure Appl. Logic 102(3), 184–198 (2000)
11. Ghilardi, S.: A resolution/tableaux algorithm for projective approximations in IPC. Log. J. IGPL 10(3), 227–241 (2002)
12. Hähnle, R.: Automated Deduction in Multiple-Valued Logics. Oxford Univ. Press (1993)
13. Iemhoff, R.: On the admissible rules of intuitionistic propositional logic. J. Symbolic Logic 66(1), 281–294 (2001)
14. Iemhoff, R., Metcalfe, G.: Proof theory for admissible rules. Ann. Pure Appl. Logic 159(1-2), 171–186 (2009)
15. Jeřábek, E.: Admissible rules of modal logics. J. Logic Comput. 15, 411–431 (2005)
16. Metcalfe, G.: An Avron rule for fragments of R-mingle. Journal of Logic and Computation (to appear)
17. Metcalfe, G., Röthlisberger, C.: Admissibility in De Morgan algebras. Soft Comput. 16(11), 1875–1882 (2012)
18. Metcalfe, G., Röthlisberger, C.: Admissibility in finitely generated quasivarieties. Logical Methods in Computer Science 9(2:9) (2013)
19. Rybakov, V.: Admissibility of Logical Inference Rules. Studies in Logic and the Foundations of Mathematics, vol. 136. Elsevier, Amsterdam (1997)
20. Sprenger, M.: Algebra Workbench, `http://www.algebraworkbench.net`

Formalizing Cut Elimination of Coalgebraic Logics in Coq

Hendrik Tews

Institute of Systems Architecture, TU Dresden, Germany
`http://askra.de/`

Abstract. In their work on coalgebraic logics, Pattinson and Schröder prove soundness, completeness and cut elimination in a generic sequent calculus for propositional multi-modal logics [1]. The present paper reports on a formalization of Pattinson's and Schröder's work in the proof assistant Coq that provides machine-checked proofs for soundness, completeness and cut elimination of their calculus. The formalization exploits dependent types to obtain a very concise deep embedding for formulas and proofs. The work presented here can be used to verify cut elimination theorems for different modal logics with considerably less effort in the future.

1 Introduction

In [1], Pattinson and Schröder give two generic proofs of cut elimination for propositional multi-modal logics. In their framework a concrete modal logic is specified by a modal similarity type (i.e., a set of modal operators with arity) and a set of one-step rules. The semantics is given by a functor T together with a fibred predicate lifting for each modal operator. Models are T-coalgebras together with a valuation.

Pattinson and Schröder identify semantic conditions that allow them to prove soundness and cut-free completeness. Together, this gives the first cut-elimination theorem. They further identify purely syntactic conditions on the rule set that permit a syntactic cut-elimination proof. A formalization of these proofs in a proof assistant has many benefits beyond the mere validation of [1]. The formalization permits to obtain machine checked cut-elimination proofs for a variety of modal logics by verifying the semantic or syntactic preconditions only. Moreover, if the utilized proof assistant permits the extraction of executable code, then certified tautology checkers can be extracted from the completeness proof. Again, because of the modularity of Pattinson's and Schröder's work, the effort necessary for every new certified tautology checker will be relatively small. Finally, a cut-free calculus provides the foundation for a syntactic proof of Craig's interpolation property (and, indeed, [1] deals with Craig interpolation as an application). A formalization of cut-elimination therefore provides the basis for the verification of the interpolation theorem and for the extraction of certified programs that compute interpolants.

In this paper I describe the formalization of about $2/3$ of the results of [1] in the Coq proof assistant [2,3]. From now on, I refer to the specifications, theorems and proofs in Coq simply as *the formalization.* The formalization covers the generic syntax and semantics of coalgebraic logics, soundness, completeness, semantic and syntactic cut elimination. As an example, I use the modal logic K. The other material of [1], in particular Craig interpolation and the examples of coalition logic and the conditional logics CK and CK + ID are not (yet) contained in the formalization.

The size of the formalization is considerable. There are about 400 definitions and about 1300 theorems and lemmas, which are proved with more than 20, 000 lines of proof script in a total of 36, 000 lines of Coq code. About 6, 000 lines of Coq deal with standard results that are used but not proved in [1] (e.g., completeness and cut elimination for propositional logic). With several years of experience with the proof assistant PVS [4], I missed a few convenient features of the PVS user interface during the work on the formalization. This led to the implementation of automatic library compilation in Proof General [5, Sect. 11.2] and the proof-tree visualization program Prooftree [6]. The complete Coq sources of the formalization and some technical documentation are freely available on the internet [7].

A formalization of this extent does always uncover a number of typos and errors in the formalized work. It is a clear sign for the quality and accurateness of the pen-and-paper proofs of Pattinson and Schröder that I found only 4 errors beyond the level of nitpicking. The most serious one is probably that their substitution lemma 3.14 is wrong: The modal rank does not necessarily decrease as indicated. At first glance this seems to break the induction on modal rank in the completeness and in the cut-elimination theorem. However, with a suitably adapted substitution lemma, these proofs only need minor modifications. All errors that I describe here have been discussed with Dirk Pattinson and Lutz Schröder to confirm that these are indeed errors and not misunderstandings on my side.

Related work. Proof theory and, in particular, cut elimination is a very nice application for theorem proving. The formalization of cut elimination in a proof assistant provides an additional value, because cut-elimination proofs are complex and it is rarely ever the case that all cases are spelled out in pen-and-paper proofs. A long debate about the validity of a cut-elimination proof for the provability logic GL has only recently been resolved [8,9]. Depending on their aims, different authors use different approaches for their formalization of proof theory. Some use a shallow embedding of proofs and formalize only provability without an explicit representation of object-logic proofs (e.g., [10,11]). For cut elimination, one usually prefers a deep embedding of proofs, where object-logic proofs are terms and can be manipulated in the meta logic of the proof assistant (e.g., [9,12]). Formalizations in Isabelle/HOL that use a parametric rule set (e.g., [9,12]) need a well-formedness predicate on proof trees. The formalization presented here uses a deep embedding for formulas and proofs and a shallow embedding for models. The data types for formulas and proofs rely on dependent types to express the necessary side conditions in a very concise way without well-formedness predicates. The present work has some similarities with the work of Chapman [13] in that the formalization covers many different logics. In a sense, the scope of [13] is much broader, because it is not limited to propositional modal logics. However, while Chapman focuses on inversion, the present work proves soundness, completeness and cut elimination. Moreover, the framework of Chapman is not applicable to propositional modal logics, because its modal rules are, in general, not invertible.

Outline. This paper has a clear presentation problem. In order to be self-contained it should comprise the formalized material of [1], which is already at odds with the page limit. Describing several thousand lines of Coq specification and proofs at a level where the reader can follow the development is simply impossible. This paper must therefore focus on a few points of the formalization. Section 2 introduces a few aspects of Coq's logic and specification language. Section 3 presents in detail the deep embedding of coalgebraic logics that is used in the formalization. Section 4.3 high-lights interesting aspects of the formalization. In particular, this section discusses the differences between [1] and the formalization and the errors that I found. Section 5 gives an overview of the main results of the formalization. Section 6 concludes.

Up to the end of Section 3, this paper is self-contained. Section 4.3 and Section 5 can be read at a high level without particular knowledge of coalgebraic logics and without access to the source code of the formalization. However, for accurateness, I provide a few technical details in Section 4.3 that require familiarity with the proofs of [1]. Coq definitions that have been omitted for space reasons can be looked up in the documentation of the formalization [7].

2 Coq Preliminaries

In Coq, **Type** is a keyword that refers to one element in the infinite hierarchy of type universes in Coq. So A : **Type** simply means that A is a type. **Prop** is the type of propositions, which may or may not have a proof. Similar to higher-order logic, a set over A is conveniently modeled as a function $\mathsf{A} \to$ **Prop**.

In contrast to higher-order logic, Coq does not distinguish between types and terms. In Coq, there are only terms and every term has a type, which is a term again. Therefore, application is always written in the usual postfix way, even for terms that represent types at the conceptual level. For instance, as usual, $\mathsf{f\ a}$ stands for the application of function f to argument a. Using the same application, $\mathsf{list\ A}$ stands for the application of the type constructor list to type A, that is, for the lists over A, and $\mathsf{list\ (list\ A)}$ stands for the lists of lists over A.

In Coq, a propositions is a type whose inhabitants are its proofs. A proof for an implication $\mathsf{F} \to \mathsf{G}$ is a function that maps proofs of proposition F to proofs of proposition G. Consequently, the simple arrow denotes both, function types $\mathsf{A} \to \mathsf{B}$ and implications $\mathsf{F} \to \mathsf{G}$.

Frequently used parameters can be declared as **Variable**'s in Coq. They are then automatically added to any definition in which they occur, saving the explicit declaration in each of them. The reader should understand a **Variable** as an arbitrary but fixed element of the given type. The Coq definitions and lemmas included in this paper do usually not mention declared variables. This is not always correct, but hopefully less confusing.

3 A Deep Embedding for Parametric Coalgebraic Logics

This section describes the base definitions for formulas, sequents, proof rules and proofs. The challenge in the formalization is that neither the formula syntax nor the rule set of the object logic is fixed, because the framework of [1] covers many different modal logics.

3.1 Formulas

Pattinson and Schröder take a simple propositional calculus with negation and conjunction and enrich it with modal formulas of the form $\heartsuit(A_1, \ldots, A_n)$, where $\heartsuit$ is a modal operator of arity n and the A_i are arbitrary formulas. The modal operators are drawn from a modal similarity type Λ. The whole development of [1] is parametric in Λ. In Coq, I define the type of Λ as a dependently typed record as follows.

```
Record modal_operators : Type :=
  { operator : Type; arity : operator → nat }.
```

The modal operators are given as an arbitrary type, the field arity determines their arity.

The following piece or source code shows the variable declarations for V, the set of propositional variables, and for L, the modal operators. They are used in almost all files of the formalization. Pattinson and Schröder assume the propositional variables to be a countably infinite set. This assumption will be explicitly added where needed.

```
Variable V : Type.
Variable L : modal_operators.

Inductive lambda_formula : Type :=
  | lf_prop : V → lambda_formula
  | lf_neg : lambda_formula → lambda_formula
  | lf_and : lambda_formula → lambda_formula → lambda_formula
  | lf_modal : forall(op : operator L),
      counted_list lambda_formula (arity L op) → lambda_formula.
```

The keyword **Inductive** introduces an inductive data type that is generated in the usual way from the given constructors. The type of formulas is called lambda_formula here and the constructors have an lf_ prefix, because Pattinson and Schröder use $\mathcal{F}(\Lambda)$ to denote it. The last constructor, lf_modal, for modal formulas, has a dependent type. It maps an operator op and a list of formulas to a new formula. In this paper, I write record selection as function application: operator L selects the type of operators and arity L op applies op to the arity function. The second argument of lf_modal must be a counted_list to ensure that its length matches the arity of op. For a type A and a natural number n, the type counted_list A n contains the lists over A of length n.[1] The use of dependent types is crucial here to capture the meaning of arity for modal operators.

3.2 Sequents

Pattinson and Schröder use a single-sided Gentzen-style sequent system. Sequents are defined as finite multisets of formulas. Multisets can be formalized as functions $A \longrightarrow \mathbb{N}$ or as a quotient type. In Coq both approaches have their drawbacks, because predicate extentionality, function extentionality as well as Hilbert's ϵ operator need additional axioms. I therefore decided to treat sequents as a setoid. A setoid is a type equipped with an equivalence relation, which represents the intended equality. As underlying type I simply use lists of formulas. Two such lists are equivalent, if one is a reordering or permutation of the other. In the formalization, I use the equivalence relation explicitly without relying on the Coq library of setoids.

```
Definition sequent : Type := list lambda_formula.

Inductive list_reorder(A : Type) : list A → list A → Prop :=
  | list_reorder_nil : list_reorder [] []
  | list_reorder_cons : forall(a : A)(l1 l2 : list A)(n : nat),
      list_reorder l1 l2 → list_reorder (a :: l1) ((firstn n l2) ++ a :: (skipn n l2)).
```

The equivalence relation on sequents is called list_reorder, because it is used for other types as well. It is defined here as an inductive relation on lists of an arbitrary type. The first constructor proves that the empty list is a reordering of itself. The second constructor proves that, whenever l1 is a reordering of l2, then also a :: l1 is a reordering of the list obtained by inserting a at an arbitrary position in l2. The functions firstn and skipn are from the Coq library. They return and cut off, respectively, the first n elements of a list; ++ denotes list concatenation.

The advantage of using lists of formulas as sequents is its simplicity. Many proofs can simply be done by induction on the list structure. The disadvantage is that the intended equality on sequents is not builtin: It always needs explicit treatment and, if forgotten, it may happen that a property holds for one sequent but not for some reordering of it.

[1] See [7] for the definition of counted_list and some other basic Coq material.

$$\frac{}{\vdash \Gamma, p, \neg p}\,(\mathrm{Ax}) \qquad \frac{\vdash \Gamma, A \qquad \vdash \Gamma, B}{\vdash \Gamma, A \wedge B}\,(\wedge) \qquad \frac{\vdash \Gamma, \neg A, \neg B}{\vdash \Gamma, \neg(A \wedge B)}\,(\neg\wedge)$$

$$\frac{\vdash \Gamma, A}{\vdash \Gamma, \neg\neg A}\,(\neg\neg) \qquad \frac{\vdash \Gamma, A \qquad \vdash \Delta, \neg A}{\vdash \Gamma, \Delta}\,(\mathrm{cut})$$

Fig. 1. Propositional rules

3.3 Rules and Rule Sets

A proof rule is a record with the assumptions and the conclusion.

```
Record sequent_rule : Type := {assumptions: list sequent; conclusion: sequent}.
```

Pattinson and Schröder do not distinguish between rules and rule instances: Proofs may only contain rules that appear literally in the respective rule set. For the propositional part of the calculus Pattinson and Schröder use the rule schemata in Figure 1. Because of the generice nature of [1], the modal rules of the calculus are not specified. Pattinson and Schröder only require that modal rules are *one-step rules*, see [1, Def. 3.3]. A one-step rule with k assumptions looks as follows:

$$\frac{\vdash a^1_1, \ldots, a^1_{n_1},\ \neg b^1_1, \ldots, \neg b^1_{m_1} \qquad \cdots \qquad \vdash a^k_1, \ldots, a^k_{n_k},\ \neg b^k_1, \ldots, \neg b^k_{m_k}}{\vdash \heartsuit_1(\ldots), \heartsuit_2(\ldots), \ldots,\ \neg\heartsuit'_1(\ldots), \neg\heartsuit'_1(\ldots), \ldots}$$

A rule of this form must fulfill 4 conditions in order to be a one-step rule: (1) all a^i_j and b^i_j must be variables, (2) the conclusion must not be the empty sequent, (3) all arguments of the modal operators in the conclusion must be (non-negated) variables and, finally, (4) all variables of the assumptions must appear in the conclusion.[2] In the framework of Pattinson and Schröder, a specific logic is specified by a set $\mathbf{R}$ of one-step rules, among others. Proofs may contain rules of the set $\mathcal{S}(\mathbf{R})$ of weakened substitution instances of $\mathbf{R}$. The set $\mathcal{S}(\mathbf{R})$ contains all rules $\Gamma_1\sigma\ \ldots\ \Gamma_n\sigma\ /\ \Gamma_0\sigma, \Delta$ for a one-step rule $\Gamma_1 \ldots \Gamma_k/\Gamma_0 \in \mathbf{R}$, an arbitrary substitution σ and an arbitrary weakening context Δ [1, Def. 3.5].

Rules are formalized as predicates on the type sequent_rule. For instance, for the $(\wedge)$-rule we have the following definition.

```
Definition is_and_rule(r : sequent_rule) : Prop :=
  exists(sl sr : sequent)(f1 f2 : lambda_formula),
    assumptions r = [sl ++ f1 :: sr; sl ++ f2 :: sr] ∧
    conclusion r = sl ++ (lf_and f1 f2) :: sr.
```

It is easy to see that is_and_rule is closed under sequent reordering in the following sense: Let s be the conclusion of a rule r, then, for every reordering s' of s there exists a rule r' such that s' is the conclusion of r'. This property is called rule_multiset in the formalization. It is proved for all rule sets and ensures that provability is closed under reordering.

[2] Condition (4) is missing in [1, Def. 3.3], see Section 4.3 below.

3.4 Proofs

To avoid confusion, one must distinguish between *meta-logic proofs* and *object-logic proofs*. The former are proofs in Coq (the meta logic), to establish properties of the latter. An object-logic proof is a proof in some coalgebraic logic.

Object-logic proofs are finite trees made of rule applications and hypotheses. Because Pattinson and Schröder frequently change the rule set and the hypotheses, I decided to make object-logic proofs parametric in the hypotheses and the rule set. Cut elimination is the main concern of the formalization. I therefore define object-logic proofs as a data type, whose elements can be manipulated by functions and meta-logic proofs. In the sense of [9] I use a deep embedding for derivations and rules (and variables).

```
Inductive proof(rules : set sequent_rule)(hypotheses : set sequent)
                                                  : sequent → Type :=
  | assume : forall(gamma : sequent),
      hypotheses gamma → proof rules hypotheses gamma
  | rule : forall(r : sequent_rule), rules r →
      dep_list sequent (proof rules hypotheses) (assumptions r) →
        proof rules hypotheses (conclusion r).
```

The type constructor for object-logic proofs takes three arguments: proof r h s is the type of proof trees with conclusion sequent s using rules and hypotheses from r and h, respectively. In a given proof tree, the sets of rules and hypothesis are constant throughout the tree (because these arguments are before the colon in the inductive definition). In contrast, the sequent may change: a proof tree of type proof r h (lf_and f g) typically contains subtrees of type proof r h f and proof r h g.

The constructor assume is for hypothesis leafs in the proof tree. It takes two arguments: the hypothesis gamma and a proof that gamma is indeed a member of the hypotheses. The constructor rule is for rule applications. It takes three arguments: a rule r, a proof that r is in the set of rules and a list of sub-proofs, one for each assumption of r. With all arguments present, it constructs a new proof tree for the conclusion of r.

The type dep_list of dependently typed lists, which occurs in the third argument of constructor rule, is slightly involved. Let T be a type constructor of arity one, A be a type and [a_1; a_2; ...; a_n] be a (conventional) list over A. Then, dep_list A T [a_1; a_2; ...; a_n] is a list of length n with the first element having type T a_1, the second having type T a_2, and so on until the last element of type T a_n. In the definition of object-logic proofs above, T is the partial application (proof rules hypothesis) that maps any sequent s to the type of proof trees with conclusion s. Therefore, dep_list sequent (proof rules hypotheses) (assumptions r) is the type of an inhomogeneous list that contains one proof for each assumption of the rule r.

3.5 Provability

Provability in the object logic is now straight-forward:

```
Definition provable(rules : set sequent_rule)(hypotheses : set sequent)
                                                  (s : sequent) : Prop :=
  exists(p : proof rules hypotheses s), True.
```

Provability is not closed under reordering of the conclusion. This property is established as lemma.

```
Lemma multiset_provability :
  forall(rules : set sequent_rule)(hypothesis : set sequent)(s r : sequent),
    rule_multiset rules →
    sequent_multiset hypothesis →
    list_reorder s r →
    provable rules hypothesis s →
      provable rules hypothesis r.
```

Here, sequent_multiset ensures that the set of hypothesis is closed under reordering and rule_multiset ensures the same for rules, as explained before.

It is worth noting again the succinctness of the deep embedding of coalgebraic logics in Coq. The definitions can be expressed in less than 20 lines and rely only on the type constructors list, dep_list and counted_list, where the first is from the Coq standard library and the other two are well-known under various names in the Coq literature. The dependent typing handles all side conditions. Separate predicates to ensure well-formedness of formulas and proofs are not necessary.

4 Highlights of the Formalization

This section presents some interesting aspects of the formalization, including the differences between [1] and the formalization and the few errors that the formalization revealed. Readers not familiar with [1] can safely skip over the technical details, which are only provided here for accurateness. Missing Coq definitions are described at a high level, readers interested in the source code are referred to [7].

4.1 Insufficient Intuitionistic Meta Logic

The object logic of Pattinson and Schröder is a classical logic and they also use classical logic in their reasoning. The logic of Coq is, however, intuitionistic. In Coq, neither $\neg\neg P \rightarrow P$ nor $P \vee \neg P$ can be proved in general. Obviously, one has to expect, that some results of Pattinson and Schröder are not provable in Coq. One can make Coq classical, by assuming, for instance, **forall**(P : **Prop**), ¬ (¬ P) → P as an axiom, which is available in a certain module of the standard library. Instead of the axiom, I prefer to use a property that must be explicitly listed in the assumptions of those results that depend on classical logic. The property, which is called classical_logic, is clearly visible in the sources and one can easily determine why theorems depend on it.

The points where classical reasoning is needed depends crucially on the encoding of sequents into formulas and on the fact that disjunction is encoded as negated conjunction in the object logic. The encoding of sequents into formulas is used for the semantics of sequents. Pattinson and Schröder associate the formula $\breve{\Gamma} = \bigvee \Gamma$ with the sequent Γ and set $[\![\Gamma]\!] = [\![\breve{\Gamma}]\!]$. Defining the finite disjunction $\bigvee \Gamma$ with an existential quantifier (i.e., $(A_1, \dots, A_n)\breve{} = \exists i \;.\; A_i$) is inappropriate, because the object logic does not contain quantification. I therefore use an iterative definition (i.e., $(A, \Gamma)\breve{} = A \vee \breve{\Gamma}$) which results in $(A_1, A_2)\breve{} = \neg(\neg A_1 \wedge \neg A_2)$, because disjunction is syntactic sugar in the object logic.

For sequents with two or more formulas, the translation into formulas leads to some kind of Gödel-Gentzen double-negation translation. Therefore, a bit unexpected, the (Ax) rule can be proved sound without using classical_logic, because $\neg(\neg P \wedge \neg\neg P)$ is an intuitionistic tautology (contrary to $P \vee \neg P$).

In contrast, the soundness of the (cut) rule depends on classical_logic. Consider the case where Γ is the empty sequent and Δ contains one formula B. Then, soundness of (cut) amounts to $A \wedge \neg(\neg B \wedge \neg\neg A) \to B$, which is not an intuitionistic tautology, in contrast to $A \wedge (B \vee \neg A) \to B$.

The second point where classical_logic is needed in the formalization is the upward correctness of the $(\neg\neg)$ rule, which is needed in the completeness proof. However, classical reasoning is here only required for the case where Γ is empty. For non-empty Γ the double-negation translation makes the statement provable.

Because of the effects of the double-negation translation, the soundness of the calculus with the (cut) rule and the completeness depend on classical_logic. Soundness without (cut) can be proved in intuitionistic logic.

There is only one third point in the whole formalization that requires classical_logic. This is a technical point inside Proposition 4.13, which is the base result for the completeness proof.

4.2 Differences in the Formalization

This subsection describes the important differences between the formalization and [1] and mentions some other noteworthy points. Errors and omissions that the formalization revealed are discussed in the next subsection.

Non-Negative Modal Rank. The modal rank of a formula or sequent is the maximal nesting level of modal operators in it. Purely propositional formulas have modal rank 0. Many proofs in [1] work by induction on the modal rank. Pattinson and Schröder occasionally use the modal rank -1 to avoid a case distinction, see for instance [1, Lem. 3.7]. No formula has rank -1.

For simplicity, the formalization uses natural numbers for the modal rank. To accommodate -1, the modal rank in the formalization is increased by one. That is, purely propositional formulas have rank 1, the formula $\heartsuit(p)$ has rank 2 for a propositional variable p, and so on. No formula has rank 0, but the empty sequent has rank 0.

Unused Results with Difficult Proofs. A few lemmas have been omitted from the formalization, mostly because no other results depend on them and they have a relatively difficult proof. One example is point 2 of Proposition 3.2, which recalls that the propositional rules including (cut) are complete with respect to propositional consequence. The proof of this result requires compactness, which is difficult to capture in the intuitionistic logic of Coq. The second example is the depth-preservation proof of Lemma 3.13, which is missing in the paper and which I discuss in the next subsection. The last example is Proposition 4.5 in [1] about *one-step completeness*. In coalgebraic logics, one-step completeness is a technical condition on the modal rules that implies completeness of the whole calculus. Proposition 4.5 states that it is sufficient to consider finite sets only for one-step completeness. This proposition is apparently only needed for the example of coalition logic.

Changes in the Syntactic Cut-Elimination Theorem. In [1], Proposition 5.6 for syntactic cut elimination states three properties together. First the admissibility of the non-atomic axiom rule, second the admissibility of contraction and third the admissibility of cut-elimination. Pattinson and Schröder prove all three properties in *one* mutual induction on the modal rank. In their proof, the induction step for non-atomic axioms of rank $n + 1$ depends on cut elimination on rank n.

In the formalization I use two substitution lemmas (which are both derived from a more general result). One for the rule set including (cut) and one for the rule set without (cut). The latter one permits me to eliminate cut from the rule set before applying the substitution lemma. Then, the proof for non-atomic axioms in the formalization only requires cut elimination on purely propositional formulas of rank 0. Therefore, the result for non-atomic axioms is a separate proposition in the formalization, which is proved before the remainder of 5.6.

Injective Substitutions. Pattinson and Schröder use injective substitutions at two points in the syntactic cut-elimination proof, because injective substitutions preserve inclusion of multisets under certain conditions. (More accurately, $\Gamma \subseteq \Delta$ implies $\Gamma\sigma \subseteq \Delta\sigma$ for sequents Γ and Δ when σ is injective, Γ is a conclusion of a one-step rule and $\subseteq$ denotes inclusion on multisets.) For obtaining an injective substitution, they write, "*We may factorise $\sigma = \sigma_m \circ \sigma_e$ where σ_e is a renaming and σ_m is an injective substitution*" [1, page 29]. To avoid non-constructive definitions, I use a slightly weaker factorization. For a sequent Γ and a substitution σ I construct an injective σ'_m and a renaming σ'_e such that only $\Gamma\sigma = \Gamma\sigma'_e\sigma'_m$, while, in general, $\sigma \neq \sigma'_m \circ \sigma'_e$. Nevertheless, the cited sentence is one of the sentences with the biggest formalization overhead that I encountered. It required about 1500 lines of Coq and one week to construct σ'_m and σ'_e out of σ and to prove the necessary properties.

4.3 Omissions and Errors

In this subsection I discuss the non-trivial problems in the formal development of [1]. During the intense work on the formalization I also discovered a number of missing side conditions and obviously missing assumptions. These points are not included here. The fact that there are only 4 non-trivial problems in the proofs of [1] and that they have only negligible consequences for the main theorems, shows the accuracy of the pen and paper proofs of Pattinson and Schröder.

One Step Rules. The definition of one-step rules in [1] omits a side condition on the propositional variables: Just as described in Section 3.3, one must actually require that the assumptions do only use propositional variables that do appear in the conclusion. This condition is needed for those proofs that proceed by induction on the modal rank. For a substitution instance of a one-step rule, these proofs simply invoke the induction hypothesis on the assumptions of the rule. For this the modal rank of the assumptions must be smaller than the one of the conclusion. The simplest way to ensure this on substitution instances of one-step rules is the side condition on propositional variables.

In Coq, the fixed definition looks as follows:

```
Definition one_step_rule(r : sequent_rule) : Prop :=
  every_nth prop_sequent (assumptions r) ∧
  simple_modal_sequent (conclusion r) ∧
  conclusion r ≠ [] ∧
  every_nth
    (fun(s : sequent) ⇒
           incl (prop_var_sequent s) (prop_var_sequent (conclusion r)))
    (assumptions r).
```

The predicate every_nth P l is equivalent to Forall[3] from Coq's standard library, using a different and, for my purposes, more convenient definition. It expresses that P holds on all elements of the list l. The predicates prop_sequent and simple_modal_sequent express the constraints on the shape of the formulas in the assumptions and the conclusion, respectively. The predicate incl l1 l2 from the standard library holds if every element in l1 appears in l2 (regardless of multiplicity and order). The function prop_var_sequent : sequent → list V collects the propositional variables in a sequent.

Missing Proof for Depth Preservation. The inversion Lemmas 3.12 and 3.13 of [1] state that the inverted rules of ($\wedge$), ($\neg\wedge$) and ($\neg\neg$) are *depth-preserving admissible*. This means, for instance, for the ($\wedge$) rule, that, if $\Gamma, A \wedge B$ is provable, then so are Γ, A and Γ, B with proof trees of the same or smaller size. The Lemmas 3.12 and 3.13 differ in the rule set for which they make this statement. Lemma 3.12 makes the statement for proofs using the propositional rules only while Lemma 3.13 applies to proofs using propositional as well as modal rules.

[3] Note the case! Forall differs from the keyword **forall**.

The proof of Pattinson and Schröder for 3.13 uses their Lemma 3.9, which states an equivalence of proofs for the two different rule sets and relies then on 3.12. The problem here is that their Lemma 3.9 makes no statement about the size of the proof trees. So the proof of Pattinson and Schröder proves the inversion property, but not the depth-preserving part of the statement.

Depth preservation is important for the syntactic cut-elimination proof, because this proof uses induction on the size of the proof tree. However, in the syntactic cut-elimination proof only 3.12 is needed. Lemma 3.13 is (apparently) never used. In the formalization of Lemma 3.13 I only prove the inversion property and omit the depth-preservation part.

Fixed Substitution Lemma. The substitution lemma 3.14 of Pattinson and Schröder makes the following statement. Assume that Γ is provable with rules of modal rank at most n (implying that Γ has rank n) and that σ is a substitution that maps propositional variables to formulas of modal rank at most k (i.e., σ has rank k). Then $\Gamma\sigma$ is provable with rules of modal rank $n+k$, using additional assumptions from the set $\mathrm{Ax}_k = \{\neg A, A, \Delta \mid A$ a formula of rank k, Δ a sequent of rank $k\}$. The proof is very simple: One takes the same proof tree and substitutes a suitable element from Ax_k for every occurrence of the (Ax) rule. Consider for instance $\Gamma = \neg p, p, \heartsuit(p)$ of rank 1, which can directly be proved with the (Ax) rule, and the substitution σ of rank 1 that maps p to $\heartsuit(p)$. Then $\Gamma\sigma = \neg\heartsuit(p), \heartsuit(p), \heartsuit(\heartsuit(p))$ should match an assumption from Ax_1, which is impossible, because $\Gamma\sigma$ has rank 2.

The substitution lemma is used inside induction proofs on the modal rank for sequents Γ of rank 1 and substitutions σ of rank n. The idea is to reduce the modal rank $n+1$ of $\Gamma\sigma$ to rank n of the elements of Ax_n, making it possible to apply the induction hypothesis to the elements of Ax_n. Therefore, the trivial change of permitting sequents of rank $n+k$ in the set Ax in the substitution lemma would fix the problem, but make the lemma useless.

For the formalization I define, for an arbitrary substitution σ

$$\mathrm{Ax}^n_\sigma \;=\; \{\neg p\sigma, p\sigma, \Delta \mid\ p \text{ a propositional variable, } \Delta \text{ a sequent of rank } n\}$$

In the substitution lemma, the proof of $\Gamma\sigma$ is permitted to use assumptions from Ax^{n+k}_σ, where n is the rank of Γ and k is the rank of σ, as before. In the proofs using the substitution lemma, one can apply the induction hypothesis on the two-element sequent $\neg p\sigma, p\sigma$, which has rank k only, and then use a suitable weakening lemma to obtain $\neg p\sigma, p\sigma, \Delta$.

The σ parameter in the set Ax^n_σ conveys some information through the application of the substitution lemma. This makes it possible to use the substitution lemma inside the proof of point 1 of Proposition 5.6 in [1], which states the admissibility of the non-atomic axiom rule. Pattinson and Schröder prove a special claim there by induction on the proof tree.

A Gap in the Completeness Proof. Proposition 4.13 in [1] states completeness for rank n, that is, if Γ of modal rank n is valid in the special n-step semantics, then it can be proved with rules of rank n. The proposition makes the statement actually twice, for the rule set including (cut) and, with stronger assumptions, for the rule set without (cut). We focus here on the proof for the rule set including the (cut) rule. The proof proceeds by induction on the modal rank of Γ. Inside the induction step the obligation to find a proof for Γ is made simpler by reducing the complexity of Γ step by step. In the first step, the propositional rules are applied until Γ has the form

$$\neg\heartsuit_1(\ldots),\ldots,\neg\heartsuit_k(\ldots),\ \heartsuit'_1(\ldots),\ldots,\heartsuit'_{k'}(\ldots),\ \neg q_1,\ldots,\neg q_m,\ q'_1,\ldots,q'_{m'} \qquad (*)$$

Pattinson and Schröder make now a case distinction: Either the left part with the modal formulas is valid or the right part with the propositional variables. In case of the right part one can simply use the (Ax) rule to construct the needed proof. In case of the left part one can use the one-step completeness of the rule set (which is an assumption of the proposition) and the induction hypothesis to obtain a proof for

$$\neg\heartsuit_1(\ldots),\ldots,\neg\heartsuit_k(\ldots),\ \heartsuit'_1(\ldots),\ldots,\heartsuit'_{k'}(\ldots) \qquad (\dagger)$$

The gap that remains in the proof of Pattinson and Schröder is how to obtain a proof of $(*)$ from a proof of $(\dagger)$. One obviously only needs a weakening lemma for the rule set including (cut). However, this result is missing from [1]. The weakening lemma 3.11 of [1] states weakening only for the rule set *without* (cut).

Weakening can be obtained with (cut) in a simple way, however, this requires the admissibility of non-atomic axioms. Non-atomic axioms are admissible, but this result is only proved much later in 5.6 and not available at this point. For the formalization I therefore proof the required weakening lemma by induction on the proof tree without using non-atomic axioms.

5 Main Theorems in the Formalization

This section presents the Coq source code of a few high-level theorems in the formalization. For space reasons, missing definitions must be looked up in the source code [7] if the explanations do not suffice.

The definitions and lemmas that deal with the semantics of coalgebraic logics need as third parameter a functor T, which is declared as **Variable**, similar to V and L.

```
Variable T : functor.
```

A functor is a record containing two functions, one for a mapping on types (the objects) and one for the mapping on functions (the morphisms). Additionally, functor contains proofs for the relevant properties, such as, for instance, the preservation of identity morphisms.

The first theorem shown is the completeness result without (cut).

```
Lemma cut_free_completeness :
  forall(enum_V : enumerator V)(LS : lambda_structure)
        (rules : set sequent_rule)(osr : one_step_rule_set rules)(s : sequent),
    classical_logic →
    non_trivial_functor T →
    one_step_cut_free_complete (enum_elem enum_V) LS rules osr →
    valid_all_models (enum_elem enum_V) LS s →
      provable (GR_set rules) empty_sequent_set s.
```

Here, enum_V is an enumerator (i.e., an injective function nat → V) for the variables. It is only needed for constructing substitutions inside the proof.[4] The lambda structure LS contains a predicate lifting of the functor T for each modal operator in the modal similarity type L together with the necessary properties. These predicate liftings are used for the semantics of the modal operators. The universally quantified variable osr is a proof for the fact that rules forms a set of one-step rules. This property appears as a quantified variable instead of as an assumption, because it is needed as argument of one_step_cut_free_complete, which expresses that the rule set rules is one-step cut-free complete with respect to the lambda structure LS. The predicate valid_all_models ensures that the sequent s is valid in all models, while non_trivial_functor T ensures that there is at least one such model. The term (enum_elem enum_V) produces one variable as witness that the set of variables is not empty. Both one_step_cut_free_complete and valid_all_models are only well-formed for non-empty variables sets V.

The next theorem is semantic cut elimination. Its proof first uses the soundness of the logic to derive the validity of those sequents that possess a proof. It then relies on cut_free_completeness to prove the existence of a cut-free proof.

```
Theorem semantic_admissible_cut :
  forall(enum_V : enumerator V)(LS : lambda_structure)
        (rules : set sequent_rule)(osr_prop : one_step_rule_set rules),
    classical_logic →
    non_trivial_functor T →
    one_step_sound (enum_elem enum_V) LS rules osr_prop →
    one_step_cut_free_complete (enum_elem enum_V) LS rules osr_prop →
      admissible_rule_set (GR_set rules) empty_sequent_set is_cut_rule.
```

Here, we have the additional assumption one_step_sound that ensures the one-step soundness and thereby soundness. The predicate is_cut_rule captures all instances of (cut) and admissible_rule_set R H C expresses that all rules in C are admissible for the rule set R and the assumptions H.

Finally, here is the syntactic cut elimination theorem. Syntactic cut elimination works by moving applications of the cut rule upwards in the proof until they finally disappear. The theorem depends on two additional **Variables**: a decidable equality relation on the operators and on the propositional variables.

[4] Actually, all proofs in the formalization only require finitely many distinct variables. The number of variables needed depends on the syntactic structure of the sequent s. Just like Pattinson and Schröder I simply assume infinitely many variables, because a suitable finite upper bound has not been identified yet.

```
Variables (op_eq : eq_type (operator L)) (v_eq : eq_type V).

Theorem syntactic_admissible_cut : forall(rules : set sequent_rule),
  countably_infinite V → one_step_rule_set rules →
  absorbs_congruence rules →
  absorbs_contraction op_eq v_eq rules →
  absorbs_cut op_eq v_eq rules →
    admissible_rule_set (GR_set rules) empty_sequent_set is_cut_rule.
```

This theorem also needs an enumerator for V (provided by countably_infinite) and the one-step property for rules, but here these points appear as conventional assumptions. The other assumptions are the three absorption properties, where the latter two need the decidable equalities.

Comparing the two cut elimination statements, we see that the syntactic one can be proved in intuitionistic logic and makes no assumptions on the functor T.

As an example, the formalization currently contains only the modal logic K (see e.g., [14]). Its purpose is to ensure that the general results of the formalization are applicable to a concrete logic and that all assumptions can be discharged as expected. For this example, natural numbers are used as propositional variables and the only modal operator $\Box$ is defined with an inductive date type. The example contains an application of each of the main results of the formalization. Here I only show syntactic cut elimination.

```
Theorem k_syntactic_cut :
  admissible_rule_set (GR_set k_rules) empty_sequent_set is_cut_rule.
```

This theorem uses the equivalent but non-standard rule set k_rules, which permits cut elimination, see [1, Ex. 4.6]. The theorem is proved with the theorem syntactic_admissible_cut and suitable lemmas for the absorption properties of K.

6 Conclusion and Future Work

This paper presents the formalization of about 2/3 of [1] in the proof assistant Coq. The formalization contains the necessary definitions to formalize and prove the results on soundness, completeness and cut elimination of coalgebraic modal logics. The formalization contains the modal logic K as example, ensuring that definitions and theorems can be employed. Using this formalization, it should be possible to obtain machine checked cut-elimination proofs and certified tautology checkers for a number of different modal logics with relatively little effort.

There are many interesting directions for continuing the work presented here. First, it would be nice to cover more examples in order to obtain machine checked cut-elimination theorems for a number of different modal logics. Second, it would be interesting to also formalize the remainder of [1], in particular the results on the interpolation property. The third point are certified programs, for instance, for checking tautologies in a particular modal logic. From definitions and proofs, Coq can extract Haskell or OCaml programs, which are correct by construction. Because the completeness proof of Pattinson and Schröder is constructive, one should be able to obtain a tautology checker from it. In the current form of the formalization, program extraction does not work, because the completeness result is formulated as theorem only. For program extraction one must restructure the completeness result into the function that constructs the proof and a correctness proof of that function.

Acknoledgements. I thank Dirk Pattinson and Lutz Schröder for several discussions on their paper.

References

1. Pattinson, D., Schröder, L.: Cut elimination in coalgebraic logics. Information and Computation 208, 1447–1468 (2010)
2. The Coq development team: The Coq proof assistant reference manual. LogiCal Project, Version 8.4 (2012)
3. Bertot, Y., Castéran, P.: Interactive Theorem Proving and Program Development. In: Coq'Art: The Calculus of Inductive Constructions. Texts in Theoretical Computer Science. Springer (2004)
4. Owre, S., Rajan, S., Rushby, J., Shankar, N., Srivas, M.: PVS: Combining specification, proof checking, and model checking. In: Alur, R., Henzinger, T.A. (eds.) CAV 1996. LNCS, vol. 1102, pp. 411–414. Springer, Heidelberg (1996)
5. Aspinall, D., Kleymann, T.: User Manual for Proof General 4.2. LFCS Edinburgh (September 2012), `http://proofgeneral.inf.ed.ac.uk`
6. Tews, H.: Automatic library compilation and proof tree visualization for Coq Proof General. Presentation at the 3rd Coq Workshop, Nijmegen (2011)
7. Tews, H.: Formalized Cut Elimination of Coalgebraic Logics: Source Code and Documentation. TU Dresden (April 2013), `http://askra.de/science/coalgebraic-cut`
8. Goré, R., Ramanayake, R.: Valentini's cut-elimination for provability logic resolved. In: Areces, C., Goldblatt, R. (eds.) Advances in Modal Logic, pp. 67–86. College Publications (2008)
9. Dawson, J.E., Goré, R.: Generic methods for formalising sequent calculi applied to provability logic. In: Fermüller, C.G., Voronkov, A. (eds.) LPAR-17. LNCS, vol. 6397, pp. 263–277. Springer, Heidelberg (2010)
10. Doczkal, C., Smolka, G.: Constructive completeness for modal logic with transitive closure. In: Hawblitzel, C., Miller, D. (eds.) CPP 2012. LNCS, vol. 7679, pp. 224–239. Springer, Heidelberg (2012)
11. Chapman, P., McKinna, J., Urban, C.: Mechanising a Proof of Craig's Interpolation Theorem for Intuitionistic Logic in Nominal Isabelle. In: Autexier, S., Campbell, J., Rubio, J., Sorge, V., Suzuki, M., Wiedijk, F. (eds.) AISC/Calculemus/MKM 2008. LNCS (LNAI), vol. 5144, pp. 38–52. Springer, Heidelberg (2008)
12. Dawson, J.E., Goré, R.: Formalised cut admissibility for display logic. In: Carreño, V.A., Muñoz, C., Tahar, S. (eds.) TPHOLs 2002. LNCS, vol. 2410, pp. 131–147. Springer, Heidelberg (2002)
13. Chapman, P.: Tools and techniques for formalising structural proof theory. PhD thesis, University of St Andrews (June 2010), `http://hdl.handle.net/10023/933`
14. Blackburn, P., de Rijke, M., Venema, Y.: Modal Logic. Cambridge Tracts in Theoretical Computer Science. Cambridge University Press (2002)

Intelligent Tableau Algorithm for DL Reasoning

Ming Zuo and Volker Haarslev

Concordia University, Montreal QC H3G 1M8, Canada
{ming_zuo,haarslev}@encs.concordia.ca

Abstract. Although state-of-the-art description logic (DL) reasoners are equipped with a comprehensive set of optimizations, reasoning performance is still a major bottleneck in both research and real world applications. In this paper, we propose a sound and complete algorithm called the intelligent tableau algorithm by incorporating comprehensive learning techniques to tackle all DL reasoning tasks. We also provide a reference implementation reasoner called LIGHT for the DL $\mathcal{ALC}$ dialect based on the algorithm we developed. Preliminary tests indicate that significant improvements can be achieved, i.e., compared to other state-of-the-art reasoners, LIGHT is up to two orders of magnitude faster for simple problems and several orders of magnitude faster for more difficult problems. Even though in this work our discussion is restricted to the $\mathcal{ALC}$ reasoning problem, our conjecture is that the algorithm developed can easily be extended to super-logics of $\mathcal{ALC}$.

Keywords: description logic, automated reasoning, learning, forgetting.

1 Introduction

Most state-of-the-art DL reasoners implement tableau-based decision procedures which typically check the consistency of an ontology by constructing a so-called pre-model for the ontology. These procedures create pre-models in an often blind way which highly depends on the syntax of the input ontologies. Despite many optimization techniques studied and implemented so far, it is easy to find ontologies where one reasoner performs very well while the other is hopelessly inefficient (e.g., see [5] for combinations of nominals and qualified cardinality restrictions).

To simplify our discussion in the following sections, we restrict our research scope in this work on the DL dialect $\mathcal{ALC}$ (Attributive Concept Language with Complements) which is a subset of nearly every expressive DL [2]. Applications of the results introduced in this work to more expressive DL dialects will be addressed in our future work.

The paper is structured as follows. We first briefly introduce the syntax and semantics of $\mathcal{ALC}$ and its relationship with other logics. Then we study a sound and complete reasoning procedure based on a special DL normal form (DLNF). Afterward we discuss the integration of different types of learning into the reasoning procedure to come up with the so-called intelligent reasoning algorithm. At last, we show that any TBox can be converted into DLNF easily. The effectiveness of the algorithm proposed in this work is demonstrated by empirical results obtained from processing a number of typical test cases based on our reference implementation.

1.1 $\mathcal{ALC}$ Description Logic

Let A be a concept name (atomic concept), C and D are arbitrary concepts, and R a role name (atomic role). In $\mathcal{ALC}$, concepts are formed with the syntax as following:

$$C, D ::= \top \mid \bot \mid A \mid \neg C \mid C \sqcap D \mid C \sqcup D \mid \forall R.C \mid \exists R.C$$

where $\top$ is the abbreviation of $\neg A \sqcup A$, and $\bot$ of $\neg A \sqcap A$. An atomic concept corresponds to a unary relation in first order predicate logic (FOL), and an atomic role to a binary relation in FOL. If C and D are concepts, then $C \sqsubseteq D$ is a terminological axiom. $C \equiv D$ is the abbreviation of the two axioms $C \sqsubseteq D$ and $D \sqsubseteq C$. A finite set of terminological axioms is called a terminology or TBox. An *interpretation* $\mathcal{I} = (\Delta, \cdot^{\mathcal{I}})$ consists of a non-empty set Δ and a mapping function $\cdot^{\mathcal{I}}$. The function $\cdot^{\mathcal{I}}$ maps every role to a subset of $\Delta \times \Delta$ and every concept to a subset of Δ. If there exists an interpretation $\mathcal{I}$ which satisfies every axiom in $\mathcal{T}$, i.e., $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ must hold for every $C \sqsubseteq D$ in $\mathcal{T}$, we call the interpretation a *model* of $\mathcal{T}$. Hence $\mathcal{T}$ is called *satisfiable* if such a model exists, and *unsatisfiable* otherwise. $\mathcal{ALC}$ is a syntactic variant of the propositional modal logic $\mathrm{K}_{(\mathrm{m})}$ and can be seen as a fragment of FOL. Axioms in a TBox can be translated into FOL sentences correspondingly [2]. Let us use the function $\mathcal{F}()$ to represent such a translation. Therefore,

$$\mathcal{F}(A \sqsubseteq \exists R.C) = \forall x \exists y : A(x) \rightarrow (R(x,y) \wedge C(y))^{1}$$
$$\mathcal{F}(A \sqsubseteq \forall R.C) = \forall x \forall y : A(x) \rightarrow (R(x,y) \rightarrow C(y))$$
$$\mathcal{F}(A \sqsubseteq B) = \forall x : A(x) \rightarrow B(x) = \forall x : \neg A(x) \vee B(x)$$

In the following sections we shall only focus on the $\mathcal{ALC}$ TBox satisfiability reasoning problem for ease of illustration and evaluation. Solving other reasoning tasks is discussed in Section 6.

1.2 The Contribution of This Paper

In this paper we present a reasoning procedure which systematically and effectively uses a highly optimized DPLL algorithm for DL reasoning. This can be partially compared to SMT (Satisfiability Modulo Theory) based approaches but offers more advantages. In addition, we also propose a learning algorithm called unsat-learning which is proven to be very effective for reasoning optimization. Prior to our work, only approaches for unsat-caching [4,3,20] have been proposed, which can only prevent unsat-nodes (their number could be exponential) to be repeatedly expanded whereas our approach can prevent unsat-nodes to be repeatedly generated, which can reduce the search space exponentially in the best case. Moreover, we also integrate so-called *forgetting* techniques into our DL reasoning algorithm, which improve an algorithm's tractability in practice. At last, besides the standard negation normal form (NNF), a new DL normal form (DLNF) specifically for reasoning optimization is also investigated.

[1] In the following sections, we find it more appropriate to replace y by a skolem function $f(x)$.

2 An Intelligent TBox Reasoning Procedure

2.1 A Reasoning Procedure for DLNF TBoxes

Suppose we are given a TBox $\mathcal{T}$ in which the axioms can be divided into three sets $\mathcal{T}_g$, $\mathcal{T}_{ue}$, and $\mathcal{T}_{ua}$. In $\mathcal{T}_g$, each axiom is in the format $\top \sqsubseteq C$ where C is a disjunction of unary literals. In other words, if we ignore the "$\top \sqsubseteq$" part in $\mathcal{T}_g$ for all axioms, $\mathcal{T}_g$ can be considered in propositional logic conjunctive normal form (CNF). Let us call it PCNF to distinguish it from CNF in FOL that allows n-ary relations rather than only unary ones. In $\mathcal{T}_{ue}$ and $\mathcal{T}_{ua}$, all axioms are in the format such that $A \sqsubseteq \exists R.C$ and $A \sqsubseteq \forall R.C$ respectively, where A is a positive unary literal; R is an atomic role and C is a concept in the format of PCNF. In addition, all positive unary literals on the left-hand side of $\mathcal{T}_{ue}$ and $\mathcal{T}_{ua}$ are unique.

Definition 1. *A TBox $\mathcal{T}$ is in **Description Logic Normal Form (DLNF)** if all axioms in $\mathcal{T}$ can be divided into the three sets $\mathcal{T}_g$, $\mathcal{T}_{ue}$, and $\mathcal{T}_{ua}$ as described above.*

Example 1.

$$\mathcal{T}^{'} = \{\top \sqsubseteq \neg A_0 \sqcup \neg B_0,\ \ \neg A_0 \sqsubseteq B_0, \quad \forall R.(A_0 \sqcap \neg B_0) \sqsubseteq \neg A_0 \sqcup \forall R.\neg B_0\}$$
$$\mathcal{T}^{''} = \{\top \sqsubseteq \neg A_0 \sqcup \neg B_0,\ \ \top \sqsubseteq A_0 \sqcup B_0,\ \top \sqsubseteq \neg A_0 \sqcup A_1 \sqcup A_2,$$
$$A_1 \sqsubseteq \forall R.\neg B_0, \qquad A_2 \sqsubseteq \exists R.(\neg A_0 \sqcup B_0)\}$$

In Example 1, $\mathcal{T}^{'}$ is not in DLNF since the last two axioms do not match the definition of any of the three sets while $\mathcal{T}^{''}$ is in DLNF. If a TBox $\mathcal{T}$ is in DLNF, then it can be easily translated to sets of skolemized FOL sentences (as shown below), where $f(x)$ is a skolemization function which maps instances to instances; x and y are variables of instances of the underlying domain:

$$\begin{aligned}
\mathcal{F}(\mathcal{T}_g) &= \forall x \bigwedge_{i=1}^{m} \bigvee_{j=1}^{n} \alpha_{ij}(x) \\
\mathcal{F}(\mathcal{T}_{ua}) &= \forall x \forall y \bigwedge_{l=1}^{q} \gamma_l(x) \rightarrow (s_l(x,y) \rightarrow d_l(y)) \\
\mathcal{F}(\mathcal{T}_{ue}) &= \forall x \bigwedge_{k=1}^{p} \delta_k(x) \rightarrow (r_k(x, f_k(x)) \wedge c_k(f_k(x)))
\end{aligned} \tag{1}$$

Therefore, in Example 1, we have the FOL translation $\mathcal{F}(\mathcal{T}^{''})$ as follows:

$$\mathcal{F}(\mathcal{T}_g^{''}) = \forall x : (\neg A_0 \vee \neg B_0,\ \ A_0 \vee B_0,\ \ \neg A_0 \vee A_1 \vee A_2)(x)$$
$$\mathcal{F}(\mathcal{T}_{ua}^{''}) = \forall x \forall y : A_1(x) \rightarrow (R(x,y) \rightarrow \neg B_0(y))$$
$$\mathcal{F}(\mathcal{T}_{ue}^{''}) = \forall x : A_2(x) \rightarrow (R(x, f(x)), (\neg A_0 \vee B_0)(f(x)))$$

where $(\neg A_0 \vee \neg B_0)(x)$ is the abbreviation of $(\neg A_0(x) \vee \neg B_0(x))$, and we replace the symbol "$\wedge$" with "," to emphasize the fit of set data structure during implementation.

Let us assume a TBox $\mathcal{T}$ is satisfiable and $\mathcal{I}^m = (\Delta, \cdot^{\mathcal{I}^m})$ is a model of $\mathcal{T}$. Therefore, Δ must be non-empty, and it should at least contain one instance. If the existence of such an instance is impossible, i.e, it is impossible to construct a mapping function $\cdot^{\mathcal{I}^m}$ to satisfy all axioms in $\mathcal{T}$, then $\mathcal{T}$ must be unsatisfiable. Without loss of generality, let us say i_0 is such an instance in Δ. We call the problem space w.r.t. only one single instance a *node*. We also use instance names to identify nodes if this does not cause any confusion. The problem space w.r.t. i_0 is called *root node* or *root*. Let us take $\mathcal{T}''$ from Example 1 to illustrate how we can solve the satisfiability problem for a TBox in DLNF effectively. As we already stated, the underlying idea of solving the satisfiability problem of $\mathcal{T}''$ is to prove that the existence of i_0 is possible.

Step (i) *Construct the root node from $\mathcal{T}$.*

The logical semantics of the root node is that w.r.t. i_0 all axioms in $\mathcal{T}$ map to the logical true under a mapping function. If we can prove that such a mapping function exists, $\mathcal{T}$ is satisfiable. Otherwise, $\mathcal{T}$ is unsatisfiable. Now let us consider the root node of $\mathcal{T}''$ which is the instantiation of x with i_0 in $\mathcal{F}(\mathcal{T}'')$ (for unary relations, we use A instead $A(i_0)$ for simplification purpose). Then, we get the root node of $\mathcal{T}''$:

1) $\{\neg A_0 \vee \neg B_0, A_0 \vee B_0, \neg A_0 \vee A_1 \vee A_2\}$
2) $A_1 \rightarrow \forall y : (R(i_0, y) \rightarrow \neg B_0(y))$
3) $A_2 \rightarrow \{R(i_0, f(i_0)), (\neg A_0 \vee B_0)(f(i_0))\}$

Mapping 3) and 2) to the logical true can be easily achieved if we include $(\neg A_2(i_0) : true)$ and $(\neg A_1(i_0) : true)$ in our mapping function. We call such kind of sentences *rules*. To differentiate the two types of rules, we call the sentences instantiated from $\mathcal{F}(\mathcal{T}_{ue})$ $_{\exists}$*-rules* and the ones from $\mathcal{F}(\mathcal{T}_{ua})$ $_{\forall}$*-rules*. As for 1), finding a mapping to satisfy all sentences in it is equivalent to finding a propositional model for the corresponding CNF. If there does not exist such a model, it means the mapping function $\cdot^{\mathcal{I}^m}$ can not be constructed, hence $\mathcal{T}''$ must be unsatisfiable. Let us assume there exists a propositional model for the CNF of the root node. We call the propositional model of the CNF inside a node a *path* of the node. We call an individual element in the path such as $\neg A_2(i_0)$, $A_1(i_1)$ an *item.*

Step (ii) *Find a path of the corresponding node.*

Provided that the path of the root node does exist, let us consider a specific item in the path. We have only three possibilities:

(a) The item matches to an item on the left-hand side of a $_{\forall}$-rule.
(b) The item matches to an item on the left-hand side of an $_{\exists}$-rule.
(c) The item does not match to any item on the left-hand side of any rule.

As for (c), the item occurring in the path has no impact on the sentences instantiated from $\mathcal{F}(\mathcal{T}_{ua})$ and $\mathcal{F}(\mathcal{T}_{ue})$. If all items in the path fall into this category, satisfiability of the underlying TBox is directly proven. For instance, in the above example, $\{\neg A_0, B_0, \neg A_1, \neg A_2\}$ is a path of the root node. However, none of the elements in it matches the elements on the left-hand side of 3) and 2). Therefore, $\mathcal{T}''$ is satisfiable, and one can easily construct a model with a complete mapping function for it. As for the other options (a) and (b), rule expansions are required.

Step (iii) *Perform rule expansions.*

If a $_\forall$-rule is triggered, the right-hand side of the rule needs to be added to the corresponding node, i.e., adding the sentence $\forall y : s_l(i, y) \to d_l(y)$ to node i. Similarly, if an $_\exists$-rule is triggered, a relation $r_k(i, f_k(i))$ needs to be added to $\cdot^{\mathcal{I}^m}$. Without loss of generality, let us use a new instance name i_1, which does not exist in $\varDelta$, to replace $f_k(i)$ instead of keeping the function name.[2] Therefore, we add a new instance i_1 to $\varDelta$ and a binary relation $r_k(i, i_1)$ to $\cdot^{\mathcal{I}^m}$. Thus a new node w.r.t. i_1 is added to the search space and c_k is added to the new node as part of the rule.

A node which generates new nodes is called a *predecessor*, and the generated nodes its *successors*. The corresponding role r_k is called an *edge*. If a PCNF c_k is added to a node because a $_\exists$-rule is triggered by some δ_k, then δ_k is called an $_\exists$-prefix and c_k is called an $_\exists$-label of the node. Similarly, if a d_l is added to a node due to a triggered $_\forall$-rule by a γ_l, then γ_l is called a $_\forall$-prefix and d_l a $_\forall$-label of the node. The set of prefixes of a node is called a *prefix set* and the set of labels a *label set* of the node. We call two nodes *equivalent* if they contain the same label set. If there is no conflict detected in any of its successor nodes, then the node is called *satisfiable*.

To illustrate how it works, let us still use $\mathcal{T}''$ from Example 1. Let us assume that the path we found for the root node is $\{\neg A_0, B_0, A_1, A_2\}$ this time.[3] A_1 triggers a $_\forall$-rule, therefore we add a special rule $\forall y : R(i_0, y) \to \neg B_0(y)$ to the root node. A_2 triggers an $_\exists$-rule. Therefore, $\varDelta = \varDelta \cup \{i_1\}$ where i_1 is a new name and $\cdot^{\mathcal{I}^m} = \cdot^{\mathcal{I}^m} \cup \{R(i_0, i_1)\}$. Furthermore, we also create a new node w.r.t. i_1. $\neg A_0 \vee B_0$ is added to the newly created node as part of the $_\exists$-rule. Since we have $R(i_0, i_1)$ in $\cdot^{\mathcal{I}^m}$, the rule $\forall y : R(i_0, y) \to \neg B_0(y)$ in the node i_0 is triggered in which we instantiate y with i_1. Therefore, we have the following node i_1 which contains all instantiated sentences (rules) of $\mathcal{F}(\mathcal{T}'')$ (due to the $\forall x$ restriction) by replacing x with i_1 together with $\neg B_0$ ($_\forall$-label) and $\neg A_0 \vee B_0$ ($_\exists$-label). We separate the label set from the instantiated PCNF below just for illustration purposes.

1) $\{\neg A_0 \vee \neg B_0, A_0 \vee B_0, \neg A_0 \vee A_1 \vee A_2\}, \{\neg \mathbf{B_0}, \neg \mathbf{A_0} \vee \mathbf{B_0}\}$
2) $A_1 \to \forall y : (R(i_1, y) \to \neg B_0(y))$
3) $A_2 \to \{R(i_1, f(i_1)), (\neg A_0 \vee B_0)(f(i_1))\}$

It is obvious that there does not exist a proposition model for the PCNF in 1). It means that the prefix set $\{A_1, A_2\}$ of node i_1 (which is part of the path of node i_0) leads to a contradiction. In such kind of situation, the path found in the predecessor node needs to be rolled back and recalculated. This rollback and recalculation procedure is applied recursively until no valid path can be found for the root node (the corresponding TBox is unsatisfiable) or all paths have been found for all nodes in the problem space (the corresponding TBox is satisfiable).

Step (iv) *Apply steps (ii) to (iii) recursively until either all paths have been found for all nodes or no path could be found for the root node.*

Proposition 1. *The reasoning procedure from step (i) to step (iv) is sound and complete.*

[2] Refer to [2] for details on the open world assumption in DLs.

[3] We chose a complete propositional model for illustration purposes here. A partial model containing only $\{\neg A_0, B_0\}$ is already sufficient for a satisfiability proof.

Proof. The proposition holds because a TBox $\mathcal{T}$ can straightforwardly be translated to FOL to which Herbrand's theorem applies. The above-mentioned procedure is exactly a procedure for constructing a herbrand model [10].

2.2 Integrating Learning into Reasoning

The underlying idea of learning w.r.t. logic reasoning is to prune unvisited search space based on the knowledge achieved from previous search steps. With the pruned search space, reasoning algorithms are supposed to find search results faster. By considering the size of the underlying search space regarding to $\mathcal{ALC}$, which can be exponential w.r.t. the size of input ontologies [2], it is almost certain that effective learning should improve the average reasoning performance significantly.

As discussed in Section 2.1, finding a path for a node is reduced to finding a propositional model for the underlying PCNF of the node. Therefore, some proven to be very effective optimization algorithms such as DPLL equipped with conflict-driven learning and back-jumping that are also employed by state-of-the-art SAT solvers [22,15] can be directly used. We call such kind of learning inside a single node *local learning*. The discussion and improvements of *local learning* are beyond the scope of this paper and we shall focus on *global learning*, i.e., the kind of learning that affects the reasoning search space on the pre-model level, which directly affects the number of nodes to be searched. To be more specific, global learning can be categorized into three types:

1. Unsat-learning: If the status of a node has already been determined as unsatisfiable, the algorithm should learn from it and block all related unexplored search space that would definitely lead to a failed search result.
2. Sat-learning: If the status of a node has already been determined as satisfiable, the algorithm should directly mark the status of its equivalent nodes as satisfiable without performing reasoning or expansion on them.
3. Unknown-learning: When the algorithm starts, the status of visited nodes are first marked as "unknown" meaning that the sat/unsat status has not yet been determined. As the model graph is expanded during reasoning, if a newly created node is equivalent to a node already marked as "unknown", then we should avoid duplicate reasoning on the latter. In this case, we mark the latter node as "blocked" meaning its satisfiability should refer to another node. The previously visited node with an unknown status is called a *blocker*, and the blocked node is called a *blockee*.

Let us first check how an algorithm can learn from an unsat node. Without loss of generality, let us suppose the label set of an unsat node is $\{c_m, d_1, d_2, \dots, d_n\}$.[4] Correspondingly, the prefix set is $\{\delta_m, \gamma_1, \gamma_2, \dots, \gamma_n\}$. If the node is marked as unsat, it means that the combination of all its prefixes leads to a conflict w.r.t. $\mathcal{T}$. That is

$$\mathcal{T} \models \exists x : \delta_m(x) \wedge \gamma_1(x) \wedge \gamma_2(x) \wedge \dots \wedge \gamma_n(x) \rightarrow \bot \quad (2)$$

It is equivalent to:

$$\mathcal{T} \models \forall x : \neg\delta_m(x) \vee \neg\gamma_1(x) \vee \neg\gamma_2(x) \vee \dots \vee \neg\gamma_n(x) \quad (3)$$

[4] For $\mathcal{ALC}$, a non-root node contains exactly one existential label.

We call the right-hand side of axiom (3) a *learned sentence*. Let us add the learned sentence to $\mathcal{F}(\mathcal{T}_g)$ and populate it to all nodes with a still unknown status. We can easily prove that all nodes whose prefix set contains the set $\{\delta_m, \gamma_1, \gamma_2, \ldots, \gamma_n\}$ are pruned from the search space. To illustrate how unsat-learning works, let us again consider the example we used in Section 2.1. Node i_1 is unsat and its prefix set is $\{A_1, A_2\}$. Then the learned sentence is $\forall x : (\neg A_1 \vee \neg A_2)(x)$. First we rollback the path found for the root node and the corresponding elements related to that path which were added/created during reasoning. The second step is to add the learned sentence to $\mathcal{T}$ and populate it to the nodes with an unknown status. In our example, after the second step we get:

$$\mathcal{F}(\mathcal{T}_g^{''}) = \forall x : \{\neg A_0 \vee \neg B_0, A_0 \vee B_0, \neg A_0 \vee A_1 \vee A_2, \neg \mathbf{A_1} \vee \neg \mathbf{A_2}\}(x)$$

The changed root node:

1) $\{\neg A_0 \vee \neg B_0, A_0 \vee B_0, \neg A_0 \vee A_1 \vee A_2, \neg \mathbf{A_1} \vee \neg \mathbf{A_2}\}$
2) $A_1 \rightarrow \forall y : (R(i_0, y) \rightarrow \neg B_0(y))$
3) $A_2 \rightarrow \{R(i_0, f(i_0)), (\neg A_0 \vee B_0)(f(i_0))\}$

Now if we recalculate the path of the root node or any other node in our problem space in future search, any supersets containing $\{A_1, A_2\}$ will be automatically excluded. Unsat-learning does not affect the soundness or completeness of the algorithm and the proof is trivial.

As for sat-learning and unknown-learning, these techniques are also called sat-caching and blocking in other papers [4,3]. They can be simply implemented as buffering, i.e., all nodes in the underlying search space are identified by their labels, and if a newly created node has a buffer hit, it can be directly marked either as ‘sat’ or ‘blocked’ without further expansion.

However, naive buffering may cause the algorithm to become unsound [8]. Solutions to fix the unsoundness are discussed in [3,6]. The solution introduced in [6] is widely considered to be the best so far. However, it requires EXPSpace in the worst case to construct a pre-model which can easily cause the reasoning algorithm to become intractable.

Considering the procedure we discussed so far, unknown-learning is not a source of unsoundness since the status of both blockers and blockees is restricted only to “unknown” whereas sat-learning may cause unsoundness due to the problematic definition of “satisfiable” in Section 2.1. For example, in the case where all successors are “blocked” by some other “unknown” nodes, the reasoning algorithm normally marks the predecessor as “sat” due to *no conflict detected in any of its successor nodes*. In our case, such a node will be saved in the sat-buffer and might be reused by others afterwards due to a buffer hit. This is the source for the unsoundness w.r.t. sat-learning since any of the related blockers could be proven as “unsat” afterwards. Therefore, in DL we characterize node satisfiability as *relative* to blockers compared to *absolute* node unsatisfiability. In fact, there is a simple solution to ensure soundness. All we need to do is to remove the nodes from the sat-buffer which directly or indirectly depend on a blocker node whenever such a node is detected as “unsat”.

Algorithm 1. Normalization to CNF

```
1: function NORMALIZE(axiom)
2:     remove $\equiv$ and non-top concept from lefthand side of $\sqsubseteq$ from axiom
3:     convert axiom to NNF
4:     if axiom matches $\top \sqsubseteq C \sqcup (D \sqcap E)$ then
5:         normalize($\top \sqsubseteq \neg\eta \sqcup D$)          ▷ $\eta$ is a new name
6:         normalize($\top \sqsubseteq \neg\eta \sqcup E$)
7:         normalize($\top \sqsubseteq C \sqcup \eta$)
```

Algorithm 2. Remove value restrictions

```
function REMOVEROLEITEM(aDLCNFClause)
    for all concept in aDLCNFClause do
        if concept matches $\exists R.C$ then
            replace concept with $\delta$          ▷ $\delta$ is a new name
            $\mathcal{T}_{ue}$.add($\delta \sqsubseteq$ concept)
        else if concept matches $\forall R.D$ then
            replace concept with $\gamma$          ▷ $\gamma$ is a new name
            $\mathcal{T}_{ua}$.add($\gamma \sqsubseteq$ concept)
```

3 Intelligent Tableau Algorithm

3.1 Normalization

Reasoning algorithms used by state-of-the-art DL reasoners usually require axioms of the input TBox to be transformed into NNF which can be done easily in linear time. However, in the above mentioned reasoning procedure we require the input TBox to be in DLNF. Therefore, before performing reasoning on a TBox, we need an algorithm to convert an arbitrary TBox into the format of DLNF which is called *normalization*.

We divide the normalization into two steps: The first step is to remove all conjunctions from all axioms in the target TBox (see Algorithm 1). In the second step, we remove all concepts with value restrictions from the resulting CNF and then add corresponding axioms to $\mathcal{T}_{ua}$ and $\mathcal{T}_{ue}$ (see Algorithm 2). Concepts C and D in Algorithm 2 can also be reduced to PCNF easily in a similar way. One can easily tell that in Example 1, $\mathcal{T}^{''}$ is the normalization result of $\mathcal{T}^{'}$.

Based on Algorithms 1 and 2, we have Proposition 2.

Proposition 2. *There exists an algorithm that converts an arbitrary TBox $\mathcal{T}$ to $\mathcal{T}'$ in polynomial time where $\mathcal{T}'$ is in DLNF and $\mathcal{T}'$ is equisatisfiable to $\mathcal{T}$.*

Proof. It is obvious that these algorithms require polynomial time. We only need to prove that $\mathcal{T}'$ is equisatisfiable to $\mathcal{T}$ after normalization. The conversion from line 4 to line 7 in Algorithm 1 is widely used in converting an arbitrary SAT problem into a 3-SAT problem, and the equisatisfiability proof needs not to be repeated here. When converting to DLNF, the only difference is the introduction of $_\exists$-rules and $_\forall$-rules. In fact, the equisatisfiability of such a conversion can be proven in exactly the same way. As for the PCNF conversion of role fillers, the proof can be easily done based on the fact that equisatisfiability is closed under conjunction.

3.2 Intelligent Reasoning Algorithm

As we already mentioned in previous sections, the algorithm for finding a path inside a specific node is implemented as finding a propositional model w.r.t. a CNF. A DPLL procedure with local learning is described in Algorithm 3.

Algorithm 3. findModel

```
 1: function FINDMODEL(aPCNF)
 2:     while $true$ do
 3:         while ¬unsat and ¬finish do
 4:             unfold()
 5:             propagateAndDeduce()
 6:         if unsat then
 7:             if currentLevel = 0 then
 8:                 return $false$
 9:             else
10:                 resolveConflict()
11:         else if ¬ decideNextBranch() then
12:             return $true$
```

Algorithm 4. $\forall$- and $\exists$-unfold

```
function ∀-UNFOLD(aPositiveUnaryLiteral)
    rule ← createLocalRule(aPositiveUnaryLiteral)
    for all successor ∈ successorList do
        if successor.role = rule.role then
            successor.addPrefix(rule.prefix)
            successor.cnf.add(rule.filler)
function ∃-UNFOLD(aPositiveUnaryLiteral)
    node ← createNewNode(aPositiveUnaryLiteral)
    for all rule ∈ getLocalRule(node.role) do
        node.cnf.add(rule.filler)
        node.prefix.add(rule.prefix)
    successorList.add(node)
```

Compared to SAT reasoning, we have to consider the $\exists$-rule and $\forall$-rules in DL reasoning. The function unfold() for rule expansions is called at line 4 in Algorithm 3. It is executed whenever a propositional model item has been added. For other functions, the details are similar to what is described in [22] except that the function resolveConflict() needs to additionally deal with unsat caused by global learning and the rollback() needs to do the opposite of unfold() if the rolling back item is unfoldable. $\exists$-unfold and $\forall$-unfold are described in Algorithm 4.

A recursive depth-first search (DFS) algorithm to determine satisfiability of an input TBox is shown in Algorithm 5. At line 3, the algorithm checks and updates the local CNF from the results of global learning, if applicable. This can avoid a global propagation when global learning results are applicable that may affect system performance at runtime. At line 15 we ensure the soundness of sat-learning, if applicable. Line 16 can be as simple as adding a disjunction of negated prefixes to $\mathcal{T}_g$ as described in Section 2.2.

Algorithm 5. satCheck

```
 1: external satBuffer, unknownBuffer
 2: function SATCHECK(aNode)
 3:     updateCNFFromGlobalLearning()
 4:     if ¬ findModel(aNode.pcnf) then
 5:         return $false$
 6:     for all successor ∈ successorList do
 7:         if successor ∈ satBuffer then
 8:             successor.status ← SAT
 9:         else if successor ∈ unknownBuffer then
10:             successor.status ← BLOCKED
11:         else
12:             unknownBuffer.add(successor)
13:             if ¬ satCheck(successor) then
14:                 unknownBuffer.remove(successor)
15:                 ensureSATLearningSoundness()
16:                 Learn from successor.prefix
17:                 return satCheck(aNode)
18:     if current ≠root then
19:         unknownBuffer.remove(current)
20:         satBuffer.add(current)
21:     return $true$
```

3.3 Forgetting

In Algorithm 5, a pre-model is constructed through DFS. Therefore, during reasoning we only need to keep one single branch in memory to construct the pre-model. Such kind of algorithms can be implemented in PSPACE as further studied and proved in [16]. As a result, many tableau-based decision procedures for DL reasoning can be considered as overall “practically tractable”. With the presence of global buffers, worst case optimal algorithms using DFS are also studied and presented in [4] in which the analysis of “practically tractable” algorithms by using global buffers focus only on the space and time required for the construction of the pre-model whereas the space used by the global buffers is ignored. As a matter of fact, we can easily prove that, if no proper action is taken, the size of global buffers, even though only unsat-caching is involved, can be exponential with regard to the size of the input TBox. Therefore, the “practically tractable” feature might no longer hold in the presence of global buffers.

By considering the algorithm we proposed in this work, without special treatment, the size of learning buffers for both global and local learning can also be EXP size in the worst case. To achieve tractability, an intuitive solution is to remove less useful learned knowledge from learning buffers, which can be seen as an opposite operation to learning, and it is normally called *forgetting*. In our work, forgetting is applicable to local learning, unsat-learning, and sat-learning but not to unknown-learning. This scheme is good enough for practical reasoning since through DFS the number of nodes with an unknown status is normally small. A forgetting algorithm could be as simple as using a size-restricted FIFO queue, and it could be as complicated as some advanced heuristic algorithms.

4 Related Work

We named the algorithm developed in this work as "intelligent tableau" to emphasize its relationship to the traditional tableau algorithms [2] in that both algorithms construct tree-like pre-models for $\mathcal{ALC}$ reasoning. One can easily prove that both are variants for finding a herbrand model to tackle DL reasoning problems. Learning and forgetting can be considered as optimization techniques, which could also possibly be integrated into traditional DL tableau algorithms. The major difference to our work is on how to deal with disjunctions, i.e., a tightly integrated DPLL vs. standard tableau branching.

Traditional tableau algorithms are implemented by almost all state-of-the-art reasoners such as FaCT++ [21], Pellet [19], RacerPro [9] and HermiT [18]. These algorithms are widely blamed for a low efficiency in the presence of many general inclusion axioms or disjunctions [12]. Even though equipped with many optimization techniques such as boolean constraint propagation (BCP), semantic branching, back-jumping, etc., most DL reasoners still easily become intractable when dealing with ontologies containing many disjunctions. Even though the JNH test cases we used in Section 5 are considered as trivial examples for a SAT solver, no state-of-the-art DL reasoner is able to provide an efficient solution. As for more complicated CNF test cases, these reasoners easily become intractable based on our test results. Modified versions of tableau algorithms such as hypertableaux [14], which uses hyper-resolution instead of simple tableau branching, are developed and applied in reasoners such as HermiT. Even though the performance in dealing with disjunctions is improved, based on our test results, HermiT is normally performing worse than others in situations where a big amount of nodes needs to be constructed in the underlying pre-model (see Section 5).

Researchers also presented approaches for DL reasoning through SMT [17], which make it possible to solve DL reasoning problems by using efficient state-of-the-art SAT solvers. Some of the underlying ideas coincidentally overlap with our work. However, even though the SMT solutions have achieved a similar performance for some benchmark test cases compared to state-of-the-art DL reasoners, they did not provide effective ways to prune the underlying search space. In addition, the encoding algorithms used to reduce DL problems to SAT problems are still as hard as EXPTime which may cause some significant overhead when considering performance in real-world applications. Moreover, the black-box consideration of the SAT portion might cause unnecessary search if a conflict could be easily detected before a full SAT model is constructed. So far, we were unable to include a practical SMT reasoner in our comparison tests.

The research on EXPTime Tableau for $\mathcal{ALC}$ [4] (by applying global caching) has been further developed and implemented by [6,7]. These kinds of algorithms either heavily use subset checking or require EXPSpace to construct the pre-model. Both cases can easily cause intractability in real world applications. Implementations of such kind of reasoning algorithms are still far from building a practical reasoner for real world DL applications.

5 Empirical Results

The primary goal of the algorithm developed in Section 3.2 is to conduct "fast" reasoning — the purpose of DPLL based algorithm is to improve the reasoning performance w.r.t. to a single node while comprehensive learning is to reduce the number of nodes to be searched. A good way to verify whether our goal has been achieved is through running typical benchmark test cases. In addition, designing an enable/disable switch on some specific optimization feature is the best way to verify its effectiveness. Based on such motivation, we provide a reference implementation called LIGHT in which sat-learning and unsat-learning can be switched on and off. We consider the features of our reasoning procedure such as being DPLL-based, employing DLNF ontology normalization and unknown-learning as so fundamental that they are tightly integrated into our architecture and therefore can not be disabled. We conducted our benchmark tests using four different settings of LIGHT (see Table 1): (i) both sat-learning and unsat-learning switched off (L-N); (ii) only sat-learning switched on (L-S); (iii) only unsat-learning switched on (L-U); (iv) both sat-learning and unsat-learning switched on (LIGHT). Part of the test results for the employed $\mathcal{ALC}$ benchmark test cases are shown in Table 1. The LIGHT reasoner for different platforms together with all test cases we used, complete test results and test scripts are available for download.[5]

All test results in Table 1 are based on a Ubuntu Linux 12.04 32 bit platform. The used hardware is a DELL Precison 390 with Intel Core 2 Duo processor 2.4G equipped with 4GB memory. For Java based reasoners, we used Oracle JDK v7.0.11. In Table 1, all runtimes are given in seconds. The word "cr" means the system crashed (out of memory or segment fault) during the test, and "to" means the system was aborted after a timeout ($\geq$ 2000 seconds). The suffix "s" and "u" of ontology names represents the corresponding TBox that is either satisfiable or unsatisfiable.

The JNH [11] test cases are CNF benchmarks converted to OWL syntax and are used to test the capability of DL reasoners for dealing with ontologies containing many (global) disjunctions. BCS (Basic Call System) [1] test cases are real-world examples and typical in the sense that large amount of nodes are required to construct a pre-model. GALEN test cases are used to evaluate the reasoners when dealing with simple problems. The test cases named "k_XX" are taken from Tableaux'98 [13].

As shown in Table 1, in some situations where very limited number of branches is required to build a pre-model or conflicts can be easily detected, sat and unsat learning have no significant impact on the results for GALEN and JNH. In these situations, the effectiveness of LIGHT's reasoning compared to other reasoners can be primarily attributed to the optimized DPLL algorithm used. Learning may also have negative impact in some situations such as K_PH_14P. In some cases, with only unsat learning enabled we can achieve better results than using the combination of the two, i.e., sat-learning only causes overhead for a test case such as K_PATH_20P. In many situations, unsat-learning is critical for obtaining a good performance. However, in the BCS test

[5] `http://www.lightreasoner.co.nf/`

Table 1. Benchmark results for $\mathcal{ALC}$ test cases (runtimes in seconds)

	L-N	L-S	L-U	LIGHT	HermiT	Pellet	Fact++	Racer
galen1s	0.12	0.12	0.14	0.12	1.2	1.3	0.44	1.7
galen2s	0.16	0.15	0.16	0.15	1.3	1.4	0.46	1.9
JNH15u	0.02	0.02	0.02	0.02	6.2	119.1	94.7	119.5
JNH16u	0.07	0.06	0.06	0.07	237.4	452.3	cr	15384
JNH17u	0.02	0.02	0.02	0.02	1.6	21.1	9.5	1165
k_d4_12nu	to	to	44.32	44.47	to	to	1054	to
k_d4_13nu	to	to	99.77	98.70	to	to	to	to
k_dum_18nu	18.04	15.05	17.33	13.99	to	to	cr	196.29
k_dum_19nu	to	to	37.59	32.27	to	to	cr	140.88
k_ph_14pu	963.7	1001	1005	1014	cr	cr	cr	to
k_tp4_21nu	15.84	15.31	5.32	0.32	to	0.54	cr	to
k_branch_20nu	0.34	0.34	0.35	0.35	to	2.3	14.7	16.1
k_branch_21nu	0.39	0.40	0.40	0.39	to	2.4	18.2	19.2
k_path_20pu	1.5	1.53	0.19	1.53	cr	21.03	5.85	7.88
k_path_21pu	1.7	1.76	0.23	1.78	cr	25.63	7.30	9.0
k_poly_15pu	18.36	18.3	18.0	0.43	179.27	27.67	34.97	1.62
k_poly_16pu	cr	cr	cr	0.61	373.4	76.98	cr	2.03
k_poly_20ns	cr	cr	to	236.7	cr	cr	cr	149.9
k_poly_21ns	cr	cr	to	325.4	cr	cr	cr	524.6
BCS3s	to	0.03	to	0.02	1.6	20.7	cr	0.69
BCS4s	to	1.37	to	0.20	133.8	to	cr	13.8
BCS5s	to	to	to	2.14	cr	to	cr	276.2

cases, we also see that sat-learning plays a critical role to ensure effective reasoning. Overall, the combination of both sat and unsat learning achieves very good results in most of the cases.

Compared to other DL reasoners, LIGHT is up to one order of magnitude faster for the GALEN test cases. For the BCS benchmarks, LIGHT is two orders of magnitude faster than Racer, which is the only reasoner besides LIGHT that can process all three variants. The overall performance of LIGHT is significantly improved for the benchmarks selected from Tableaux'98 (the test cases with prefix "k_"). The JNH benchmark results demonstrate the effectiveness of LIGHT by using an optimized DPLL algorithm in dealing with situations where one has only one node in the pre-model that has many disjunctions while the other reasoners are several orders of magnitude slower than LIGHT.

6 Discussion

The TBox satisfiability problem we discussed in this work can be seen as a special case of the ($\top$) concept satisfiability problem w.r.t. a non-empty TBox. From this perspective, once an algorithm solves the TBox satisfiability problem, all other DL reasoning tasks such as concept satisfiability, classification, concept subsumption, ABox satisfiability etc. can also be solved easily by using exactly the same algorithm [2].

Some may consider the normalization algorithm we presented in this work that introduces additional variables to be a source of inefficiency. After all, the worst case complexity analysis even for DPLL based algorithms is tightly related to the number of variables involved. As a matter of fact, this kind of concern is unjustified. First of all, the normalization algorithm requires only polynomial (linear) time which is normally trivial compared to the EXPtime reasoning algorithm. Another fact is that no proof or test results indicate that the introduction of variables can significantly affect the reasoning performance. Our test results have shown that the introduction of new variables such as the conversion from 5CNF to 3CNF in SAT reasoning in most of the cases interestingly improved the reasoning performance.

At last, one may wonder the necessity of the algorithm we proposed in this work. After all, the algorithm proposed in this work can be easily reduced to finding a Herbrand model in FOL which is also the case for traditional tableau algorithms. In fact, one can easily reduce the algorithm proposed in this work to traditional tableau-based algorithms for further analysis such as computational complexity and termination analysis. From our perspective, the major benefits of the algorithm proposed in this work are based on two points. First, this algorithm helps us reduce a DL-based problem to a SAT based problem so that we can delegate efficient reasoning by using proven to be efficient algorithms. The other reason is that we simplified the pre-model structure from an AND-OR graph [6] to an AND-only graph, i.e., all nodes in the discourse have to be satisfiable to make the corresponding TBox satisfiable. The "OR" portion in the graph with its reasoning algorithm is completely merged to the "AND" node. Thus, with the simplified model structure, it is easier to develop and integrate more efficient optimization algorithms such as comprehensive learning.

7 Conclusion and Future Work

In this paper we presented an efficient reasoning algorithm that incorporates learning for solving the TBox satisfiability problem. It is based on searching herbrand models, which is related to but also different from traditional DL tableau algorithms. Preliminary test results have shown that our presented algorithms are significantly more efficient than other existing ones. Besides a systematical discussion of learning on DL reasoning, our DLNF normalization form has been systematically presented and investigated, which makes it easier to incorporate effective optimization techniques into automated reasoning algorithms due to the structured format. Even though the discussion in this work is restricted to the DL $\mathcal{ALC}$, our conjecture is that the algorithm can be applied to super-logics of $\mathcal{ALC}$ or even other DL related logics with slight modifications which will be presented in our future work.

References

1. Areces, C., Bouma, W., de Rijke, M.: Description logics and feature interaction. In: Proceedings of the International Workshop on Description Logics (DL 1999), Linköping, Sweden, pp. 28–32 (1999)

2. Baader, F., Calvanese, D., McGuinness, D., Nardi, D., Patel-Schneider, P.F. (eds.): The Description Logic Handbook, 2nd edn. Cambridge University Press (2007)
3. Ding, Y., Haarslev, V.: Tableau caching for description Logics with inverse and transitive roles. In: Proceedings of the 2006 International Workshop on Description Logics (DL 2006), pp. 143–149 (2006)
4. Donini, F.M., Massacci, F.: EXPTIME tableaux for $\mathcal{ALC}$. Artificial Intelligence 124(1), 87–138 (2000)
5. Faddoul, J.: Reasoning algebraically with description logics. PhD thesis, Department of Computer Science and Engineering, Concordia University (2011)
6. Goré, R., Nguyen, L.: Exptime tableaux for $\mathcal{ALC}$ using sound global caching. Journal of Automated Reasoning, 1–27 (2011)
7. Goré, R., Postniece, L.: An experimental evaluation of global caching for $\mathcal{ALC}$ (system description). In: Armando, A., Baumgartner, P., Dowek, G. (eds.) IJCAR 2008. LNCS (LNAI), vol. 5195, pp. 299–305. Springer, Heidelberg (2008)
8. Haarslev, V., Möller, R.: Consistency testing: The RACE experience. In: Dyckhoff, R. (ed.) TABLEAUX 2000. LNCS, vol. 1847, pp. 57–61. Springer, Heidelberg (2000)
9. Haarslev, V., Möller, R.: Racer system description. In: Goré, R.P., Leitsch, A., Nipkow, T. (eds.) IJCAR 2001. LNCS (LNAI), vol. 2083, pp. 701–705. Springer, Heidelberg (2001)
10. Herbrand, J.: Recherches sur la théorie de la démonstration. PhD thesis, University of Paris (1930)
11. Hooker, J.: Satlib - benchmark problems. Website (2011), `http://www.cs.ubc.ca/~hoos/SATLIB/benchm.html`
12. Horrocks, I.: Using an expressive description logic: FaCT or fiction? In: Proc. of KR 1998, pp. 636–647 (1998)
13. Horrocks, I., Patel-Schneider, P.: DL systems comparison. In: Proc. of the 1998 Description Logic Workshop (DL 1998). CEUR, vol. 11, pp. 55–57 (1998)
14. Motik, B., Shearer, R., Horrocks, I.: Hypertableau reasoning for description logics. Journal of Artificial Intelligence Research 36, 165–228 (2009)
15. Ryan, L.O.: Efficient algorithms for clause learning SAT solvers. Master's thesis, Simon Fraser University, BC, Canada (2004)
16. Schmidt-Schauß, M., Smolka, G.: Attributive concept descriptions with complements. Artificial Intelligence 48(1), 1–26 (1991)
17. Sebastiani, R., Vescovi, M.: Automated reasoning in modal and description logics via SAT encoding: the case study of K(m)/ALC-Satisfiability. Journal of Artificial Intelligence Research 35 (2009)
18. Shearer, R., Motik, B., Horrocks, I.: Hermit: A highly efficient OWL reasoner. In: 5th OWL Experiences and Directions Workshop (2008)
19. Sirin, E., Parsia, B., Cuenca Grau, B., Kalyanpur, A., Katz, Y.: Pellet: A practical OWL-DL reasoner. Journal of Web Semantics 5(2), 51–53 (2007)
20. Steigmiller, A., Liebig, T., Glimm, B.: Extended caching, backjumping and merging for expressive description logics. In: Gramlich, B., Miller, D., Sattler, U. (eds.) IJCAR 2012. LNCS, vol. 7364, pp. 514–529. Springer, Heidelberg (2012)
21. Tsarkov, D., Horrocks, I.: FaCT++ description logic reasoner: System description. In: Furbach, U., Shankar, N. (eds.) IJCAR 2006. LNCS (LNAI), vol. 4130, pp. 292–297. Springer, Heidelberg (2006)
22. Zhang, L.: Searching for truth: techniques for satisfiability of boolean formulas. PhD thesis, Departement of Electrical Engineering, Princeton University (2003)

Author Index