# PAAR-2012

# Third Workshop on Practical Aspects of Automated Reasoning

June 30 & July 1, 2012

Affiliated with the 6th International Joint Conference on
Automated Reasoning (IJCAR 2012)
Manchester, United Kingdom

http://www.eprover.org/EVENTS/PAAR-2012.html

## Preface

This volume contains the papers presented at the Third Workshop on Practical Aspects of Automated Reasoning (PAAR-2012). The workshop was held on June 30 and July 1, 2012, in Manchester, UK, in association with the Sixth International Joint Conference on Automated Reasoning (IJCAR-2012), as part of the Alan Turing Year 2012, held just after The Alan Turing Centenary Conference.

PAAR provides a forum for developers of automated reasoning tools to discuss and compare different implementation techniques, and for users to discuss and communicate their applications and requirements. The workshop brought together different groups to concentrate on practical aspects of the implementation and application of automated reasoning tools. It allowed researchers to present their work in progress, and to discuss new implementation techniques and applications. Papers were solicited on topics that include all practical aspects of automated reasoning. More specifically, some suggested topics were:

- automated reasoning in propositional, first-order, higher-order and non-classical logics;
- implementation of provers (SAT, SMT, resolution, tableau, instantiation-based, rewriting, logical frameworks, etc);
- automated reasoning tools for all kinds of practical problems and applications;
- pragmatics of automated reasoning within proof assistants;
- practical experiences, usability aspects, feasibility studies;
- evaluation of implementation techniques and automated reasoning tools;
- performance aspects, benchmarking approaches;
- non-standard approaches to automated reasoning, non-standard forms of automated reasoning, new applications;
- implementation techniques, optimization techniques, strategies and heuristics, fairness;
- support tools for prover development;
- system descriptions and demos.

We were particularly interested in contributions that help the community to understand how to build useful and powerful reasoning systems in practice, and how to apply existing systems to real problems.

The workshop this year was particularly successful. We received seventeen submissions, for a workshop whose duration was initially one day. Each submission was reviewed by three program committee members. Due to the quality of the submissions, and to prevent the workshop to be intolerably selective, we decided to extend the duration of the event to two days. Also, the AREIS Workshop on Automated Reasoning for Enterprise Information Systems joined PAAR as a special session. In the end and altogether, nineteen papers were submitted, fifteen of which were accepted for presentation. The program included two invited talks:

- *Practical Aspects of SAT Solving* by Armin Biere,
- *Building an Efficient OWL 2 DL Reasoner* by Boris Motik.

Besides a session on the topic of AREIS, PAAR shared a session with the PxTP Workshop on Proof eXchange for Theorem Proving, and included a joint invited talk with the SMT Workshop on Satisfiability Modulo Theories.

The workshop organizers would like to thank the following people for helping to make PAAR a success.

- the authors and participants of the workshop;
- the invited speakers;
- the program committee and the reviewers for their effort;
- Peter Baumgartner and Silvio Ranise, the organizers of the AREIS Workshop, for proposing to group PAAR and AREIS;
- the organizers of the PxTP Workshop, for including the PAAR session on proofs in the PxTP program and making it a joint PAAR-PxTP session;
- the organizers of the SMT workshop, for the organization of the joint SMT-PAAR invited talk.

We are very grateful to the IJCAR organizers for their support and for hosting the workshop, and are indebted to the EasyChair team for the availability of the EasyChair Conference System.

June 2012 Pascal Fontaine, Renate Schmidt, Stephan Schulz

## Program Committee

Clark Barrett (New York University)
Peter Baumgartner (NICTA)
Christoph Benzmüller (Freie Universität Berlin)
Jasmin Christian Blanchette (Technische Universität München)
Chad Brown (Universität des Saarlandes)
Koen Claessen (Chalmers University)
Pascal Fontaine (INRIA and University of Nancy), Co-Chair
Martin Giese (University of Oslo)
Alberto Griggio (Fondazione Bruno Kessler)
John Harrison (Intel)
Yevgeny Kazakov (Universität Ulm)
Konstantin Korovin (University of Manchester)
Daniel Le Berre (Université d'Artois)
Hans de Nivelle (Uniwersytet Wrocławski)
Albert Oliveras (Universitat Politècnica de Catalunya)
Nicola Ollivetti (Université Paul Cézanne)
Jens Otten (Universität Potsdam)
Jeff Pan (University of Aberdeen)
Lawrence Paulson (University of Cambridge)
Adam Pease (Articulate Software)
Nicolas Peltier (CNRS)
Ruzica Piskac (Max-Planck Institute for Software Systems)
Stefan Schlobach (Vrije Universiteit Amsterdam)
Renate Schmidt (University of Manchester), Co-Chair
Stephan Schulz (Technische Universität München), Co-Chair
Geoff Sutcliffe (University of Miami)
Laurent Théry (INRIA Sophia-Antipolis)
Dmitry Tishkovsky (University of Manchester)
Christoph Weidenbach (Max-Planck-Institut für Informatik)
Florian Widmann (Imperial College)
Christoph M. Wintersteiger (Microsoft Research)

## External reviewers

Carlos Areces
Christian Doczkal
Guido Governatori
Guillaume Hoffmann
Brian Huffman
Ullrich Hustadt
Dejan Jovanović
Tim King
Juan Antonio Navarro Perez
Christophe Ringeissen

# Table of Contents

# Practical Aspects of SAT Solving

Armin Biere
Johannes-Kepler-Universität Linz, Austria

**Abstract**

SAT solving techniques are used in many automated reasoning engines. This talk gives an overview on recent developments in practical aspects of SAT solver development. Beside improvements of the basic conflict driven clause learning (CDCL) algorithm, we also discuss improving and integrating advanced preprocessing techniques as inprocessing during search. The talk concludes with a brief overview on current trends in parallelizing SAT.

# Building an Efficient OWL 2 DL Reasoner

Boris Motik
University of Oxford

**Abstract**

The Ontology Web Language (OWL) has received considerable traction recently and is used in a number of industrial and practical applications. While decidable, all basic reasoning tasks for OWL are intractable (most of them are N2ExpTime-complete). Thus, in order to obtain a system capable of solving practically-relevant nontrivial problems, a number of theoretical and practical issues need to be resolved. In my talk I will present an overview of the techniques employed in HermiT, a state-of-the-art OWL reasoner developed at Oxford University. I will present the main ideas behind the hypertableau calculus and contrast them with the tableau calculi used in similar systems. Furthermore, I will discuss optimization techniques used in HermiT such as the blocking cache, individual reuse, and core blocking. Finally, I will discuss certain higher-level optimizations implemented on top of the basic calculus, such as the recently-developed optimized classification algorithm.

# Escape to Mizar from ATPs

Jesse Alama[*]
Center for Artificial Intelligence
New University of Lisbon
Portugal
j.alama@fct.unl.pt, http://centria.di.fct.unl.pt/~alama/

**Abstract**

We announce a tool for mapping E derivations to Mizar proofs. Our mapping complements earlier work that generates problems for automated theorem provers from Mizar inference checking problems. We describe the tool, explain the mapping, and show how we solved some of the difficulties that arise in mapping proofs between different logical formalisms, even when they are based on the same notion of logical consequence, as Mizar and E are (namely, first-order classical logic with identity).

## 1 Introduction

The problem of translating formal proofs expressed in different formats is an important research problem for automated reasoning. Proofs today come from many sources, and there are about as many implemented proof formats as there are different systems for interactive and automated theorem proving, not to mention the "pure" proof formats coming from mathematical logic. There is a choice about which axioms and rules of inference to pick. Even natural deduction comes in a number of shapes: Jáskowski, Gentzen, Fitch, Suppes... [17]. It seems likely that as the use of proof systems grows we will need to have better tools for mapping between different formalisms. This need has been recognized for a long time [26, 1], and it still seems we have some way to go. This paper discusses the problem of transforming derivations output by the E [20] automated theorem prover into Mizar texts.[1]

Mizar[2] is a language for writing mathematical texts in a "natural" style combined with a library of reasoning formalized in the Mizar language and verified by the Mizar proof checker. For the purpose of the present paper, the main feature of Mizar is its natural deduction-style proof language, grounded on a notion of "obvious inference" (to be explained below). We will ignore the large Mizar Mathematical Library (MML), an impressive collection going from the axioms of set theory to graduate-level pure mathematics. We will thus treat Mizar as a language and a suite of tools for carrying out arbitrary reasoning in first-order classical logic.

Related work is discussed in Section 2. Section 3 concerns the translation from E derivations to Mizar proofs. Because of the fine-grained level of detail offered by E and the simple multi-premise "obvious inference" rule of Mizar, the mapping is more or less straightforward, save for *skolemization* and *resolution*, neither of which have direct analogues in "human friendly" Mizar texts. Skolemization is discussed in Section 3.2 and our treatment of resolution is discussed in 3.3. The problem of making the generated Mizar texts more humanly comprehensible is discussed in Section 3.4. Section 4 concludes and proposes applications and further opportunities for development. Appendix A is a complete example of a text (a solution to the Dreadbury Mansion puzzle found by E, translated to Mizar) produced by our translation.

[*]Supported by the ESF research project *Dialogical Foundations of Semantics* within the ESF Eurocores program *LogICCC* (funded by the Portuguese Science Foundation, FCT LogICCC/0001/2007). Research for this paper was partially done while a visiting fellow at the Isaac Newton Institute for the Mathematical Sciences in the programme 'Semantics & Syntax'. Josef Urban inspired this project and provided many helpful suggestions. Artur Korniłowicz clarified some important details of Mizar proofs.

[1]Our work is available at https://github.com/jessealama/tptp4mizar.

[2]http://mizar.org

## 2 Related work

In recent years there is an interest in adding automation to interactive theorem proving systems. An important challenge is to make sense, at the level of the interactive theorem prover, of solutions produced by external automated reasoning tools. Such *proof reconstruction* has been done for Isabelle/HOL [15]. There, the problem of finding an Isabelle/HOL text suitable for solving an inference problem $P$ is done as follows:

1. Translate $P$ to a first-order theorem proving problem $P*$.
2. Solve $P*$ using an automated theorem prover, yielding solution $S*$.
3. Translate $S*$ into a Isabelle/HOL text, yielding a solution $S$ of the original problem.

The work described in this paper could be used to provide a similar service for Mizar. It is interesting to note that in the case of Mizar the semantics of the source logic and the logic of the external theorem prover are (essentially) the same: first-order classical logic with identity. In the Isabelle/HOL case, at step (1) there is a potential loss of information because of a mismatch of Isabelle/HOL's logic and the logic of the ATPs used to solve problems (which may not in any case matter at step (3)). In the Mizar context, two-thirds (steps (1) and (2)) of the problem has been solved [19]; our work was motivated by that paper. Steps toward (3) have been taken in the form of Urban's ott2miz[3]. In fact, more than 2/3 of the problem is solved. Our work here builds on ott2miz by accounting for the clause normal form transformation, rather than starting with the clause normal form of a problem. Our translated proofs thus start with (the Mizar form of) the relevant initial formulas, which arguably improves the readability of the proofs. Moreover, our tool works with arbitrary TPTP FOF problems and TSTP derivations produced by E, rather than with Otter proof objects. The restriction to E is not essential; there is no inherent obstacle to extending our work to handle TSTP derivations produced by other automated theorem provers, provided that these derivations (proof objects) are sufficiently detailed, like E's. One must acknowledge, of course, that providing high-quality, fine-grained proof objects is a challenging practical problem for automated theorem provers.

To account for the clausal normal form transformation, one needs to deal with skolemization. This is a well-known issue in discussions surrounding proof objects for automated theorem provers [4]. Interestingly, our method for handling skolemization (to be described below) is analogous to the handling of quantifiers in the problem opposite ours, namely, converting Mizar proofs to TSTP derivations [24] in the setting of MPTP (Mizar Problems for Theorem Provers) [23]. There, Henkin-style implications are a natural solution to the problem of justifying a substitution instance $\varphi(a)$ of a formula given that its generalization $\forall x \varphi$ is justified. Our translation of skolemization steps is virtually the same as this; see Section 3.2 for details.

Exporting and verifying of Mizar proofs by ATPs has been carried out [24]. Such work is an inverse of ours since it goes from Mizar proofs to ATP problems.

One can reasonably ask to what extent the derivation produced by E and the generated Mizar text are the same proof. We do not intend to enter into a discussion about the proof identity problem. For a discussion, see Došen [6]. Certainly the intension behind the mapping is to preserve the proof expressed by the E derivation. That the E derivation and the Mizar text generated from it are isomorphic will be clarified (but not proved) below. Mappings such as the one discussed in this paper can help contribute to a concrete investigation of the proof identity problem.

[3]See its homepage `https://github.com/JUrban/ott2miz` and its announcement `http://mizar.uwb.edu.pl/forum/archive/0306/msg00000.html` on the Mizar users mailing list.

It is well-known that derivations carried out in clause-based calculi (such as resolution and kindred methods) tend to be difficult to understand, if not downright inscrutable. An important problem for the automated reasoning community for many years is to find methods of understanding machine-discovered proofs. One approach to this problem is to map resolution derivations into natural deduction proofs. Much work has been done in this direction [12, 13, 8, 7, 10, 11]. The transformations we employ are rather simple. To "clean up" the generated text, we take advantage of the various proof "enhancers" bundled with the standard Mizar distribution [9, §4.6]. These enhancers suggest compressions of a Mizar text that make it more parsimonious while preserving its semantics. In the end, though, it would seem that the judgment of whether an "enhanced" Mizar text is the best representative of a resolution proof is something that has to be left to the reader.

## 3 Translating E derivations into Mizar texts

To construct a Mizar text from a first-order TSTP derivation, one first identifies the function and predicate symbols of the derivation and creates an *environment* for the text. Constructing an environment for a Mizar text amounts to creating a handful of XML files specifying the syntax and semantics of the symbols appearing in the derivation. Normally, one does not create Mizar environments by hand from scratch but rather builds on some preexisting formalizations. Since we do not use the Mizar library, we cannot use the usual Mizar toolchain to construct an environment.

To generate the Mizar text, we exploit recent developments concerning the Mizar parser [2]. We generate XML representations (parse trees) of Mizar texts which can then be rendered as a plain-text Mizar file. The XML representation leaves open the possibility of further manipulation of the text through, e.g., XSL transformations.

The input to our procedure is an E derivation in TSTP format [22].

Section 3.1 discusses the overall organization of the generated proof. In Section 3.2 we discuss the skolemization problem. In Section 3.3 we discuss the problem of resolution.

### 3.1 Global and local organization of the proof

After the first batch of transformations, the refutation is "groomed" in the following ways:

1. Linearly order the formulas.

   In TPTP problems, the order of formulas is immaterial. However, in a natural deduction argument, the order of formulas in Mizar cannot be arbitrary. We topologically sort the input ordered in the obvious way (if conclusion $A$ uses formula $B$ as a premise, then $B$ should appear earlier than $A$) and work with a linear order.

2. Separate reasoning done among the input assumptions from reasoning done with the negation of the conjecture.

To capture the spirit of proof by contradiction we refactor E refutations into so-called diffuse reasoning blocks. We write:

```
theorem φ
proof
  now
    assume ¬φ;
    S1: ⟨conclusion 1⟩ by ...;
    S2: ⟨conclusion 2⟩ by ...;
    ...
    Sn: ⟨conclusion n⟩ by ...;
    thus contradiction by S_a1, S_a2, ..., S_am
  end;
  hence thesis;
end;
```

This concludes the discussion of the organization of the generated Mizar proof.

## 3.2 Skolemization

E's finely detailed proof output contains not simply the derivation of $\bot$ starting from the clause form of the input formulas. E can also record the transformation of the input formulas into clause normal formal. It is important to preserve these inferences because they give information about what was actually given to E.Accounting for skolemization a well-known issue in generating proof objects [4, 5]. The difficulty is that skolem functions are curious creatures in an interactive setting like Mizar's. Introducing a function into a Mizar text requires that the use can prove existence and uniqueness of its definiens. But what is the definiens of a skolem function, and how can it be justified?

Our solution to the skolemization problem is to introduce axioms. To take a simple example, suppose we have $\forall x \exists y \varphi$, and from this $\forall x \varphi[y := f(x)]$ is "derived". We introduce at this point a new definition (treated as an axiom) whose definiens is:

$$(\forall x \exists y \varphi) \to \forall x \varphi[y := f(x)]$$

Our axiom-based solution to the skolemization problem is admittedly not ideal. Other approaches for dealing with skolemization are available. In principle, one could reconstruct all E's skolemization steps in Mizar using Mizar's choice operator.[4] To do this, given a formula $\psi := \forall x \exists y \varphi$, one can proceed as follows:

1. Introduce a new (non-dependent) type $\tau\psi$ inhabited (by definition) by those objects that satisfy the sentence $\forall x \exists y \varphi$.
2. Prove that $\tau_\psi$ is inhabited by exploiting the fact that the domain of interpretation of any first-order structure is non-empty.
3. Define $f$ outright using Mizar's built-in Hilbert choice operator:

   ```
   definition
     let x;
     func f equals the T;
   end;
   ```

   where T is the Mizar type corresponding to $\tau_\psi$.

Despite the advantage of being explicit, initial experiments with this "explicit skolemization" approach make clear that the precise details of skolemization steps matter: we have found that skolemization steps in which multiple skolem functions are introduced at once complicates the explicit approach; the algorithm we have just sketched does not apply to such cases. Since E's skolemization procedure can in fact produce such steps, explicit skolemization limits the scope of our tool compared to axiom-based skolemization. Of course, we could implement our own clausifier that provides us the required level of granularity of clausification. However, if we wish to account for every step of an arbitrary E derivation, then the axiom-based solution seems preferable.

[4]Unlike in Hilbert's $\varepsilon$-calculus, where the choice operator applies to formulas, the choice operator in Mizar applies to types.

## 3.3 Resolution

Targeting Mizar is sensible because it has a single rule of inference, by, which takes a variable number of premises. The intended meaning of an application

$$\frac{\varphi_1, \ldots, \varphi_n}{\varphi}\ \texttt{by}$$

of by is that $\varphi$ is an "obvious" inference from premises $\varphi_1, \ldots, \varphi_n$. See Davis [3] and Rudnicki [18] for more information about the the tradition of "obvious inference" in which Mizar works. The implementation in Mizar diverges from these proposals [25], but roughly speaking a conclusion in Mizar is obtained by an "obvious inference" in from some premises if there is a derivation of the conclusion from a set of assumptions in which at most one substitution instance of at most one universal premise is chosen.

The main difficulty for mapping arbitrary E derivations to Mizar texts is that Mizar's notion of "obvious inference" overlaps with resolution, but is neither weaker nor stronger than it. The consequence of this is that it is generally not the case that an application of resolution can be mapped to a single acceptable application of Mizar's by rule. Consider the following example:

$$\frac{\forall x[\neg A(x) \vee B(x)] \qquad \forall x, y[\neg B(x) \vee \neg B(x) \vee \neg B(y)]}{\forall x, y[\neg A(x) \vee \neg B(y)]}\ \text{Resolution}$$

This application of resolution[5] simply eliminates $B(x)$ from the premises. The difficulty here is that we cannot choose a single substitution instance of the premises such that we can find a Herbrand derivation, and hence the inference is non-obvious even though it is essentially (i.e., at the clause level) a single application of propositional resolution.

The reason for the difficulty is that we are working at the level of formulas rather than clauses. A solution is available: map the application of resolution not to a single application of Mizar's by rule, but to a proof:

```
premise1:
for X1 holds ((not A X1) or B X1);

premise2:
for X1, X2 holds ((not A X1) or (not B X1) or (not B X2));

theorem
for X1, X2 holds ((not A X1) or B X2)
proof
  let c1, c2;
  (not A c1) or B c1 by premise1;
  hence thesis by premise2;
end;
```

This Mizar proof has three steps and two applications of by. In each application of by, there is a single instance of a single universal formula (in the first case the universal formula is premise1, and in the second application the universal premise is premise2). Note that the substitution instances are not built from constants and function symbols, but from (fixed) variables.

## 3.4 Compressing Mizar proofs

The "epicycles" of resolution notwithstanding, Mizar is able to compress many of E's proof steps: many steps can be combined into a single acceptable application of Mizar's by rule of

[5]To be precise, an application of factoring is suppressed in this example.

inference. For example, if $\varphi$ is inferred from $\varphi'$ from variable renaming, and $\varphi'$ is inferred by an application of conjunction elimination to $\varphi''$, typically in the Mizar setting $\varphi$ can be inferred from $\varphi''$ alone by a single application of `by`. This is typical for most of the fine-grained rules of E's calculus: their applications are acceptable according to Mizar's `by`, and often they can be composed (sometimes multiple times) while still being acceptable to `by`. Other rules in E's proof calculus that can often be eliminated are variable rewritings, putting formulas into negation normal form, reordering of literals in clauses. More interesting compressions exploit the gap between "obvious inference" and E's more articulated calculus.

It seems to be a hard AI problem to transform arbitrary resolution proofs into human-comprehensible natural deductions. Machine-found proofs seem to have an artificial "flavor" that no rewriting spice can overcome. Still, some simple organizational principles can help to make the proof more manageable.

Compressing proofs helps us to get a sense of what the proof is about. The Mizar notion of obvious inference has been tested through daily work with substantial mathematical proofs for decades, and thus enjoys a time-tested robustness (though it is not always uncontroversial). It seems to be an open problem to specify what we mean by the "true" or "best" view of a proof. When Mizar texts come from E proofs, Mizar finds that the steps are usually excessively detailed (i.e., most steps are obvious) and can be compressed. On the other hand, often the whole proof cannot be compressed into a single application of `by`. We employ the algorithm discussed in [19]: a simple fixed-point algorithm is used to maximally compress a Mizar text. Thus, by repeatedly attempting to compress the proof until we reach the limits of `by`. Yet proof compression is not without its pitfalls. If one compresses Mizar proofs too much, the text can become as "inhuman" as the resolution proof from which it comes. This is a well-known phenomenon in the Mizar community [14]. Experience with texts generated by our translation shows that often considerable compression is possible, but at the cost of introducing a new artificial "scent" into the Mizar text.

## 4 Conclusion and future work

One naturally wants to extend the work here to work with output of other theorem provers, such as Vampire. There is no inherent difficulty in that, though it appears that the TSTP derivations output by Vampire contain different information compared to E proofs; the generic transformations described in Section 3.1 would carry over, but the mapping of skolemization and resolution steps of Sections 3.2 and 3.3 will likely need to be customized for Vampire.

The TPTP language recognizes definitions, but whether an automated theorem prover treats them differently from an axiom is unspecified. In Mizar, definitions play a vital role. After all, Mizar is designed to be a language for developing mathematical theories; only secondarily is it a language for representing solutions to arbitrary reasoning problems, as we are using it in this paper. One could try to detect definitions either by scanning the problem looking for formulas that have the form of definitions, or, if the original TPTP problem is available, one can extract the formulas whose TPTP role is `definition`. Such definition detection and synthesis has no semantic effect, but could make the generated Mizar texts more manageable and perhaps even facilitate new compressions.

At the moment the tool simply translates E derivations to Mizar proofs. A web-based frontend to the translator could help to spur increased usage (and testing) of our system. One can even imagine our tool as part of the SystemOnTPTP suite [21].

An important incompleteness of the current solution is the treatment of equality. Some atomic equational reasoning steps (specifically, inferences involving non-ground equality literals) in E derivations can be non-Mizar-obvious. One possible solution is to use Prover9's Ivy proof objects. Ivy derivations provide some information (namely, which instances of which variables in non-ground literals) that (at present) is missing from E's proof object output.

For the sake of clarity in the mapping of skolemization steps in E derivation to Mizar steps, we restricted attention to those E derivations in which each skolemization step introduces exactly one new skolem function. The restriction does not reflect a weakness of Mizar; it is a merely technical limitation and we intend to remove it.

We have thus completed the cycle started in [19] and returned from ATPs to Mizar. We leave it to the reader to decide whether he wishes to escape again.

## References

[1] P.B. Andrews. More on the problem of finding a mapping between clause representation and natural-deduction representation. *Journal of Automated Reasoning*, 7(2):285–286, 1991.

[2] Czesław Bylinski and Jesse Alama. New developments in parsing Mizar. 2012. To appear in the proceedings of *Intelligent Computer Mathematics*, 2012.

[3] M. Davis. Obvious logical inferences. In *Proceedings of the Seventh International Joint Conference on Artificial Intelligence*, pages 530–531, 1981.

[4] Hans de Nivelle. Extraction of proofs from the clausal normal form transformation. In *Computer Science Logic*, volume 2471 of *Lecture Notes in Computer Science*, pages 921–951, 2002.

[5] Hans de Nivelle. Translation of resolution proofs into short first-order proofs without choice axioms. *Information and Computation*, 199(1-2):24 – 54, 2005.

[6] Kosta Došen. Identity of proofs based on normalization and generality. *Bulletin of Symbolic Logic*, 9:477–503, 2003.

[7] U. Egly and K. Genther. Structuring of computer-generated proofs by cut introduction. *Computational Logic and Proof Theory*, pages 140–152, 1997.

[8] A. Felty and D. Miller. Proof explanation and revision. In *AAAI-86 Proceedings*, 1987.

[9] A. Grabowski, A. Kornilowicz, and A. Naumowicz. Mizar in a nutshell. *Journal of Formalized Reasoning*, 3(2):153–245, 2010.

[10] C. Lingenfelder. Structuring computer generated proofs. In *International Joint Conference on Artificial Intelligence*. Citeseer, 1989.

[11] Andreas Meier. System description: Tramp: Transformation of machine-found proofs into natural deduction proofs at the assertion level. In David McAllester, editor, *Automated Deduction - CADE-17*, volume 1831 of *Lecture Notes in Computer Science*, pages 460–464. Springer Berlin / Heidelberg, 2000.

[12] Dale Miller. Expansion tree proofs and their conversion to natural deduction proofs. In R. Shostak, editor, *7th International Conference on Automated Deduction*, volume 170 of *Lecture Notes in Computer Science*, pages 375–393. Springer Berlin / Heidelberg, 1984.

[13] Dale A. Miller. A compact representation of proofs. *Studia Logica*, 46:347–370, 1987. 10.1007/BF00370646.

[14] Karol Pąk. The methods of improving and reorganizing natural deduction proofs. In *MathUI10*, 2010.

[15] L. Paulson and Kong Woei Susanto. Source-level proof reconstruction for interactive theorem proving. In Klaus Schneider and Jens Bran, editors, *Theorem Proving in Higher-Order Logics*, volume 4732 of *Lecture Notes in Computer Science*, pages 232–245. Springer, 2007.

[16] F.J. Pelletier. Seventy-five Problems for Testing Automatic Theorem Provers. *Journal of Automated Reasoning*, 2(2):191–216, 1986.

[17] Francis Jeffry Pelletier. A brief history of natural deduction. *History and Philosophy of Logic*, 20(1):1–31, 1999.

[18] P. Rudnicki. Obvious inferences. *Journal of Automated reasoning*, 3(4):383–393, 1987.

[19] Piotr Rudnicki and Josef Urban. Escape to ATP for Mizar. In Pascal Fontaine and Aaron Stump, editors, *PxTP 2011: First International Workshop on Proof eXchange for Theorem Proving*, pages 46–59, 2011.

[20] Stephan Schulz. E-a brainiac theorem prover. *AI Communications*, 15(2):111–126, 2002.

[21] G. Sutcliffe. The TPTP Problem Library and associated infrastructure: The FOF and CNF parts, v3.5.0. *Journal of Automated Reasoning*, 43(4):337–362, 2009.

[22] Geoff Sutcliffe, Stephan Schulz, Koen Claessen, and Allen Van Gelder. Using the TPTP language for writing derivations and finite interpretations. In Ulrich Furbach and Natarajan Shankar, editors, *Automated Reasoning*, volume 4130 of *Lecture Notes in Computer Science*, pages 67–81. Springer Berlin / Heidelberg, 2006.

[23] Josef Urban. MPTP 0.2: Design, implementation, and initial experiments. *Journal of Automated Reasoning*, 37(1-2):21–43, 2006.

[24] Josef Urban and Geoff Sutcliffe. ATP-based cross-verification of Mizar proofs: Method, systems, and first experiments. *Mathematics in Computer Science*, 2(2):231–251, 2008.

[25] Freek Wiedijk. Checker. Available online at http://www.cs.ru.nl/~freek/mizar/by.pdf.

[26] Larry Wos. The problem of finding a mapping between clause representation and natural-deduction representation. *Journal of Automated Reasoning*, 6(2):211–212, 1990.

## A Pelletier's Dreadbury Mansion Puzzle: From E to Mizar

```
Ax1: ex X1 st (lives X1 & killed X1,agatha) by AXIOMS:1;

Ax2: lives X1 implies (X1 = agatha or X1 = butler or X1 = charles) by AXIOMS:2;

Ax3: killed X1,X2 implies hates X1,X2 by AXIOMS:3;

Ax4: killed X1,X2 implies (not richer X1,X2) by AXIOMS:4;

Ax5: hates agatha,X1 implies (not hates charles,X1) by AXIOMS:5;

Ax6: (not X1 = butler) implies hates agatha,X1 by AXIOMS:6;

Ax7: (not richer X1,agatha) implies hates butler,X1 by AXIOMS:7;

Ax8: hates agatha,X1 implies hates butler,X1 by AXIOMS:8;

Ax9: ex X2 st (not hates X1,X2) by AXIOMS:9;

Ax10: not agatha = butler by AXIOMS:10;

S1: killed skolem1,agatha by Ax1,SKOLEM:def 1;

S2: agatha = skolem1 or butler = skolem1 or charles = skolem1 by Ax2,Ax1,SKOLEM:def 1;

S3: not hates agatha,(skolem2 butler) by Ax9,SKOLEM:def 2,Ax8;

S4: hates charles,agatha or skolem1 = butler or skolem1 = agatha by Ax3,Ax1,SKOLEM:def 1,S2;

S5: butler = (skolem2 butler) by S3,Ax6;

S6: not hates butler,butler by Ax9,SKOLEM:def 2,S5;

S7: hates butler,butler or skolem1 = agatha by Ax4,Ax7,Ax1,SKOLEM:def 1,Ax5,S4,Ax6,Ax10;

S8: skolem1 = agatha by S7,S6;

theorem
killed agatha,agatha
proof
  now
    assume S9: not killed agatha,agatha;
    thus contradiction by S1,S8,S9;
  end;
  hence thesis;
end;
```

Pelletier's Dreadbury Mansion [16] goes as follows:

> Someone who lives in Dreadbury Mansion killed Aunt Agatha. Agatha, the butler, and Charles live in Dreadbury Mansion, and are the only people who live therein. A killer always hates his victim, and is never richer than his victim. Charles hates no one that Aunt Agatha hates. Agatha hates everyone except the butler. The butler hates everyone not richer than Aunt Agatha. The butler hates everyone Aunt Agatha hates. No one hates everyone. Agatha is not the butler.

The problem is: Who killed Aunt Agatha? (Answer: she killed herself.) The problem belongs to the TPTP Problem Library (it is known there as PUZ001+1) and can easily by solved by many automated theorem provers. Above is the result of mapping E's solution to a standalone Mizar text and then compressing it as described in Section 3.4. Two skolem functions `skolem1` (arity 0) and `skolem2` (arity 2) are introduced. There are 10 axioms and 8 steps that do not depend on the negation of the conjecture (`killed agatha,agatha`) This problem is solved essentially by forward reasoning from the axioms; proof by contradiction is unnecessary, but that is the nature of E's solution.

# Implementing Different Proof Calculi for First-order Modal Logics

**– Extended Abstract –**

Christoph Benzmüller[1]*, Jens Otten[2] and Thomas Raths[2]†

[1] Institut für Informatik, FU Berlin, Germany
`c.benzmueller@googlemail.com`
[2] Institut für Informatik, University of Potsdam, Germany
`jeotten|traths@cs.uni-potsdam.de`

## 1 Introduction

*Modal logics* extend classical logic with the modalities "it is necessarily true that" and "it is possibly true that" represented by the unary operators $\Box$ and $\Diamond$, respectively. *First-order* modal logics (FMLs) extend propositional modal logics by *domains* specifying sets of objects that are associated with each world, and the standard universal and existential quantifiers [7].

FMLs have many applications, e.g., in planning, natural language processing, program verification, querying knowledge bases, and modeling communication. These applications motivate the use of *automated theorem proving* (ATP) systems for FMLs. Whereas there are some ATP systems available for propositional modal logics, e.g., MSPASS [9] and modleanTAP [1], there were — until recently — no (correct) ATP systems that can deal with the full first-order fragment of modal logics.

This abstract presents several new ATP systems for FML and sketches their calculi and working principles. The abstract also summarizes the results of a recent comparative evaluation of these new provers (see [4] for further details).

The syntax of first-order modal logic adopted here is: $F, G ::= P(t_1, \ldots, t_n) \mid \neg F \mid F \wedge G \mid F \vee G \mid F \Rightarrow G \mid \Box F \mid \Diamond F \mid \forall x F \mid \exists x F$. The symbols $P$ are $n$-ary ($n \geq 0$) relation constants which are applied to terms $t_1, \ldots, t_n$. The $t_i$ ($0 \leq i \leq n$) are ordinary first-order terms and they may contain function symbols. The usual precedence rules for logical constants are assumed.

Regarding semantics a well accepted and straightforward notion of Kripke style semantics for FML is adopted [7]. In particular, it is assumed that constants and terms are denoting and rigid, i.e. they always pick an object and this pick is the same object in all worlds. Regarding the universe of discourse constant domain, cumulative domain and varying domain semantics are considered.

The following new ATP systems for FML were developed by the authors (partly as extensions of other systems); they support different combinations of modal logics and domain semantics (GQML-Prover [19] has not been included since it returned incorrect results in our experiments for several formulae):

*This author is funded by the German Research Foundation DFG under reference number BE2501/9-1.
†This author is funded by the German Research Foundation DFG under reference number KR858/9-1.

| ATP system | base technique | modal logics | domain semantics |
|---|---|---|---|
| MleanSeP 1.2 | sequent calculus | K,K4,D,D4,T,S4 | const.,cumul. |
| MleanTAP 1.3 | tableau calculus | D,T,S4,S5 | const.,cumul.,varying |
| MleanCoP 1.2 | connection calculus | D,T,S4,S5 | const.,cumul.,varying |
| f2p-MSPASS 3.0 | instance-based method | K,K4,K5,KB,D,T,S4,S5 | const.,cumul. |
| LEO-II 1.3.2-M1.0 | embedding in HOL | K,K4,K5,KB,D,D4,T,S4,S5 | const.,cumul.,varying |
| Satallax 2.2-M1.0 | embedding in HOL | K,K4,K5,KB,D,D4,T,S4,S5 | const.,cumul.,varying |

## 2 Calculi and ATP Systems for FML

**Sequent Calculus.** The classical sequent calculus $LK$ [8] is probably the most elegant calculus for classical logic and used in many interactive proof systems. This calculus can be extended to modal logics with cumulative domains by adding the *modal rules* $\Box$*-left*, $\Box$*-right*, $\Diamond$*-left*, and $\Diamond$*-right*. These rules introduce the modal operators $\Box$ and $\Diamond$ into the left side or the right side of the sequent, respectively. All rules of the classical sequent calculus, e.g., the rules for the quantifiers remain unchanged [20].

The *sequent calculus* for the modal logics K, K4, D, D4, T, and S4 with cumulative domains consists of the axiom and rules of the classical sequent calculus and the four additional rules shown in Figure 1, with $\Gamma_\Box := \{\Box G \mid \Box G \in \Gamma\}$, $\Delta_\Diamond := \{\Diamond G \mid \Diamond G \ \in \Delta\}$, $\Gamma_{(\Box)} := \{G \mid \Box G \in \Gamma\}$, $\Delta_{(\Diamond)} := \{G \mid \Diamond G \in \Delta\}$, $\Gamma_{[\Box]} := \Gamma_\Box \cup \ \Gamma_{(\Box)}$, and $\Delta_{[\Diamond]} := \Delta_\Diamond \cup \Delta_{(\Diamond)}$. A *sequent proof* for a modal formula $F$ is a derivation of $\vdash F$ in the modal sequent calculus, in which all leaves are closed by axioms.

$$\frac{\Gamma^+, F \vdash \Delta^+}{\Gamma, \Box F \vdash \Delta}\ \Box\textit{-left} \qquad \frac{\Gamma^* \vdash F, \Delta^*}{\Gamma \vdash \Box F, \Delta}\ \Box\textit{-right} \qquad \frac{\Gamma^*, F \vdash \Delta^*}{\Gamma, \Diamond F \vdash \Delta}\ \Diamond\textit{-left} \qquad \frac{\Gamma^+ \vdash F, \Delta^+}{\Gamma \vdash \Diamond F, \Delta}\ \Diamond\textit{-right}$$

| logic | $\Gamma^+$ | $\Delta^+$ | $\Gamma^*$ | $\Delta^*$ |
|---|---|---|---|---|
| K | (no rules) | | $\Gamma_{(\Box)}$ | $\Delta_{(\Diamond)}$ |
| K4 | (no rules) | | $\Gamma_{[\Box]}$ | $\Delta_{[\Diamond]}$ |
| D | $\Gamma_{(\Box)}$ | $\Delta_{(\Diamond)}$ | $\Gamma_{(\Box)}$ | $\Delta_{(\Diamond)}$ |

| logic | $\Gamma^+$ | $\Delta^+$ | $\Gamma^*$ | $\Delta^*$ |
|---|---|---|---|---|
| D4 | $\Gamma_{[\Box]}$ | $\Delta_{[\Diamond]}$ | $\Gamma_{[\Box]}$ | $\Delta_{[\Diamond]}$ |
| T | $\Gamma$ | $\Delta$ | $\Gamma_{(\Box)}$ | $\Delta_{(\Diamond)}$ |
| S4 | $\Gamma$ | $\Delta$ | $\Gamma_\Box$ | $\Delta_\Diamond$ |

Figure 1: The additional modal rules of the modal sequent calculus

The modal sequent calculus captures the cumulative domain condition. There are no similar cut-free sequent calculi for modal logics with constant or varying domains or for the modal logic S5 [20].

MleanSeP is an ATP system written in PROLOG that implements the sequent calculus for several modal logics. It can be downloaded at `http://www.leancop.de/mleansep/`. MleanSeP performs proof search in an analytic way, i.e. the sequent rules are applied from bottom to top. Furthermore, free-variables are used in combination with a dynamic Skolemization that is calculated during the proof search. Together with the occurs-check of the term unification algorithm this ensures that all derivations respect the Eigenvariable condition. To deal with constant domains, the Barcan formula is automatically added to the given formula in a preprocessing step. The *Barcan formula (scheme)* has the form $\forall\vec{x}(\Box P(\vec{x})) \Rightarrow \Box\forall\vec{x}P(\vec{x})$ with $\vec{x} = x_1, \ldots, x_n$ for all predicates $P$ with $n \geq 1$.

$$\frac{(\Box F)^1\!: p}{F^1\!: p \circ V^*}\,\Box^1 \qquad \frac{(\Diamond F)^0\!: p}{F^0\!: p \circ V^*}\,\Diamond^0 \qquad \frac{(\Box F)^0\!: p}{F^0\!: p \circ a^*}\,\Box^0 \qquad \frac{(\Diamond F)^1\!: p}{F^1\!: p \circ a^*}\,\Diamond^1$$

Figure 2: The four additional rules of the modal tableau calculus

**Tableau Calculus.** In general, the (classical) tableau calculus can be seen as compact representations of the (classical) sequent calculus. The classical tableau calculus [16] can be extended to several modal logics by adding a prefix to each formula occurring in a tableau rule [6]. The following tableau calculus for modal logic uses free variables not only within terms but also within prefixes. It is based on the matrix characterization for modal logic [20] but uses a tableau-based search to ensure that all paths contain a complementary connection. A *prefix* is a string consisting of (prefix) variables and (prefix) constants. Essentially, it represents a world path that captures the particular Kripke semantics of the modal logic in question. A *prefixed formula* has the form $F^{pol}:p$, where $F$ is a (first-order) modal formula, *pol* $\in \{0,1\}$ is its polarity and $p$ is its prefix.

The *(prefixed) tableau calculus* for the modal logics D, T, S4, and S5 consists of the rules of the classical tableau calculus [6], which do not change the prefix $p$ of formulae, and the four additional rules shown in Figure 2. $V^*$ is a new prefix variable, $a^*$ is a new prefix constant and $\circ$ is the composition of two strings. A branch is closed if, and only if, it contains a pair of literals of the form $\{A_1^1:p_1, A_2^0:p_2\}$ that are complementary under a term substitution $\sigma_Q$ and an additional modal substitution $\sigma_M$, i.e. $\sigma_Q(A_1)=\sigma_Q(A_2)$ and $\sigma_M(p_1)=\sigma_M(p_2)$. A tableau proof for a prefixed formula $F^{pol}:p$ is a tableau derivation such that all branches are (simultaneously) closed for a pair of term and modal substitutions $(\sigma_Q,\sigma_M)$. A *tableau proof* for a modal formula $F$ is a tableau proof for $F^0:\varepsilon$.

In the prefixed tableau calculus the particular modal logic is specified by distinct properties of the modal substitution $\sigma_M$. An additional admissible criterion on $\sigma_M$ is used to capture the different domain variants, i.e., constant, cumulative, or varying domains.

MleanTAP is a compact ATP system written in PROLOG that implements the modal tableau calculus. In can be downloaded at `http://www.leancop.de/mleantap/`. The proof search of MleanTAP is split up into two phases. The first phase performs a purely classical proof search. In the second phase, after a classical tableau proof is found, the prefixes $p_1$ and $p_2$ of all literals that close branches in the classical tableau are unified. The unification of these prefixes is done by a specialized string unification algorithm. If the prefix unification fails, alternative classical proofs (and prefixes) are computed. In order to fulfill the distinct properties of the modal substitution $\sigma_M$, a specific unification algorithm is used for each modal logic that also respects the admissible criterion.

**Connection Calculus.** Connection calculi use a *connection-driven* search strategy and are already successfully used for automated theorem proving in classical and intuitionistic logic [11, 12]. A *connection* is a pair of literals, $\{A,\neg A\}$ or $\{A^1,A^0\}$, with the same predicate symbols but different polarities. The connection calculus for classical logic is adapted to modal logic by adding prefixes to all literals. Formally, a *prefix* is a string over an alphabet $\mathcal{V}\cup\Pi$, where $\mathcal{V}$ is a set of *prefix variables*, denoted by $V$, and $\Pi$ is a set of *prefix constants*, denoted by $a$. It is defined in the same way as in the tableau calculus. Subformulae of the form $(\Box F)^1$ or $(\Diamond F)^0$ extend the prefix by a variable $V$, subformulae of the form $(\Box F)^0$ or $(\Diamond F)^1$ extend the prefix by a constant $a$ (see also Figure 2). For the modal logic S5 only the last character of all prefixes is considered (or $\varepsilon$ if the prefix is the empty string $\varepsilon$).

Proof-theoretically, a prefix of a formula $F$ captures the modal context of $F$ and specifies the sequence of modal rules of the sequent calculus that have to be applied (analytically) in order to obtain $F$ in the sequent. Semantically, a prefix denotes a specific world in a model [6, 20]. The prefixes of the two literals in a connection, which corresponds to an axiom in the sequent calculus, need to denote

*Axiom (A)* $\dfrac{}{\{\}, M, Path}$ *Start (S)* $\dfrac{C_2, M, \{\}}{\varepsilon,\ M,\ \varepsilon}$ and $C_2$ is copy of $C_1 {\in} M$

*Reduction (R)* $\dfrac{C, M, Path{\cup}\{L_2{:}\,p_2\}}{C{\cup}\{L_1{:}\,p_1\}, M, Path{\cup}\{L_2{:}\,p_2\}}$ and $\{L_1{:}\,p_1, L_2{:}\,p_2\}$ is $\sigma$-complementary

*Extension (E)* $\dfrac{C_2\backslash\{L_2{:}\,p_2\}, M, Path{\cup}\{L_1{:}\,p_1\} \quad C, M, Path}{C{\cup}\{L_1{:}\,p_1\}, M, Path}$ and $C_2$ is a copy of $C_1 \in M$, $L_2{:}p_2 \in C_2$, and $\{L_1{:}\,p_1, L_2{:}\,p_2\}$ is $\sigma$-complementary

Figure 3: The modal connection calculus

the same world, hence, they need to unify under a modal substitution. A connection $\{A_1^1 : p_1, A_2^0 : p_2\}$ is *$\sigma$-complementary*, for $\sigma := (\sigma_Q, \sigma_M)$, if $\sigma_Q(A_1) = \sigma_Q(A_2)$ and $\sigma_M(p_1) = \sigma_M(p_2)$, where $\sigma_Q$ is the standard *term substitution* and $\sigma_M : \mathcal{V} \to (\mathcal{V} \cup \Pi)^*$ is the *modal substitution* that assigns a string over the alphabet $\mathcal{V} \cup \Pi$ to every element in $\mathcal{V}$. The substitutions $\sigma_Q$ and $\sigma_M$ induce a reduction ordering, which has to be irreflexive [20]. Alternatively, a Skolemization technique can be used for the term Eigenvariables and for the prefix constants, as demonstrated by Otten [10].

For the modal logics D and T the *accessibility condition* $|\sigma_M(V)| = 1$ or $|\sigma_M(V)| \leq 1$ has to hold for all $V \in \mathcal{V}$, respectively. The accessibility condition encodes the characteristics of each modal logic. Like for the modal tableau calculus, $\sigma_M$ has to be admissible with respect to $\sigma_Q$. The admissible criterion depends on the domain condition, i.e. it is different for constant, cumulative and varying domains.

The matrix of a formula $F$ is a set of clauses that represents the disjunctive normal form of $F$ [5]. In the *prefixed matrix* $M$ of $F$ each literal $L$ is additionally marked with its prefix $p$. The axiom and the rules of the *modal connection calculus* are defined in Figure 3. $M$ is the prefixed matrix of $F$, the *subgoal clause* $C$ and the *active path* $Path$ are sets of (prefixed) literals or $\varepsilon$. $\sigma = (\sigma_Q, \sigma_M)$ is an admissible substitution and $\sigma_Q$ and $\sigma_M$ are rigid, i.e. they are applied to the whole derivation.

A connection proof for $C, M, Path$ is a derivation such that all leaves are axioms for an admissible substitution $\sigma = (\sigma_Q, \sigma_M)$. A *modal connection proof* for the matrix $M$ is a modal connection proof for $\varepsilon, M, \varepsilon$. Correctness and completeness proofs are based on the the matrix characterization for modal logic [20] and the correctness and completeness of the connection calculus [5].

MleanCoP [13] is an implementation of the modal connection calculus. It can be downloaded at `http://www.leancop.de/mleancop/`. It is based on leanCoP, an automated theorem prover for first-order classical logic [11]. To adapt the implementation to the modal connection calculus the leanCoP prover is extended by (a) prefixes that are added to literals and collected during the proof search and (b) an additional list that contains term variables together with their prefixes in order to check the domain condition. First, MleanCoP performs a classical proof search. After a classical proof is found, the prefixes of the literals in each connection are unified and the domain condition is checked. A different unification algorithm is used for each of the modal logics D, T, S4, and S5. For the modal logic K, the matrix characterization [20] requires to check an additional criterion, which cannot be integrated into the modal connection calculus [13] in a straightforward way. This also applies to the modal tableau calculus presented above. MleanCoP uses additional techniques to prune the search space: regularity, lemmata, restricted backtracking, a definition clausal form translation, and a fixed strategy scheduling; see [12] for details.

**Instance-Based Method.** Instance-based methods consist of two components. The first component adds instances of subformulae to the given formula and grounds the resulting formula, i.e. removes quantifiers and replaces all variables by a unique constant. The second component consists of an ATP system for propositional logic to find a proof or counter model for the ground formula. This method can be adapted to modal logic by using an ATP system for modal propositional logic. The basic approach works for the cumulative domain condition and formulae that contain either only existential or only universal quantifiers. This restriction is due to the dependency of applications of the modal rules and the quantifier rules, which cannot be captured by the standard Skolemization technique.

f2p-MSPASS is an implementation of the instance-based method for first-order modal logic. The first component, called first2p, adds instances of subformulae to the FML formula and grounds the resulting formula. It does not translate the given formula into any clausal form but preserves its structure. For the second component the propositional modal ATP system MSPASS [9] is used. MSPASS is an extension of and incorporated into the resolution-based ATP system SPASS. By default the standard relational translation from modal logic into classical logic is applied. To deal with constant domains, first2p adds the Barcan formula (scheme) to the given FML formula in a preprocessing step.

**Embedding into Classical Higher-Order Logic.** Various non-classical logics, including FMLs, can be embedded in classical higher-order logic (HOL) [2, 3]. The approach exploits the fact that Kripke structures can be elegantly modeled in HOL [3]: FML propositions $F$ are associated with HOL terms $F_\rho$ of predicate type $\rho := \iota \rightarrow o$. Type $o$ denotes the set of truth values and type $\iota$ is associated with the domain of possible worlds. Thus, the application $(F_\rho w_\iota)$ corresponds to the evaluation of FML proposition $F$ in world $w$. Consequently, validity is formalized as $vld_{\rho\rightarrow o} = \lambda F_\rho \forall w_\iota F w$. Classical connectives like $\neg$ and $\vee$ are simply lifted to type $\rho$ as follows: $\neg_{\rho\rightarrow\rho} = \lambda F_\rho \lambda w_\iota \neg F w$ and $\vee_{\rho\rightarrow\rho\rightarrow\rho} = \lambda F_\rho \lambda G_\rho \lambda w_\iota (F w \vee G w)$. $\Box$ is modeled as $\Box_{\rho\rightarrow\rho} = \lambda F_\rho \lambda w_\iota \forall v_\iota (\neg R w v \vee F v)$, where constant symbol $R_{\iota\rightarrow\iota\rightarrow o}$ denotes the accessibility relation of the $\Box$ operator, which remains unconstrained in logic K. Further logical connectives are defined as usual: $\wedge = \lambda F_\rho \lambda G_\rho \neg(\neg F \vee \neg G)$, $\Rightarrow = \lambda F_\rho \lambda G_\rho (\neg F \vee G)$, $\Diamond = \lambda F_\rho \neg\Box\neg F$. To obtain e.g. modal logic S4, $R$ is axiomatized as reflexive and transitive. Generally, this can be done 'semantically' (e.g. with axiom $\forall x(Rxx)$ for reflexivity) or 'syntactically' (e.g. with axiom $vld\ \forall F_\rho\ \Box F \Rightarrow F$, where quantification over propositions is employed [3]). Arbitrary normal modal logics extending K can be axiomatized this way. However, in some cases only the semantic approach (e.g. for the irreflexivity of $R$) or the syntactic approach (e.g. for McKinsey's axiom) is applicable.

For individuals a further base type $\mu$ is reserved in HOL. Universal quantification $\forall x F$ is introduced as syntactic sugar for $\Pi \lambda x F$, where $\Pi$ is defined as follows: $\Pi_{(\mu\rightarrow\rho)\rightarrow\rho} = \lambda H_{\mu\rightarrow\rho} \lambda w_\iota \forall x_\mu H x w$. For existential quantification, $\Sigma = \lambda H_{\mu\rightarrow\rho} \neg\Pi\lambda x_\iota \neg H x$ is introduced. $\exists x F$ is then syntactic sugar for $\Sigma\lambda x F$. $n$-ary relation symbols P, $n$-ary function symbols $f$ and individual constants $c$ in FML obtain types $\mu_1 \rightarrow \ldots \rightarrow \mu_n \rightarrow \rho$, $\mu_1 \rightarrow \ldots \rightarrow \mu_n \rightarrow \mu_{n+1}$ (with $\mu_i = \mu$ for $0 \leq i \leq n+1$) and $\mu$, respectively.

For any FML formula $F$ holds: $F$ is a valid in modal logic K for constant domain semantics if and only if $vld\, F_\rho$ is valid in HOL for Henkin semantics. This correspondence provides the foundation for proof automation of FMLs with HOL-ATP systems. The correspondence follows from Benzmüller and Paulson [3], who prove a more general result for FMLs with additional quantification over propositional variables. (However, function and constant symbols are avoided in their work to achieve a leaner theory.)

The above approach is adopted for varying domain semantics as follows: 1. $\Pi$ is now defined as $\Pi = \lambda H_{\mu\rightarrow\rho} \lambda w_\iota \forall x_\mu \mathtt{exInW} x w \Rightarrow H x w$, where relation $\mathtt{exInW}_{\mu\rightarrow\iota\rightarrow o}$ (for 'exists in world') relates individuals with worlds. 2. The non-emptiness axiom $\forall w_\iota \exists x_\mu \mathtt{exInW} x w$ for these individual domains is added. 3. For each individual constant symbol $c$ an axiom $\forall w_\iota \mathtt{exInW} c w$ is postulated; these axioms enforce the designation of $c$ in the individual domain of each world $w$. Analogous designation axioms are required for function symbols.

Table 1: Number of proved monomodal problems of the QMLTP library

| Logic | Domain | f2p-MSPASS | MleanSeP | MleanTAP | LEO-II | Satallax | MleanCoP |
|---|---|---|---|---|---|---|---|
| K | varying | - | - | - | 73 | 104 | - |
| | cumulative | 70 | 121 | - | 89 | 122 | - |
| | constant | 67 | 124 | - | 120 | 146 | - |
| D | varying | - | - | 100 | 81 | 113 | 179 |
| | cumulative | 79 | 130 | 120 | 100 | 133 | 200 |
| | constant | 76 | 134 | 135 | 135 | 160 | 217 |
| T | varying | - | - | 138 | 120 | 170 | 224 |
| | cumulative | 105 | 163 | 160 | 139 | 192 | 249 |
| | constant | 95 | 166 | 175 | 173 | 213 | 269 |
| S4 | varying | - | - | 169 | 140 | 207 | 274 |
| | cumulative | 121 | 197 | 205 | 166 | 238 | 338 |
| | constant | 111 | 197 | 220 | 200 | 261 | 352 |
| S5 | varying | - | - | 219 | 169 | 248 | 359 |
| | cumulative | 140 | - | 272 | 215 | 297 | 438 |
| | constant | 131 | - | 272 | 237 | 305 | 438 |

For cumulative domain semantics the axiom $\forall x_\mu \forall v_\iota \forall w_\iota \mathtt{exInW} x v \wedge R v w \Rightarrow \mathtt{exInW} x w$ is additionally postulated. It states that the individual domains are increasing along relation $R$.

The above approach can be employed in combination with any HOL ATP system (various candidate systems are presented by Sutcliffe and Benzmüller [17]). Here we use LEO-II (`http://www.leoprover.org`) and Satallax (`http://www.ps.uni-saarland.de/~cebrown/satallax`). The conversion to `thf0`-syntax [17] and the provision of the above axioms is realized with the new preprocessor tool FMLtoHOL (1.0) (hence the suffices '-M1.0' on p. 1).

## 3 Evaluation Summary

The introduced ATP systems were evaluated on all 580 monomodal problems of version 1.1 of the QMLTP library [14]. The QMLTP library is a benchmark library for testing and evaluating ATP systems for FML, similar to the TPTP library for classical logic [18] and the ILTP library for intuitionistic logic [15]. In the experiments the following modal logics were considered: K, D, T, S4, and S5 with constant, cumulative, and varying domain semantics. These modal logics are supported by most of the described ATP systems. All tests were conducted on a 3.4 GHz Xeon system with 4 GB RAM running Linux 2.6.24-24.x86_64. The CPU time limit was set to 600 seconds. All ATP systems and components written in PROLOG use ECLiPSe PROLOG 5.10. LEO-II 1.3.2 was compiled with OCaml 3.12, and uses prover E 1.4. For Satallax a binary of version 2.2 is used. For MSPASS the sources of SPASS 3.0 were compiled using the GNU gcc 4.2.4 compiler.

Table 1 gives an overview of the test results for each prover. It contains the number of proved problems for each considered logic and each domain condition. MleanCoP is the strongest prover for logics D, T, S4 and S5, followed by Satallax. For logic K Satallax performs best. f2p-MSPASS cannot be applied to 299 problems as these problems contain both existential and universal quantifiers. f2p-MSPASS, Satallax and MleanCoP also find counter models for many (invalid) FML formulae. E.g., for T with cumulative domains, these ATP systems found counter models for 89, 90, and 125 problems, respectively. In addition to the 580 monomodal logic problems there are also 20 multimodal logic problems in the QMLTP library. Currently, only Satallax and LEO-II are applicable to them. LEO-II proves 15 of these and Satallax 14. The theorem prover leanTAP 2.3 for first-order classical logic was run on the 580 monomodal problems in the QMLTP library, in which all modal operators have been removed. It (classically) proves 296 problems and refutes one problem.

**Acknowledgements.** We thank Geoff Sutcliffe and Chad Brown for their valuable input to this work.

## References

[1] B. Beckert, R. Goré. Free Variable Tableaux for Propositional Modal Logics. In D. Galmiche, Ed., *TABLEAUX-1997*, LNAI 1227, pp. 91–106, Springer, 1997.

[2] C. Benzmüller, Combining and Automating Classical and Non-Classical Logics in Classical Higher-Order Logic, Annals of Mathematics and Artificial Intelligence, 62:103-128, 2011.

[3] C. Benzmüller, L. Paulson. Quantified Multimodal Logics in Simple Type Theory. Logica Universalis, 2012. DOI 10.1007/s11787-012-0052-y

[4] C. Benzmüller, T. Raths, J. Otten. Implementing and Evaluating Provers for First-order Modal Logics, Proceedings of ECAI'2012. To appear.

[5] W. Bibel. *Automated Theorem Proving*. Vieweg, Wiesbaden, 1987.

[6] M. Fitting. *Proof Methods for Modal and Intuitionistic Logic*. D. Reidel, Dordrecht, 1983.

[7] M. Fitting, R. L. Mendelsohn. *First-Order Modal Logic*. Kluwer, 1998.

[8] G. Gentzen. Untersuchungen über das logische Schließen. *Mathem. Zeitschrift*, 39:176–210, 405–431, 1935.

[9] U. Hustadt, R. A. Schmidt. MSPASS: Modal Reasoning by Translation and First-Order Resolution. R. Dyckhoff., Ed., *TABLEAUX-2000*, LNAI 1847, pp. 67–81. Springer, 2000.

[10] J. Otten. Clausal Connection-Based Theorem Proving in Intuitionistic First-Order Logic. In B. Beckert, Ed., *TABLEAUX 2005*, LNAI 3702, pp. 245–261. Springer, 2005.

[11] J. Otten. leanCoP 2.0 and ileanCoP 1.2: High Performance Lean Theorem Proving in Classical and Intuitionistic Logic. *IJCAR 2008*, LNCS 5195, pp. 283–291. Springer, 2008.

[12] J. Otten. Restricting Backtracking in Connection Calculi. *AI Communications* 23:159–182, 2010.

[13] J. Otten. Implementing Connection Calculi for First-order Modal Logics. *9th International Workshop on the Implementation of Logics*, Merida/Venezuela, 2012.

[14] T. Raths, J. Otten. The QMLTP Problem Library for First-order Modal Logics. *IJCAR-2012*, to appear.

[15] T. Raths, J. Otten, C. Kreitz. The ILTP Problem Library for Intuitionistic Logic. *Journal of Automated Reasoning*, 38(1–3): 261–271, 2007.

[16] R. M. Smullyan. *First-Order Logic*. Springer, 1968.

[17] G. Sutcliffe and C. Benzmüller. Automated Reasoning in Higher-Order Logic using the TPTP THF Infrastructure. Journal of Formalized Reasoning, 3(1):1-27, 2010.

[18] G. Sutcliffe. The TPTP Problem Library and Associated Infrastructure: The FOF and CNF Parts, v3.5.0. *Journal of Automated Reasoning*, 43(4):337–362, 2009.

[19] V. Thion, S. Cerrito, M. Cialdea Mayer. A General Theorem Prover for Quantified Modal Logics. In U. Egly, C. G. Fermüller, Eds., *TABLEAUX-2002*, LNCS 2381, pp. 266–280. Springer, 2002.

[20] L. Wallen. *Automated deduction in nonclassical logic*. MIT Press, Cambridge, 1990.

# Experiments on the feasibility of using a floating-point simplex in an SMT solver

Diego C B de Oliveira
CNRS / Verimag
Grenoble, France
diego.caminha@imag.fr

David Monniaux
CNRS / Verimag
Grenoble, France
david.monniaux@imag.fr

**Abstract**

SMT solvers use simplex-based decision procedures to solve decision problems whose formulas are quantifier-free and atoms are linear constraints over the rationals. State-of-art SMT solvers use rational (exact) simplex implementations, which have shown good performance for typical software, hardware or protocol verification problems over the years. Yet, most other scientific and technical fields use (inexact) floating-point computations, which are deemed far more efficient than exact ones. It is therefore tempting to use a floating-point simplex implementation inside an SMT solver, though special precautions must be taken to avoid unsoundness.

In this work, we describe experimental results, over common benchmarks (SMT-LIB) of the integration of a mature floating-point implementation of the simplex algorithm (GLPK) into an existing SMT solver (OpenSMT). We investigate whether commonly cited reasons for and against the use of floating-point truly apply to real cases from verification problems.

## 1 Introduction

Arithmetic is widely present in verification problems. The most common method for solving satisfiability problems modulo the theory of linear real (or, equivalently, rational) arithmetic is a combination of a DPLL SAT-solver and a decision procedure for conjunctions of linear (in)equalities, obtained by running phase I of the simplex algorithm [5].[1] Furthermore, problems over the theory of linear *integer* arithmetic are most often reduced to problems over the reals by judicious use of Gomory cuts, branch-and-bound, and other techniques from integer linear programming; thus the performance of the decision procedure for linear real arithmetic also conditions that of linear integer arithmetic.

The simplex implementation inside an SMT solver based on the DPLL($\mathcal{T}$) approach [8, 5] has the following tasks:

- Given a conjunction $C$ of linear (in)equalities, say whether it is satisfiable or not and, if it is so, provide a solution.
- Optionally, given $C$, say whether it trivially implies certain other predicates (*theory propagation*).

Furthermore, the algorithm should be organized so that it is easy to add and remove constraints in $C$.

Most efficient implementations of the simplex algorithm operate over floating-point numbers. However, because of floating-point roundoff errors, such implementations may produce incorrect results in some cases; this will not do inside a verification tool such as an SMT solver. SMT solvers thus use implementations of the simplex algorithm over the rational numbers; although such an implementation may be slower than one over floating-point numbers, it provides assurance that the results it obtains are sound and not subject to rounding errors.

[1]In operation research settings, this satisfiability phase is followed by another for optimization. This second phase is not commonly used inside SMT solvers.

Despite this weakness of floating-point implementations of the simplex algorithm, there have been several proposals to use them in SMT solvers to improve performance, of course with appropriate workarounds to ensure the soundness of results [7, 11, 2]. There exist indeed several ways to use a floating-point simplex implementation (whether one uses a primal or dual simplex, how to "correct" possibly unsound results...), and it was not so clear from experimental results whether one is better than another or even whether it is actually interesting to use a floating-point simplex.

As an example of a difficulty for evaluating simplex implementations, the main weakness of implementations using rational numbers is that the size of numerators and denominators may grow considerably for certain problems. While this depends of the problems being solved as well as the implementation of the simplex algorithm that may use techniques to reduce such tendency, such cases seldom seem to occur on some solvers handling real examples arising from software or hardware verification problems, as opposed to, say, random instances [11]. Furthermore, it is difficult to evaluate such systems outside of a full SMT solver, for the performance of a SMT solver not only depends on that of the theory solver (here, the simplex algorithm) but also on the size of the clauses output by the theory solver (smaller clauses are more efficient from the point of view of the SAT solver) and on other factors whose impact on overall performance is unclear.

In this article, we report on experiments of integration of a floating-point simplex solver (GLPK [10]) inside OpenSMT[2] [3].

## 2 Comparing the exact and floating-point simplex implementations

We shall distinguish two implementations of the simplex algorithm: the "floating-point simplex" and the "exact simplex", the former being implemented with floating-point numbers and the second with an exact rational representation, with arbitrary-precision numbers.

We investigate a combination of both implementations in order to provide exact results faster than a pure exact implementation. Our first step is to know how much faster the floating-point simplex can be compared to the exact simplex, and how often it is wrong.

### 2.1 Inside an SMT solver

The original implementation of OpenSMT calls an exact simplex as the theory solver for linear real arithmetic (LRA); it follows the same basic setup as Yices [5] or Z3. We modified OpenSMT so as to run both simplex implementations at the same time: its preexisting exact simplex and the floating-point simplex GLPK. As the floating-point simplex cannot handle strict inequalities directly, a strict inequality $\sum_i a_i x_i < b$ (the $a_i$ and $b$ are constants) is interpreted as $\sum_i a_i x_i \leq b - 10^{-9}$.

At this point, we are only interested in comparing how long both implementations take to verify whether a set of linear arithmetic constraints is satisfiable or not. All the extra work that is usually done by decision procedures (bookkeeping, communication with the SAT solver) is done by the original exact simplex and is discounted from this comparison.

The incremental nature of the decision procedure is preserved. The GLPK solver object is carried along the exact simplex state, and is not reinitialized as constraints are tightened or

[2]The version used was the latest public available with the source code, OpenSMT 1.0.1, dated from October 2010.

loosened. The GLPK solver tableau is created with all the linear forms $\sum_i a_i x_i$ present in the SMT formula; when constraints are tightened (i.e. $\sum_i a_i x_i \leq +\infty$, also known as "contraint not asserted", gets changed to $\sum_i a_i x_i \leq C$ where $C$ is a finite constant, or when $\sum_i a_i x_i \leq C$ gets changed to $\sum_i a_i x_i \leq C'$ where $C' < C$) or loosened (the converse of the above), only the GLPK bounds (but not the tableau) are changed (of course, subsequent checks may induce pivoting operations and thus tableau updates).

## 2.2 The first experiment

The benchmarks used in this experiment are those from the division QF_LRA (quantifier-free linear real arithmetic) in the SMT-LIB [1]. These benchmarks are used by the SMT community to check their solvers and compare their respective speeds. They come from a variety of sources: there are random, crafted or industrial examples.

The distribution of the times taken to run the 634 benchmarks is shown in Figure 1. The time limit was set to 2 minutes. With a total of over 38 millions distinct set of arithmetic constraints tested, the total accumulated time of the exact simplex was 5 h 26 m 55 s, while the accumulated time of the floating-point simplex was 4 h 44 m 45 s. The times include benchmarks that timed out.

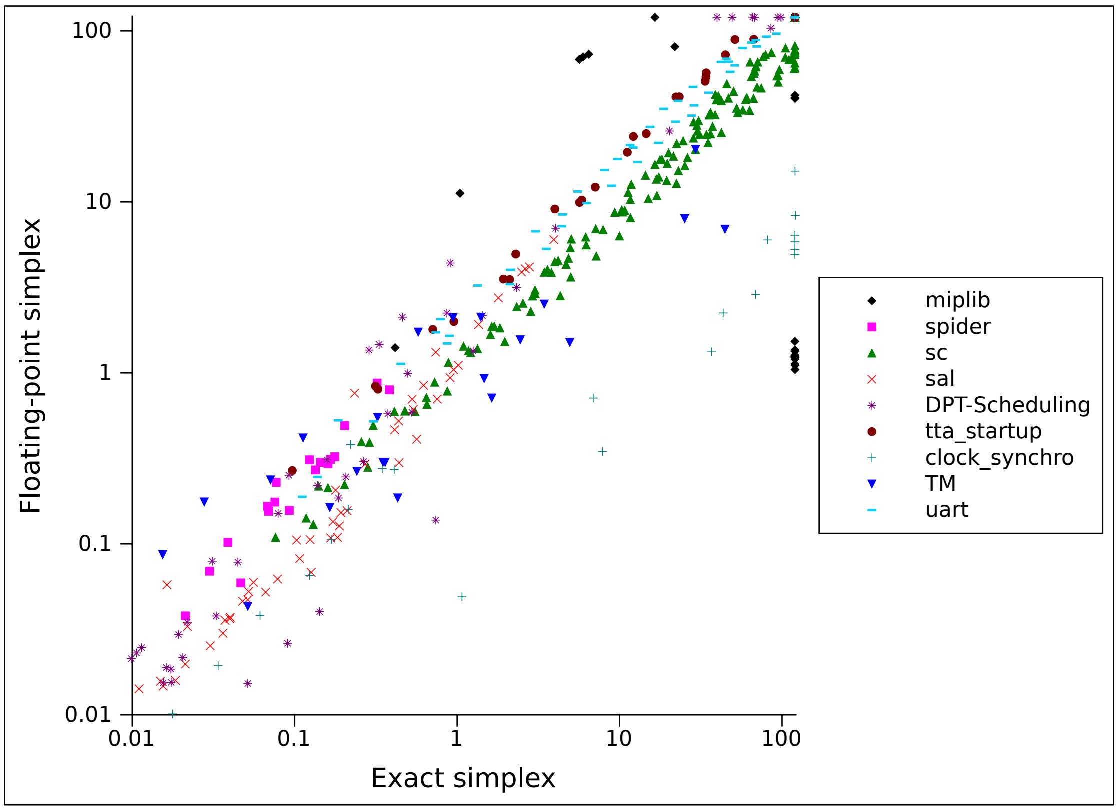


Figure 1: Exact simplex vs floating-point simplex in QF_LRA.

In the overall time comparison, the floating-point simplex is 14% faster than the exact simplex.

These results may even be discouraging: with the extra overhead of ensuring that the results from the floating-point simplex are sound, the final system is likely not to be faster than one using only exact computations. Yet, looking at the distribution, we can see that the floating-point simplex was much faster in a small number of cases, at least 10 times faster in approximately 3% of the benchmarks.

Apart from the timing distribution, the following points are important for understanding the feasibility of a floating-point simplex inside an SMT solver when solving practical problems:

**Soundness.** 11 (1.7%) of the 634 benchmarks tested had at least one incorrect check (meaning that the floating-point and exact simplex implementations yield different results). If we consider the total number of sets of arithmetic constraints checked, there were 852 (0.002%) incorrect results out of 38,566,969.

As expected, the floating-point simplex may give incorrect results. Even though they happen rarely, this should be taken in consideration when incorporating a floating-point simplex into an SMT solver; such situations must be detected and worked around. However, they are not statistically significant with respect to overall efficiency.

**Overflow.** OpenSMT and other SMT solvers like Z3 [4], implement arbitrary precision rational arithmetic in two layers: it first attempts computing the numerator and denominator inside machine words, and only if impossible, because of overflow, allocates extended precision numbers and branches into an extended precision library (OpenSMT[3] and early versions of Z3 use the GMP library [6][4]). GMP features very efficient procedures for computing on large integers, with both advanced high-level algorithms and highly optimized assembly code for basic operations. Yet, calling GMP for operations over small numbers incurs significant overhead — because of the cost of function calls to the library, and because the memory where GMP numbers are stored is allocated using `malloc()` and `free()`, thus incurring the cost of memory allocation and deallocation and breaking memory locality with respect to the processor cache.

It was recognized that in most cases from verification instances (as opposed to, say, random instances [11]), the solver never has to handle overflows and never calls the extended precision library. It is therefore common practice for tools such as decision procedures or static analyzers to adopt this layered approach, which, in the case of OpenSMT is implemented by overloaded C++ operators.

We shall now discuss our findings in this respect. 23 (3.6%) of the 643 benchmarks had at least one overflow. However, this proportion decreases even more when considering all the numbers manipulated (15,123,297,625), only 1,100,988 (0.007%) of them had overflow. All the benchmarks that had overflow are from the *clock_synchro* family.

One of the reasons that the floating-point simplex can be much faster than the exact simplex is when the latter starts operating on extended precision numbers due to overflow. Figure 1 shows that the problems from the family *clock_synchro* which had overflows are constantly solved faster by the floating-point simplex. Yet, as we can see that in the QF_LRA benchmarks, overflow is very rare and in this case, efficient rational number libraries can perform as well as floating-point computation.

[3]D. Monniaux implemented the C++ FastRationals layered arithmetic in OpenSMT. A similar library, ZArith, is available for OCaml from X. Leroy and A. Miné.

[4]L. de Moura, personal communication.

The problems of QF_LRA are sparse in general. Additionally, numbers presented in the formulas are usually small. Overflows happen much easier in dense problems when the frequent linear combination of the expression makes the coefficient of the variables to grow very fast [11].

**Size and running times.** It is also worth noticing the average running time per query of the simplex implementations: 0.44 ms for the floating-point simplex and 0.51 ms for the exact simplex. Both times are very low. That means that a decision procedure for linear arithmetic should be optimized to run several small/medium problems incrementally rather than a few large ones; this may explain earlier disappointing results when calling an industry-strength external linear programming package [7], which is optimized for solving very large instances from operation research.

The distribution of the running times seems randomly spread. Even though both implementations are variants on the simplex algorithm, they implement different heuristics (e.g. different pivoting strategies), thus explaining that the respective running times occasionally differ considerably.

With this first experiment, we conclude that the floating-point simplex is not always faster, but has the potential to solve a few extra problems. In the next section, we discuss a way of integrating the floating-point simplex into an SMT solver.

## 3 Integrating a floating-point simplex into an SMT solver

Our goal is to integrate a floating-point simplex into an SMT solver, keeping the exact simplex to maintain soundness. Finding a solution is generally more costly than just verifying it. The method we propose is to use the floating-point simplex to find a solution and use the exact simplex to check whether the solution is correct.

The set of solutions of the linear programming problem $AX \leq B$ ($X \in \mathbb{R}^n$, $B \in \mathbb{R}^m$, $A$ a $m \times n$ matrix) is a convex polyhedron. Let us now recall the workings of phase II of the simplex algorithm, the optimization phase. It starts from an initial feasible vertex (provided by phase I), and moves from a vertex to neighboring vertex in successive improvements of the objective; it stops if further improvement is impossible, on the optimal solution. There exist different strategies for choosing among possible next vertices, thus different running times. Some strategies may also lead to infinite cycling, which is prevented by using Bland's rule; a typical efficient strategy is steepest descent first, and Bland's rule after maximal number of pivoting iterations. Figure 2 illustrates the result of several iterations over a arbitrary simplex problem in a geometrical perspective, where the current solution is moving through the vertices.

In the case of verification problems, we are not interested in optimization, only in phase I, but the algorithm works similarly. The details of different implementations can be found e.g. in [5, 13, 12].

Once we have a point that represents a solution in the floating-point simplex, we go directly to this point in the exact simplex and verify it [11]. This is much faster than using the exact simplex to search for this solution point, since the number of pivot operations in this case is at most the number of variables, while it is at most exponential when searching. The verification is done by simply executing the exact simplex algorithm which does a linear scan over the variables only when a variable had a bound violated during the check call.

Verification proceeds as follows:

- if the point is a valid solution, the method will detect it immediately and will stop;

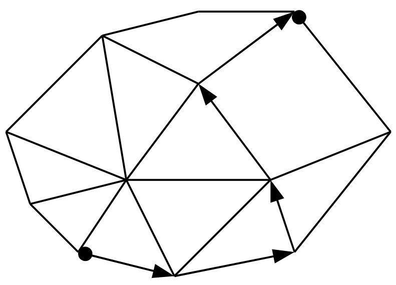


Figure 2: In the simplex algorithm, the point that represents the solution is moving through the neighbor vertices at each iteration.

- in the case the solution given by the floating-point simplex is incorrect, an uncommon case in practice, the exact simplex will find a correct solution starting from that point.

In other words, the verification procedure will use the solution found by the 'fast' floating-point simplex as the initial point of the 'slow' exact simplex, skipping a potentially expensive search every time the solution given is correct, as illustrated in Fig. 3.

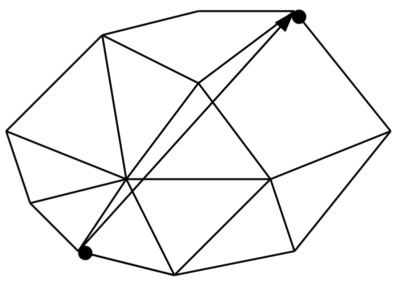


Figure 3: To verify a solution, we move directly to the point we are checking found by the floating-point simplex and 'continue' from there.

In order to direct the exact simplex to the same point reached by the floating-point simplex ("same" being defined as "the intersection of the same set of constraint planes", not "the same numerical value"), we reproduce inside it the same partition of variables into "basic" and "non-basic" and the same use of lower or upper bounds for variables as in the floating-point simplex [11], a purely combinatorial information. This avoids having to extract floating-point information and reinsert it into exact computations.

In any case, facts for the underlying SAT-solver are derived solely from the exact simplex: if the floating-point simplex answers a model, then this model is checked by exact computations, and if it answers that the system is unsatisfiable, then unsatisfiability is checked by the exact simplex, which also generates the conflicts (blocking clauses). In case of disagreement, the exact simplex has the last word.

## 4 The second experiment

In this section, we compare the combination implementation from Section 3 with the original exact implementation from OpenSMT: does our combination method actually improve the time the SMT solver takes to solve the problems? Figure 4 shows a comparison of their timings, for the 634 benchmarks with a time limit set to 4 minutes. Again, the benchmarks are the QF_LRA section of SMT-LIB.

The total accumulated time of the OpenSMT was 11h06m42s, while the accumulated time of the floating/exact OpenSMT was 13h26m35s. The times include benchmarks that timed out. *Unknown* results are considered time out. 37 problems were only solved by the original OpenSMT, while 8 other problems were only solved by the modified version. Of the problems that only the combination method could solve, 6 are from the *miplib* family and 2 of them are from the *tta_startup* family. The problems of the *miplib* family are a portion of the ones that in the first experiment were solved at least 10 times faster by the floating-point simplex.

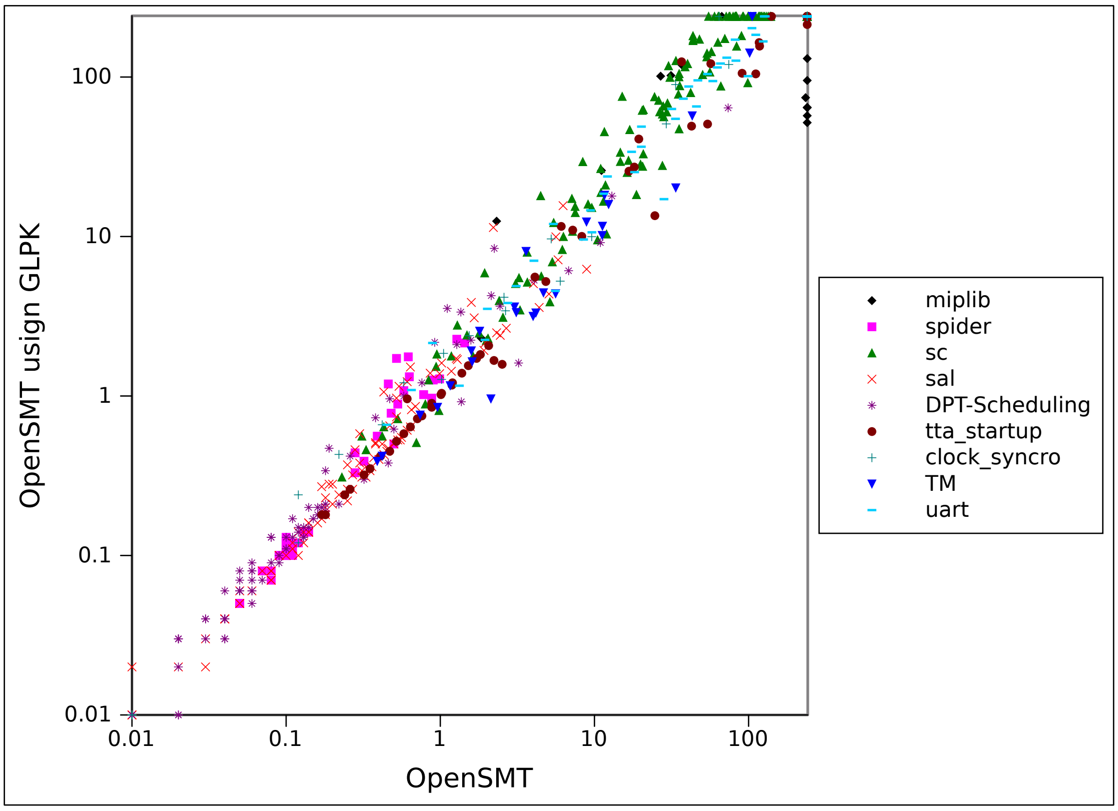


Figure 4: OpenSMT vs floating/exact OpenSMT in QF_LRA.

The result of this second experiment was rather unsurprising given the observations we obtained from the first one. The combined method was able to solve a few extra problems, but overall it was slower: its overhead is generally not compensated by the slightly higher speed of the floating-point simplex, though there exist some cases where it is markedly faster. We recapitulate the main points with a few extra observations here:

**Soundness** The number of incorrect checks done by GLPK is still very low in this experiment. GLPK and the exact simplex disagreed only on 0.0002% of the queries.

It is well-known that a naive floating-point implementation of the simplex algorithm, by direct implementation of the textbook description of variable selection and pivoting, will generally fail to solve larger problems because floating-point roundoff will cause it to enter "absurd" configurations, loop forever etc. GLPK, however, is not a naive implementation and generally avoids such pitfalls.

**Overflow.** One of the main reasons one could gain from the use of the floating-point simplex is the presence of overflow, which makes using floats (although unreliable) much faster than using an arbitrary precision library. However, we have seen in Section 2.2 that this does not happen often in QF_LRA and it continues very rare when the search is guided by GLPK. The presence of a few overflows in the family *clock_synchro* was not enough to make the combined method faster.

**Size and running time.** The average running time of the individual queries continues very low, being less than 1 ms. The high total running time comes from the very large number of linear programming feasibility queries to be solved per SMT problem, and not from the size of each individual query.

According to the GLPK FAQ [9], GLPK is "intended for solving large-scale linear programming [...] problems" and "is able to handle problems with up to 100,000 constraints". However, the average size of the problems in QF_LRA is 500 variables and 200 constraints. The advanced engineering implemented to solve large problems may not be the best to solve small ones. Additionally, the average number of checks is 60,000 per problem in these benchmarks, with consecutive checks usually changing very little. A good implementation of a simplex for SMT-solving is one able to solve fast thousands of relatively small, sparse and very similar set of constraints rather than just one very large.

Finally, there is not a clear reason of why the modified OpenSMT solves the few extra problems. The structure of these problems likely exhibits some characteristics that made them better solvable with the help of GLPK, yet the main reasons we could invoke for the viability of using a floating-point simplex, such as the presence of overflows and the large size of the problems, are not applicable in this case. Neither do these problems have overflows, nor are they very large in size (the average number of variables is 267 and the average number of constraints is 529).

## 5 Conclusion and possible future work

Despite the investigations of several authors, the merits of the integration of a floating-point simplex implementation into an SMT solver were unclear. We have therefore done a deep experimental comparison between the floating-point and exact simplex approaches on the QF_LRA benchmarks of SMT-LIB to shed some light on the issue.

On the one hand, the reason generally cited for possible better efficiency of the floating-point simplex — the use of expensive extended precision arithmetic in the exact simplex — applies to randomly generated benchmarks (on which pivoting quickly yields dense matrices with large numerators and denominators [11]), but not to benchmarks from SMT-LIB. On the other hand, the weakness generally cited about the floating-point simplex implementation — that floating-point roundoff errors could lead it to wrong answers that would be expensive to correct in order to obtain sound results — neither applies to SMT-LIB benchmarks.

Despite the generally slightly higher speed of the floating-point simplex compared to the exact simplex (our first experiment), the combination method using the floating-point simplex to guide the exact simplex to a solution (thus avoiding a potentially expensive search in exact representation) is generally slightly slower than the exact simplex. We have however identified some families of instances on which it is much faster, but it is unclear which characteristics of the problem produce this behavior and how they could be detected so as to choose which of the two implementations is more likely to be faster.

Our implementation currently reconstructs a starting point for the exact simplex by pivoting of the variables until their partition into basic and nonbasic matches that of the floating-point simplex. This boils down to moving from one base of a vector space to another by Gaussian elimination, and it is well-known that Gaussian elimination in exact rational arithmetic may generate, in its intermediate steps, dense matrices with large numerators and denominators even though the initial and final matrices are sparse and with small numerators and denominators.[5] For this reason, advanced methods for exactly solving systems of linear equations proceed by other means, such as performing Gaussian elimination modulo some prime numbers (if those numbers are not too large, all computations fit within machine words) and reconstructing the solution using the Chinese remainder theorem [14]. It seems possible to use such methods in lieu of pivoting to reconstruct the starting tableau of the exact simplex. It would perhaps be interesting to investigate such methods, though our second experiment indicates that the pivoting in the exact simplex seldom incurs overflows in practice and thus that it is unlikely that the gains from such advanced linear solving methods would be great (especially since they incur some overhead).

Theory propagation consists in deriving opportunistically facts that are implied by the current satisfiable configuration: for instance, if a line in the current simplex tableau implies immediately that $x \leq 20$ and there is a $x \leq 30$ atom $A$, then the solver can immediately assert $A$. This tends to be more efficient than waiting until the SAT solver branches on $A$ and, for instance, asserts $\neg A$ only to discover, through more pivoting, that it makes the system unsatisfiable. Currently, theory propagation is performed only by the underlying simplex implementation, and thus only in the rare case where the floating-point simplex answers "unsatisfiable" but the exact simplex disagrees: the results of the floating-point simplex are not trusted in this respect. Two improvements are possible so as to perform more theory propagation. The simpler is to use inequalities "implied" by the floating-point simplex tableau (we use quotes so as to stress the unsound character of this implication) as mere hints in the SAT solver: instead of being asserted as truths, they are suggested as appropriate polarities for the associated atoms. A more ambitious endeavour is to run pivoting steps in the exact simplex (or use linear algebra techniques) in order to exactly reconstruct the tableau lines used for theory propagation, which can then yield sound facts.

## References

[1] Clark Barrett, Aaron Stump, and Cesare Tinelli. The Satisfiability Modulo Theories Library (SMT-LIB), 2010. `http://www.SMT-LIB.org`.

[2] Frédéric Besson. On using an inexact floating-point LP solver for deciding linear arithmetic in an SMT solver. In *8th International Workshop on Satisfiability Modulo Theories*, Edinburgh, Royaume-Uni, 2010.

[5] Take for instance a square invertible matrix with small integer coefficients and echelonize it, finally obtaining an identity matrix: intermediate steps may still generate rather large numerators and denominators.

[3] Roberto Bruttomesso, Edgar Pek, Natasha Sharygina, and Aliaksei Tsitovich. The OpenSMT solver. In Javier Esparza and Rupak Majumdar, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, volume 6015 of *Lecture Notes in Computer Science*, pages 150–153. Springer Berlin / Heidelberg, 2010.

[4] Leonardo Mendonça de Moura and Nikolaj Bjørner. Z3: An efficient smt solver. In *TACAS*, pages 337–340, 2008.

[5] Bruno Dutertre and Leonardo Mendonça de Moura. A fast linear-arithmetic solver for DPLL(T). In Thomas Ball and Robert B. Jones, editors, *Computer Aided Verification, 18th International Conference, CAV 2006, Seattle, WA, USA, August 17-20, 2006, Proceedings*, volume 4144 of *Lecture Notes in Computer Science*, pages 81–94. Springer, 2006.

[6] Torbjörn Granlund et al. GNU multiple precision arithmetic library, 2000-2012. `http://gmplib.org/`.

[7] Germain Faure, Robert Nieuwenhuis, Albert Oliveras, and Enric Rodríguez-Carbonell. SAT modulo the theory of linear arithmetic: exact, inexact and commercial solvers. In *Proceedings of the 11th international conference on Theory and applications of satisfiability testing*, SAT'08, page 77–90, Berlin, Heidelberg, 2008. Springer-Verlag.

[8] Harald Ganzinger, George Hagen, Robert Nieuwenhuis, Albert Oliveras, and Cesare Tinelli. DPLL(T): Fast decision procedures. In *Computer-aided verification (CAV)*, pages 175–188. Springer, 2004.

[9] Harley Mackenzie. Frequently asked questions about the gnu linear programming kit, 2004.

[10] Andrew Makhorin. GNU linear programming kit, 2000-2008. `http://www.gnu.org/software/glpk/`.

[11] David Monniaux. On using floating-point computations to help an exact linear arithmetic decision procedure. In *Computer-aided verification (CAV)*, number 5643 in LNCS, pages 570–583. Springer, 2009.

[12] Diego Caminha Barbosa De Oliveira. *Fragments de l'arithmétique dans une combinaison de procédures de décision.* PhD thesis, Université Nancy II, March 14 2011.

[13] Harald Rueß and Natarajan Shankar. Solving linear arithmetic constraints. Technical Report SRI-CSL-04-01, SRI International, 2004.

[14] William Stein. *Modular forms, a computational approach*, volume 79 of *Graduate studies in mathematics.* American Mathematical Society, 2007.

# Authorization Enforcement in Workflows: Maintaining Realizability Via Automated Reasoning

Jason Crampton[1], Michael Huth[2] and Jim Huan-Pu Kuo[2]

[1] Information Security Group, Royal Holloway
University of London
jason.crampton@rhul.ac.uk
[2] Department of Computing
Imperial College London
m.huth,jimhkuo@imperial.ac.uk

**Abstract**

We investigate automated reasoning techniques as a means of supporting authorization enforcement functions of security-aware workflow management systems. The aim of such support is that one may statically or dynamically guarantee the realizability of a workflow instance given the security constraints of the underlying workflow specification.

We develop two such automated reasoning methods and experimentally evaluate their suitability for giving such support. One method uses a propositional encoding of realizability implemented through binary decision diagrams, another method uses a linear-time temporal logic encoding implemented via bounded model checking.

We chose these particular methods and implementations since they render representations that, at least in principle, capture many potential solutions so that dynamic guarantees of realizability can be made through efficient queries on these representations. Preliminary experimental results identify issues of scalability and of balancing flexibility in task allocation with complexity of computing such allocations.

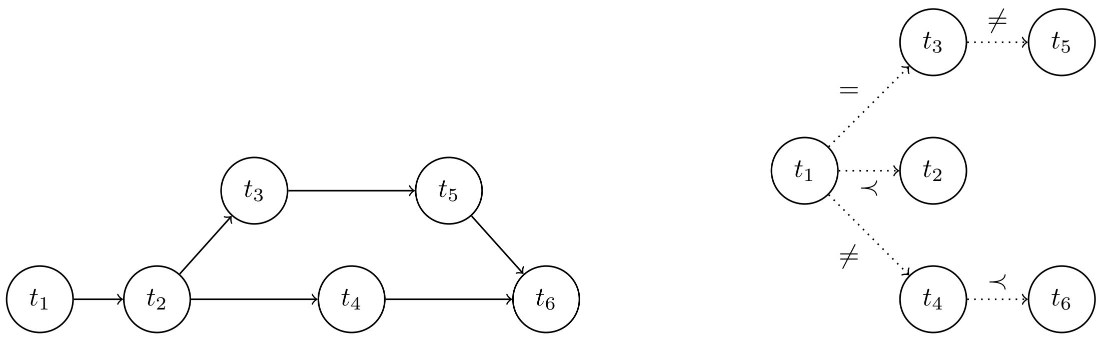
## 1 Introduction

It is increasingly common for organizations to computerize their business and management processes. The co-ordination of the tasks or steps that comprise a computerized business process is managed by workflow management systems or business process management systems.

A workflow typically specifies the tasks that comprise a business process and the order in which those tasks should be performed. Moreover, it is often the case that some form of access control should be applied to the execution of tasks. Hence, most workflow management systems may implement security controls that enforce authorization rules and business rules, in order to comply with statutory requirements or best practice. It is such "security-aware" workflows that will be the focus of this paper. Among the most useful security controls are:

- *user/task authorization* constraints, which specify which users may, in principle, execute what tasks;
- *binding of duty* (BoD) constraints, which require that certain tasks be executed by the same user in any given workflow instance;
- *separation of duty* (SoD) constraints, which require that certain tasks be executed by different users in any given instance of the workflow.

An illustrative example of a constrained workflow for purchase order processing is shown in Fig. 1. The purchase order is created and approved (and then dispatched to the supplier). The supplier will present an invoice, which is processed by the create payment task. When the supplier delivers the ordered goods, a goods received note must be signed and countersigned; only then may the payment be approved. A workflow specification need not be linear: the processing of the goods received note and of the invoice can occur in parallel, for example.



(a) Task ordering (b) Constraints

| | |
|---|---|
| $t_1$ | create purchase order |
| $t_2$ | approve purchase order |
| $t_3$ | sign goods received note |
| $t_4$ | create payment |
| $t_5$ | countersign goods received note |
| $t_6$ | approve payment |
| $\neq$ | users performing the tasks must be different |
| $=$ | users performing the tasks must be the same |
| $\prec$ | user performing the second task must be senior to the user performing the first |

(c) Figure legend

Figure 1: A simple constrained workflow for purchase order processing

In addition to constraining the order in which tasks are to be performed, some business rules are specified to prevent fraudulent use of this workflow. These rules take the form of constraints on users that can perform pairs of tasks in the workflow: for example, that the same user must not sign and countersign the goods received note.

The aggregate effect of such constraints may make it impossible to find an allocation of tasks to users and satisfy all the constraints. In other words, it may be that a workflow is rendered unrealizable by the inclusion of security controls. Hence, it is important to be able to determine whether a workflow specification can be realized.

There are different ways in which a workflow management system might choose to allocate tasks to users. These "execution models" give rise to different realizability problems but share the need to guarantee the continued realizability of a workflow instance. Hence, efficient decision procedures for workflow realizability are needed.

In this paper, we consider methods by which an authorization enforcement engine for workflow management systems might be designed. By construction, these methods should maintain the realizability of a workflow instance during its execution. We describe two such methods – a decision procedure and a search procedure – that can be called by such an engine. In particular, we explain how we can use binary decision diagrams (BDDs) to build a decision procedure and bounded model checking to build a search procedure. We then describe our experimental work that compares the relative merits of these two methods.

**Outline of paper.** In Section 2, we present technical background of security-aware workflow systems. Two methods for supporting authorization enforcement functions for workflow instances, and their encodings through automated reasoning methods, feature in Section 3. Preliminary experimental data for implementations of these encodings are reported in Section 4. A brief discussion and our conclusions make up Section 5.

## 2 Preliminaries

We recall the definition of a constrained workflow authorization schema [2], which has formed the basis for a number of papers on workflow realizability, for example [1, 6].

**Definition 1.** *A* constrained workflow authorization schema $\mathcal{AS}$*, is a tuple* $(T, \leq, U, A, C)$ *where*

- $T$ *is a set of tasks and* $(T, \leq)$ *is a partial order,*
- $U$ *is a set of users and* $A \subseteq T \times U$ *an authorization relation,*
- $C$ *is a finite set of* entailment constraints*, tuples of form* $(D, t \to t', \rho)$ *where* $D \subseteq U$*,* $t, t' \in T$ *and* $\rho \subseteq U \times U$*.*

The order $t \leq t'$ models that either $t$ equals $t'$ or task $t$ has to be completed before task $t'$ begins. Thus $\leq$ models *temporal* constraints on task execution. The authorization $(t, u)$ in $A$ models that user $u$ is, at least in principle, authorized to execute task $t$. As we will see, the authorization enforcement engine may not allow an authorized user to execute a task because doing so would render the workflow instance unrealizable. An entailment constraint $(D, t \to t', \rho)$ models that if user $u$ executes task $t$ and $u$ is from target set $D$, then the user $u'$ who executes task $t'$ (and who need not be from set $D$) must be related to $u$ in the manner specified by $\rho$, i.e. $(u, u')$ must be in $\rho$. For example, when $D$ equals $U$ the entailment constraint models BoD when $\rho$ equals $=$, and it models SoD when $\rho$ equals $\neq$. Note that entailment constraints $(D, t \to t', \rho)$, in and of themselves, do not impose any temporal order on the relative occurrence of $t$ and $t'$.

### 2.1 Workflow Realizability

It is apparent that the existence of an authorization policy and entailment constraints may mean that there is no possible allocation of users to tasks. Hence, an important, practical question is whether a workflow authorization schema $\mathcal{AS}$ is *realizable* (also known as *satisfiable* in the literature). For our kind of schema, realizability means that one can allocate all tasks $t$ in $T$ to users in $U$ such that all schema constraints (temporal order, authorization, and entailment) are satisfied. We now define this notion formally.

**Definition 2.** *Let* $\mathcal{AS}$ *denote a constrained authorized workflow schema as above. Then* $\mathcal{AS}$ *is* realizable *if there exists a total function* $\alpha\colon T \to U$ *such that*

- $(t, \alpha(t))$ *is in* $A$ *for all t in* $T$*,*
- *for all* $(D, t \to t', \rho)$ *in* $C$*, if* $\alpha(t)$ *is in* $D$*, then* $(\alpha(t), \alpha(t'))$ *is in* $\rho$*.*

In other words, a workflow schema is realizable if there exists an allocation of users to tasks such that each user is authorized and all entailment constraints are satisfied. The reason that $\alpha$ suffices as a solution for realizability of $\mathcal{AS}$ is that our schema allows for the decoupling of temporal orderings from other constraints, and partial orders are always realizable ("linearizable"). We write $\mathsf{Sol}(\mathcal{AS})$ to denote the set of all functions $\alpha$ that realize the workflow $\mathcal{AS}$; this is the *solution space* of $\mathcal{AS}$, which may be empty.

## 2.2 Executing Workflows

A workflow management system (WfMS) is responsible for instantiating workflow schemas. The WfMS is also responsible for managing the execution of the tasks in a *workflow instance*. In particular, the WfMS will maintain a pool of *ready tasks*: the set of ready tasks in a workflow instance is the set of minimal tasks (with respect to the ordering on $T$) that have not yet been completed. Using the example in Fig. 1, the ready tasks once $t_2$ has been performed, for example, are $t_3$ and $t_4$; if $t_4$ is then performed, the set of ready tasks will be $\{t_3, t_6\}$.

In a workflow instance, the user/task allocation may be done in different ways [4].

- The WfMS creates a task list to which authorized users are allocated when a workflow is instantiated.
- The WfMS allocates authorized users to only those tasks that are presently ready.
- The WfMS maintains a pool of ready tasks from which users select tasks to execute.

We refer to these execution models as *static task allocation*, *dynamic task allocation* and *task selection*, respectively.

## 3 Two automated reasoning methods for authorization enforcement

The WfMS must incorporate a module, which we call the *authorization enforcement function* (AEF), that can ensure that

- a workflow instance is completed by users who are authorized for the respective tasks they perform,
- all constraints are satisfied, and
- the workflow instance completes.

The first of these responsibilities is a standard one for access-control functions and we will assume that it can be performed efficiently. The interesting question is how to implement the remaining functionality of the AEF.

The nature of the AEF will be determined by the execution model. In particular, there is an important distinction between static task allocation and the other two execution models. With static task allocation, the AEF computes a single mapping $\alpha$ of users to tasks, meaning that a single check for realizability is performed. Precomputing such a mapping maintains realizability by construction but does not allow for the modification of user-task bindings (which may perhaps be required for load-balancing, for example).

In contrast, no "up-front" computation of $\alpha$ is performed for dynamic task allocation and task selection. Instead, the AEF must perform a series of realizability checks on modified versions of the workflow schema $\mathcal{AS}$. Once a task $t$ has been performed by user $u$, then we transform the authorization relation of $\mathcal{AS}$ so that the only authorized user for $t$ is $u$. We then determine the realizability of the modified schema. Henceforth, we only consider the task selection execution model, since the design of our AEF can be readily modified to accommodate the dynamic task allocation model.

The AEF traps all user requests to execute tasks, makes a decision on requests, and enforces that decision. We may model the AEF mathematically as a function of type

$$\mathsf{AEF}\colon \mathsf{accessRequest} \times \mathsf{state} \to \mathsf{decision} \times \mathsf{state} \qquad (1)$$

where accessRequest is the set of request events the AEF has to process (here, access requests of form $(t, u)$ in $T \times U$), state is an internal state that AEF maintains, and decision is the set of access-control decisions that AEF can make (here either $grant$ or $deny$). In other words, the AEF may inspect its internal state when making a decision on the current access request, and it may possibly alter its state as a result of that decision.

In the remainder of this paper we explore how such an AEF can be designed so that it may make access-control decisions that maintain the *realizability* of a constrained authorized workflow schema $\mathcal{AS}$. Concretely, we will discuss how automated reasoning tools may be used to give an AEF the ability to maintain realizability if at all possible. In particular, the state of an AEF will need to contain information that supports the maintenance of realizability of the workflow.

A key challenge in using automated reasoning tools is here that they should not incur a computational cost that would lead to unacceptable delays of access control decisions. It is this design constraint that will suggest to us methods that may precompute a representation of a large portion of the solution set, so that dynamic requests can be decided by an efficient inspection (and perhaps adjustment) of that representation. A formula of propositional logic, for example, may not be a suitable representation: although it can capture the entire solution space, querying it may involve a full SAT check that may simply take too long to complete in this application context.

## 3.1 Constructing an AEF with a Decision Procedure

We now describe how to construct an AEF from any decision procedure for workflow realizability so that this AEF maintains realizability whenever it grants access requests. Let $\mathcal{AS}$ denote a constrained authorized workflow schema as above. We assume the state $\sigma$ maintained by the AEF to be a list of pairs of form $((u, t), d)$, where $(u, t)$ is a request and $d$ the decision the AEF made on request $(u, t)$. In particular, there is at most one pair in $\sigma$ with first component $(u, t)$ – we assume that repeated tasks are distinct in $\mathcal{AS}$ – and we can extract from $\sigma$ all requests to execute tasks that have been granted.

Let $\sigma_{\text{complete}}$ be the set of tasks in $T$ such that there exists an entry in $\sigma$ of the form $((u, t), grant)$. For $t \in \sigma_{\text{complete}}$, we write $\sigma(t)$ to denote the user that was granted permission to execute $t$. [1] We write $\sigma_{\text{incomplete}}$ for $T \setminus \sigma_{\text{complete}}$. We define

$$\begin{aligned}\mathcal{AS}[\sigma] &\stackrel{\text{def}}{=} (T, \leq, U, A[\sigma], C), \text{ where}\\ A[\sigma] &\stackrel{\text{def}}{=} (A \cap (\sigma_{\text{incomplete}} \times U)) \cup \{(t, \sigma(t))\colon t \in \sigma_{\text{complete}}\}\,.\end{aligned}$$

In other words, for all $t$ in $\sigma_{\text{complete}}$, we replace all instances of $(t, u)$ occurring in $A$ with the sole entry $(t, \sigma(t))$, and leave all instances of $t$ in $\sigma_{\text{incomplete}}$ untouched in $A$.

Having established these concepts and notation, we can now sketch one possible approach to maintaining the realizability of $\mathcal{AS}$ through an AEF that is consistent with the type declared in (1). The

[1] Although $\sigma(t)$ might be a set of users, we assume that tasks are unique and so repeated tasks are differentiated through their instances.

```
(decision,state) AEF-DP(schema $\mathcal{AS}$, state $\sigma$, accReq $(t,u)$)
{
  if ($(t,u) \in A$ && isRealizable($\mathcal{AS}[\sigma \mid ((u,t), grant)]$))
    { return ($grant$, $\sigma \mid ((u,t), grant)$); }
  else
    { return ($deny$, $\sigma \mid ((u,t), deny)$); }
}
```

Figure 2: An AEF incorporating a decision procedure for workflow realizability

pseudocode for AEF-DP is depicted in Fig. 2. The decision procedure isRealizable takes a workflow schema $\mathcal{AS}$ as input and returns true if and only if $\mathsf{Sol}(\mathcal{AS})$ is non-empty (i.e. returns true if and only if $\mathcal{AS}$ is realizable).

We now describe the behavior of AEF-DP, where we write $\sigma \mid x$ for the state that appends to list $\sigma$ the item $x$ of appropriate type. A request $(t,u)$ is denied if either $(t,u)$ is not in the authorization relation $A$ of $\mathcal{AS}$,[2] or if isRealizable, when supplied with input $\mathcal{AS}[\sigma \mid ((u,t,), grant)]$, returns false – meaning that there is no function $\alpha$ in $\mathsf{Sol}(\mathcal{AS})$ that maps $u$ to $t$ and $\sigma(t')$ to $t'$ for all $t'$ that have been executed – as granting it would make the remaining workflow unrealizable. Otherwise, the request is granted. In any event, $\sigma$ is updated to reflect the decision made.

The crucial invariant that this AEF guarantees is that

> "**Invariant:** All grants of access requests mean that the workflow $\mathcal{AS}$ is realizable in the updated state."

One possible drawback of this approach is that the decision procedure isRealizable needs to be called each time an access-control request is made. As already discussed, one limiting factor will certainly be the space and time requirements for such a decision procedure. Therefore, we will now explore whether automated reasoning tools can be devised that fare better in this regard.

## 3.2 Constructing an AEF with Solution Sets

The method AEF-DP maintains the realizability of a workflow, but makes no use of "witness" information for such realizability. One price we pay for this is that we need to recompute realizability decisions each time a request is processed by AEF-DP.

Hence, we now discuss an alternative approach that uses a *search procedure* to compute a (representation of a) subset of $\mathsf{Sol}(\mathcal{AS})$. The procedure relies on an abstraction of $\mathsf{Sol}(\mathcal{AS})$. More precisely, it computes two partitions

$$T = \bigcup_{i=0}^{n} T_i \qquad \text{and} \qquad U \supseteq \tilde{U} = \bigcup_{i=0}^{n} U_i \tag{2}$$

where *all functions* $\alpha\colon T \to U$ such that $(t_i, \alpha(t_i))$ belongs to $T_i \times U_i$ for all $0 \leq i \leq n$ belong to $\mathsf{Sol}(\mathcal{AS})$. The intuition here is that we may assign to any task in $T_i$ any user in $U_i$, and that we can be sure that this will not interfere with any constraints within $(T_i, U_i)$ nor across any of the set-valued task/user pairs $(T_j, U_j)$. Note that (2) partitions task set $T$ but only partitions a subset $\tilde{U}$ of users of $U$ that will be allocated to tasks in that workflow instance. In effect, this is an under-approximation of

[2]In the interests of brevity, our pseudocode does not include sanity checks such as ensuring that the requested task has not already been performed. As already mentioned, we also assume that multiple occurrences of the same task are distinguishable in the schema.

```
(decision,state) AEF-SS(schema $\mathcal{AS}$, state $(\Sigma,\sigma)$, accReq $(t,u)$)
{
  if (there exists $(T,U) \in \Sigma$ such that $t \in T$ and $u \in U$)
    { return $(grant, (\Sigma,\ \sigma \mid ((u,t), grant)))$ }
  else
    { return $(deny, (\Sigma,\ \sigma \mid ((u,t), deny)))$ }
}
```

Figure 3: Constructing an AEF using a static prepartitioning of solutions

$\mathsf{Sol}(\mathcal{AS})$ as it represents a subset of that space of solutions and does not represent functions that aren't solutions.

Given a method `getAbs` for computing such partitions, we can write a second AEF, `AEF-SS`, pseudocode for which is shown in Fig. 3. This approach assumes the state is an ordered pair $(\Sigma,\sigma)$, where $\Sigma$ is some representation of two partitions as in (2) computed using `getAbs`, and (as before) $\sigma$ is a list that records which requests have been processed with what decisions. In particular, $\sigma$ records which tasks have already been allocated to which users by `AEF-SS`.

Given a request $(u,t)$, `AEF-SS` inspects whether $u$ and $t$ belong to the same task/user pair computed by `getAbs`, i.e. whether there is some $i$ so that $u$ is in $U_i$ and $t$ in $T_i$. If so, access is granted; otherwise it is denied. In particular, the $\Sigma$ part of the state never changes and `AEF-SS` never has to recompute realizability information. However, it is possible that `AEF-SS` may deny a request that would not prevent the completion of a workflow instance.

## 4 Implementation and Evaluation

In this section, we describe how the procedures `isRealizable` and `getAbs` can be constructed. For the `isRealizable` procedure, we encode the realizability problem as an instance of SAT for propositional logic (PL); we do this since we want to test whether BDDs might serve as an effective representation of the solution space. For the procedure `getAbs` we capture this also as a SAT instance but in linear-time temporal logic (LTL), as done in [3]. We use LTL and a bounded model checker here as we can instrument the LTL encoding so that it precomputes partitions as in (2) that can be used as a basis for `AEF-SS`. We also report on experimental work that tests the performance of our methods and these encodings when applied to synthetic (randomly generated) workflow schemas.

### 4.1 Procedure `isRealizable`

Formula $\eta_{\mathcal{AS}}$ encodes the realizability problem for $\mathcal{AS}$ as an instance of SAT for PL, where models of $\eta_{\mathcal{AS}}$ correspond to elements of $\mathsf{Sol}(\mathcal{AS})$ and vice versa. This encoding is shown in Fig. 4. Its set of propositional variables is

$$\{x_{(t,u)} \mid (t,u) \in A\}$$

where we define the sets of users $U_t$ and $u.\rho$ as

$$U_t = \{u \in U \mid (t,u) \in A\} \tag{3}$$

$$u.\rho = \{u' \in U \mid (u,u') \in \rho\} \tag{4}$$

This encoding is sound and complete since we can show that $\eta_{\mathcal{AS}}$ is satisfiable if and only if $\mathsf{Sol}(\mathcal{AS})$ is non-empty, i.e. $\mathcal{AS}$ is realizable. The intuition behind the encoding is that $t$ *may be allocated* to $u$ if $x_{(t,u)}$ is true, and that $t$ *must not be allocated* to $u$ if $x_{(t,u)}$ is false.

$$\begin{aligned}\eta_{bind} &= \bigwedge_{t\in T}\bigvee_{u\in U_t} x_{(t,u)} \\ \eta_C &= \bigwedge_{(D,t\to t',\rho)\in C} \eta_{(D,t\to t',\rho)} \\ \eta_{(D,t\to t',\rho)} &= \bigwedge_{u\in D\cap U_t}\left(x_{(t,u)} \to \bigwedge_{u'\in U_{t'}\setminus u.\rho} \neg x_{(t',u')}\right)\end{aligned} \qquad (5)$$

Figure 4: Encoding $\eta_{\mathcal{AS}} \overset{\mathrm{def}}{=} \eta_{bind} \wedge \eta_C$: workflow realizability as instance of SAT for PL

Specifically, formula $\eta_{bind}$ specifies that all tasks may be allocated to some user – a necessary requirement for realizability. Formula $\eta_C$ simply stipulates that all formulas $\eta_{(D,t\to t',\rho)}$ that encode entailment constraints must be true. And such a formula $\eta_{(D,t\to t',\rho)}$ states that if a user $u$ from set $D$ may be allocated to task $t$, then all users $u'$ that are authorized to execute task $t'$ but are not in a relationship to $u$ via $\rho$ are such that they must not be allocated to $t'$. Note that "It is not the case that $u'$ may allocate task $t'$ " is equivalent to "It is the case that $u'$ must not be allocated to task $t'$ ". The need for this indirection is that the variables do not represent the modality "must be allocated".

A Boolean function (and so $\eta_{\mathcal{AS}}$ as well) can be represented as a binary decision diagram (BDD), a DAG-type data structure that eliminates redundancies in binary decision trees; and this representation is unique for a fixed order of variables in the BDD. The main reason why we are interested in BDDs here is that one can efficiently compute specializations of BDDs (in which the truth values of some variables are fixed) in order to decide the realizability of a workflow in an updated state. Thus we could implement non-initial calls to `isRealizable` efficiently relative to the complexity of computing the "initial" BDD from $\eta_{\mathcal{AS}}$. Our experiments therefore focus on the latter computation.

Given $\eta_{\mathcal{AS}}$, we first synthesize from it a BDD $B_{\mathcal{AS}}$ (using a standard BDD library `JavaBDD`, which relies on the CUDD implementation in `C`, and its default variable ordering) and then check (in constant time) whether that BDD is equal to the canonical BDD that contains only leaf 0 (and so represents "unsatisfiable"). If and when this BDD has been built, we can implement the call to `isRealizable` in Figure 2 by simply computing the specialization of this BDD that eliminates one variable.

## 4.2 Procedure `getAbs`

Our implementation of `getAbs` is through a reduction of realizability of $\mathcal{AS}$ to SAT for the NP-complete fragment [5] $\mathsf{LTL}(\mathsf{F})$ of LTL. We quickly review the syntax and semantics of $\mathsf{LTL}(\mathsf{F})$: Given a finite set $\mathsf{AP}$ of atomic propositions (this is $T \cup U$ here), the propositional temporal logic $\mathsf{LTL}(\mathsf{F})$ is generated by the following grammar:

$$\phi ::= p \mid \neg\phi \mid \phi\wedge\phi \mid \mathsf{F}\,\phi$$

where $p$ is from $\mathsf{AP}$ and $\mathsf{F}$ is the temporal connective "Future" such that $\mathsf{F}\,p$ states that $p$ will be true at some point in the future.

A *model* of a formula $\phi$ is an infinite sequence of states $\pi = s_0 s_1 \ldots$, where each $s_i$ is a subset of $\mathsf{AP}$. We write $\pi \models \phi$ if $\pi$ is a model for $\phi$. We say that a formula $\phi$ is *satisfiable* if and only if it has a model. We write $\pi^i$ to denote the infinite suffix $s_i s_{i+1} \ldots$ of $\pi$. The formal semantics of formulas is then given in Figure 5.

$$\pi \models p \quad \text{iff} \quad p \in s_0 \qquad\qquad \pi \models \neg\phi \quad \text{iff} \quad \text{not } \pi \models \phi$$
$$\pi \models \phi_1 \wedge \phi_2 \quad \text{iff} \quad (\pi \models \phi_1 \text{ and } \pi \models \phi_2) \qquad\qquad \pi \models \mathsf{F}\,\phi \quad \text{iff} \quad \text{there is } i \geq 0\colon \pi^i \models \phi$$

Figure 5: Formal semantics of temporal logic $\mathsf{LTL}(\mathsf{F})$ over models $\pi = s_0 s_1 \dots$

$$\begin{aligned}
\delta_{\mathsf{F}T} &\overset{\text{def}}{=} \bigwedge_{t \in T} \mathsf{F}\,t \\
\delta_{\mathsf{G}U} &\overset{\text{def}}{=} \mathsf{G}(\bigvee_{u \in U} u) \\
\delta_A &\overset{\text{def}}{=} \bigwedge_{t \in T} \mathsf{G}\Big(t \to \neg(\bigvee_{(t,u) \notin A} u)\Big) \\
\delta_C &\overset{\text{def}}{=} \bigwedge_{(D,t \to t',\rho) \in C} \delta_{(D,t \to t',\rho)} \\
\delta_{(D,t \to t',\rho)} &\overset{\text{def}}{=} \bigwedge_{u \in D} \big(\mathsf{F}\,(t \wedge u)\big) \to \mathsf{G}\Big(t' \to \neg(\bigvee_{(u,u') \notin \rho} u')\Big) \\
\delta_{\mathsf{F}U} &\overset{\text{def}}{=} \bigwedge_{u \in U} \mathsf{F}\,u
\end{aligned}$$

Figure 6: Encoding $\delta_{\mathcal{AS}} \overset{\text{def}}{=} \delta_{\mathsf{F}T} \wedge \delta_{\mathsf{G}U} \wedge \delta_A \wedge \delta_C \wedge \delta_{\mathsf{F}U}$ of [3]: workflow realizability in $\mathsf{LTL}(\mathsf{F})$

We use the usual abbreviations for disjunction ($\vee$), implication ($\to$), logical equivalence ($\leftrightarrow$) and the "Global" temporal connective $\mathsf{G}\,\phi$, which stands for $\neg\mathsf{F}\,\neg\phi$ (the informal interpretation being "always $\phi$").

The realizability of $\mathcal{AS}$ we encode as a SAT instance for $\mathsf{LTL}(\mathsf{F})$ formula $\delta_{\mathcal{AS}}$ shown in Fig. 6. Its satisfiability is decided using the model checker NuSMV on a fully connected model, formula $\neg\delta_{\mathcal{AS}}$, and in an incremental bounded model-checking mode. The set of variables for this encoding is the disjoint union $T \cup U$. The interpretation of a variable $t$ (respectively, $u$) being true in state $s_i$ is that it is in set $T_i$ (respectively, $U_i$) of the constructed partition. Thus the $\delta_{\mathcal{AS}}$ encoding allows the possibility that several tasks and users may hold in a state.[3]

If the model checker returns a "counterexample", a finite trace of states $s_0 s_1 \dots s_n$ that represents a "lasso" path $\pi$ that makes $\delta_{\mathcal{AS}}$ true, then we can derive a partition

$$T_i = s_i \cap T \qquad\qquad U_i = s_i \cap U \tag{6}$$

and show (see [3]) that all $\alpha$ that allocate tasks consistent with all $(T_i, U_i)$ pairings are in $\mathsf{Sol}(\mathcal{AS})$.

We now discuss this encoding in greater detail. Formula $\delta_{\mathsf{F}T}$ demands that all tasks have to be true at some state, whereas $\delta_{\mathsf{G}U}$ ensures that all states make some user(s) true. Formula $\delta_A$ indirectly captures the authorization relation $A$: for all tasks $t$, if $t$ is true at some state then no users that are un-authorized to execute $t$ can be true at that state. The reason for this indirect encoding is that we need to rule out that user and task groupings at a state violate any constraints, and that we cannot control or predict these groupings.

[3]An encoding of workflow realizability in $\mathsf{LTL}(\mathsf{F})$ in which we insist that a single user and task are executed in all states has poor model checking results [3].

Formula $\delta_C$ states that all entailment contraints have to be met. And formula $\delta_{(D,t\to t',\rho)}$ captures such an entailment constraint. If a user $u$ from set $U$ is grouped with task $t$ at some state, then at all states that make $t'$ true there are no users $u'$ true there which are not in relationship $\rho$ with $u$. Again, this indirection is needed in order for the model checker to discover such groupings of users and tasks at states.

Intuitively, it is desirable to have states $s_i$ in which there are as many tasks and users as possible, as this gives us more flexibility when dealing with access requests. Similarly, we want this search procedure to have the tendency of accommodating, and so possibly allocating, as many users and tasks in the sets $T_i$ and $U_i$. This tendency is actively encouraged through the conjunct $\delta_{\mathsf{F}U}$ in encoding $\delta_{\mathcal{AS}}$. The intuition behind the inclusion of this conjunct is that we use a bounded model checker that will find the shortest possible "lasso" trace that represents a model of the formula. So the model checker will indeed try to pack as many users into states as possible to capture a solution, and will put all those users that were not needed for the solution into a "junk" state, and only into one such junk state. We found this to be beneficial when compared to an encoding that does not include this conjunct [3].

## 4.3 Experimental data

We now discuss our experimental results, which compare the performance of the BDD approach ($\eta_{\mathcal{AS}}$ for `isRealizable` in `AEF-DP`) to a bounded model-checking approach ($\delta_{\mathcal{AS}}$ for `getAbs` in `AEF-SS`) on randomly generated workflows $\mathcal{AS}$ (be they realizable or not). These experiments were conducted on the same Ubuntu Linux machine with Intel$^{®}$ Core$^{\mathrm{TM}}$ 2 Duo Processor at 2.8 Gigahertz and 4 Gigabytes of RAM.

We now describe the set of configurations for the workflow schemas $\mathcal{AS}$ used in our experiments. Each $\mathcal{AS}$ was generated according to three parameters: the number of users, the *authorization density*, and the *constraint density*. In each configuration the number of tasks was equal to the number of users, taking values $10, 20, \ldots, 140, 150$. Authorization density is defined to be the ratio of $|A|$ to the product of $|U|$ and $|T|$, where the latter represents the maximum possible cardinality of $A$. The authorization densities we considered in our experiments are 0.1, 0.5, and 1.0, therefore ranging from a rather sparse authorization policy through to one in which all users are authorized to perform all tasks. The constraint density is defined to be the ratio of $|C|$ to $|U|$. We let this value range over 0.05, 0.10, and 0.20. The reason for choosing these lower values, but still having a good spread within that low range, is that higher values of the constraint density tend to produce only unrealizable *ramdomly* generated $\mathcal{AS}$ and we are interested in realizable $\mathcal{AS}$ as we mean to support such realizability as an invariant in an AES.

We present our results in graphical form in Figures 7 to 9. Each figure shows results for a different authorization density. In each figure, the $y$-axis represents the time (on a *logarithmic* scale) taken to determine realizability, where that time is the average time over 10 schemas $\mathcal{AS}$ for the respective configuration type. The $x$-axis represents the configuration type for our experiments. A configuration type has form `uu-cd` where `uu` is the number of users (and so the number of tasks as well) and `cd` is the constraint density. The absence of a bar for a given configuration type indicates that the experiment timed out after 20 minutes or ran out of memory.

Figure 9 suggests that the LTL approach outperforms the BDD approach for high values of `ad` such as 1.0: the latter cannot even generate BDDs for workflows with more than 20 users whereas the LTL approach can do this for at least 150 users. Looking at the data on Figures 8 and 7, we can see that the BDD approach seems to catch up to the LTL approach as the value of `ad` becomes lower. The effect of `ad` seems to be reversed in both approaches.

We now analyze how both approaches vary with the value of `cd`. Inspecting the three figures, we note that in each figure its three "zones" of constraint densities have a very similar shape for both approaches. Therefore, we can hypothesize that, in general, this value has less of an effect and the same

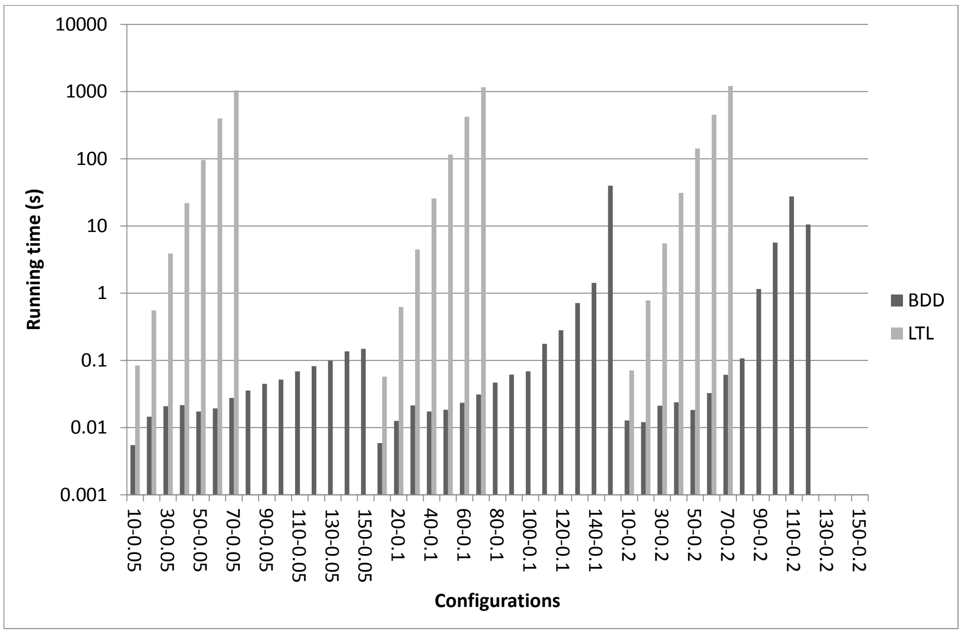


Figure 7: Comparison of time taken to determine realizability using BDDs vs. LTL model checking for authorization density 0.1

type of effect on both the BDD and LTL approaches.

Finally, both approaches find it difficult to scale the decidability of realizability in that the running time appears to grow exponentially in the number of users (and tasks), as the effort resembles a linear function on a logarithmic scale. For the LTL approach, we tried to determine its limits when ad equals 0.5 and cd equals 0.1. These experiments (not reported here) suggest that this approach fails consistently on our machine for models with more than 230 users.

## 5 Conclusions

We presented constrained authorized workflow schemas and motivated the need for workflow management systems to maintain the realizability of such “security-aware” workflows. We suggested two authorization enforcement functions that use automated reasoning methods in order to maintain realizability as an invariant of task execution.

One of these methods relies on a decision procedure for realizability encoded in propositional logic. As a workflow instance executes, this costly procedure needs to be called at each access request instance. Unfortunately, our attempt to circumvent this need through the synthesis of BDDs and their dynamic specialization leads to discouraging experimental results for the build of the initial BDD.

The second method is already reported in [3] and, in effect, computes a subset of the set of all re-

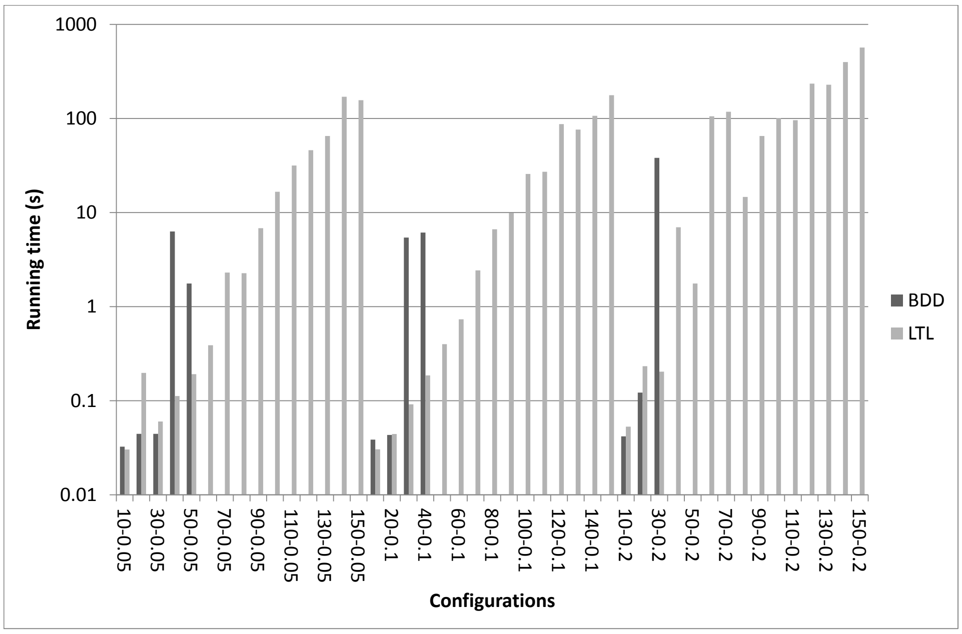


Figure 8: Comparison of time taken to determine realizability using BDDs vs. LTL model checking for authorization density $0.5$

alizability solutions for a workflow instance, where this subset is defined by a sequence of task-user subsets. Such a sequence can be computed using an appropriately configured bounded model-checker for linear-time temporal logic with a suitable encoding derived from the workflow schema. The experimental results for this approach are more encouraging but at least two issues need to be resolved in order to make this approach viable in practice. Firstly, we need to develop refinements of this approach in order to make the model checking more scalable, for example through the use of further abstraction techniques.

Secondly, we need to investigate whether the compact "Boolean" subset of solutions computed by the model checker can, implicitly represent even more solutions and so make the authorization enforcement function more flexible. Delegation models of workflow schemas [4], where users may delegate task execution rights to other users, are just one motivation for such increased flexibility. This second issue has also a more general form: we want to understand the trade-offs between the complexity of computing realizability information that supports an authorization enforcement function and the frequency of denying access requests that, if granted, could in principle lead to realizable workflow instances.

Of course, there are many other approaches to automated reasoning that we may test for their suitability of supporting workflow realizability. Perhaps an incremental SAT solver may allow for a relatively quick decision of the realizability of access requests; we mean to investigate this in future experimental work.

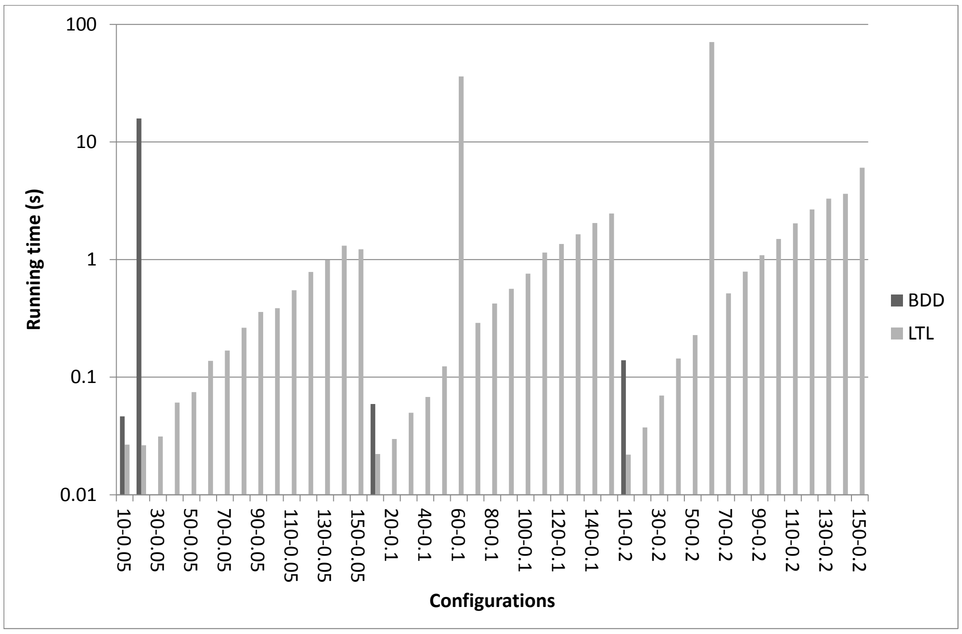


Figure 9: Comparison of time taken to determine realizability using BDDs vs. LTL model checking for authorization density 1.0

The authorized workflow schemas studied in this paper share with existing approaches in the literature that the population of users is already part of the schema. It seems undesirable, somehow, to do the automated reasoning over such a concrete population. In future work, we therefore mean to investigate whether such automated reasoning can be done over a dynamically expanding, symbolic set of users. The aim would be to compute a user/task assignment for symbolic users, which then leaves us with the orthogonal problem of mapping symbolic users into concrete user populations, be it statically or at runtime. Our preliminary study of this new approach suggests that one may fruitfully use constraint satisfaction solvers or efficient instances of colorability problems for the computation of such symbolic solutions.

## References

[1] David Basin, Samuel J. Burri, and Guenter Karjoth. Obstruction-free authorization enforcement: Aligning security and business objectives. In *24th Computer Security Foundations Symposium (CSF 2011)*, pages 99–113, Cernay-la-Ville, France, 06 2011. IEEE.

[2] J. Crampton. A reference monitor for workflow systems with constrained task execution. In *Proceedings of the 10th ACM Symposium on Access Control Models and Technologies*, pages 38–47, 2005.

[3] J. Crampton, M. Huth, and J. H.-P. Kuo. Authorized workflow schemas: Deciding realizability through $\mathsf{LTL}(\mathsf{F})$ model checking. Technical Report 3, Imperial College London, Department of Computing, May 2012. Submitted.

[4] J. Crampton and H. Khambhammettu. Delegation and satisfiability in workflow systems. In I. Ray and N. Li, editors, *Proceedings of 13th ACM Symposium on Access Control Models and Technologies*, pages 31–40, 2008.

[5] A. P. Sistla and E. M. Clarke. The complexity of propositional linear temporal logics. *Journal of the ACM*, 32:733–749, July 1985.

[6] Qihua Wang and Ninghui Li. Satisfiability and resiliency in workflow authorization systems. *ACM Trans. Inf. Syst. Secur.*, 13(4):40, 2010.

# BDD-based automated reasoning in propositional non-classical logics: progress report

Rajeev Goré Jimmy Thomson

**Abstract**

Recent work has shown that a technique using Binary Decision Diagrams (BDDs) to decide **CTL** and **Int** gives promising results. Based on this we explore how the method can be extended to other non-classical logics. In particular, we describe a putative method for deciding the modal $\mu$-calculus using BDDs.

## 1 Introduction

For many logics, we can decide the validity of a given formula $\varphi_0$ by constructing the set of all subsets of some closure $cl(\varphi_0)$, and checking whether these subsets can support a (counter) model that makes $\varphi_0$ false. If no such model exists, then we can safely declare $\varphi_0$ to be valid. Typically, we proceed by first building a finite pseudo-model where each "world" is a member of $2^{cl(\varphi)}$, and then showing that the pseudo-model can be "unfolded" into a model.

At first sight, this "finite pseudo-model (fpm) method" seems impractical since the first step requires us to "construct" the set of all (exponentially many) subsets of $cl(\varphi_0)$, thus giving a procedure whose worst case and best case complexity is always of order $O(2^{|cl(\varphi_0)|})$. However, for **K** and **CTL**, Pan et al. [5] and Marrero [4] have shown that Binary Decision Diagrams (BDDs) can be used to represent the required subsets efficiently, without actually "constructing" them explicitly. We have recently shown how to extend this method to handle modal, tense and bi-extensions of intuitionistic logic **Int** [3]. In particular, for **CTL** and **Int** the resulting reasoners were highly competitive with the current state of the art [2, 3].

In light of this, we are exploring whether such BDD-based implementations can be extended to handle a number of other non-classical logics, and if so, to see whether the practicality remains. Here we concentrate on extending the method to many different classical modal logics, and in particular, the modal mu-calculus. Practicality remains to be seen since we are still implementing the various classical modal logics described here. We do have an initial unoptimised implementation of the putative mu-calculus BDD-method which minimal testing has shown to give the correct answers so far. We have not made it available since it is quite possible that our soundness and completeness proofs for the mu-calculus may not pan out.

Since the focus of PAAR is on practicality, we have deliberately given our descriptions at a lower level than for **Int** [3]. Thus while previously we elided explicit BDD aspects, here they are included, so it may be beneficial to read the other paper first.

We assume that the reader is familiar with non-classical logics in general, in particular with the notion of Kripke semantics and the fpm method for deciding satisfiability. Before discussing extensions to our implementation of the fpm method, we begin by presenting the ideas behind the fpm method in general as a guide for following its actual BDD-based implementation.

### 1.1 An Abstract View of the fpm Method

Given a semantically formulated logic **L**, a naive way to determine satisfiability and validity is to consider the set of all models for said logic and literally determine whether some world in some model exists that makes a formula $\varphi$ true, or if all worlds in all models make $\varphi$ true. In theory, the set of all models is infinite, and in some cases the set of worlds in one model may also be infinite, so we need a way to explore this possibility in a finitary way.

Given a formula $\varphi_0$, the intent of the fpm method is to find a finite filtration of the set of all worlds, such that each member in the filtration represents an equivalence class in the original worlds, and the denotation of the formula $\varphi_0$ in the original model depends only on the truth value of this formula at some representative of this equivalence class. This allows us to examine all worlds in a finitary way. We will in general be referring to members of the filtration as worlds or potential worlds, effectively taking any representative of the equivalence class.

The finite filtration itself is constructed by identifying a finite set of formulae $cl(\varphi_0)$, usually called the "closure of $\varphi_0$", and transforming the given, possibly infinite, model $\mathcal{M}$ into a finite pseudo-model $\mathcal{M}_{cl(\varphi_0)}$ such that the truth value of members of $cl(\varphi_0)$ is preserved.

## 1.2 An operational view of the fpm method

With the space of all potential worlds restricted to a finite space $2^{cl(\varphi_0)}$ (initially), it remains to identify which of these potential worlds correspond to worlds in actual models. The pseudo-model method constructs a finite pseudo-model $(W_f, R^f)$ which is canonical in the following sense: if $\varphi_0$ is satisfiable (falsifiable) then some world of $W_f$ satisfies (falsifies) $\varphi_0$. We find these worlds $W_f$ by defining a monotonic function on sets of potential worlds that removes worlds from the argument set if they contradict the semantics of the logic. For example, a potential world claiming to satisfy both $\Box p$ and $\neg\Box p$ goes against the semantics of most logics, and thus must be removed if present. Thus we construct a chain $W_0, W_1, \cdots W_f$ of refinements on the set of an initial set $W_0$, until $W_f$ is immune to our monotonic function (a fixpoint).

Any (non-empty) fixpoint of this appropriately-constructed function corresponds to a set of worlds which all agree with the semantics of the logic. The "completeness" of this approach requires us to show that every world in any model must have a representative in $W_f$. Because the function we construct satisfies the conditions of the Knaster-Tarski theorem [7] it has a greatest fixpoint, and moreover the greatest fixpoint is a superset of all fixpoints. Thus the greatest fixpoint contains representatives for all worlds in all models. For this reason we start with $W_0 = 2^{cl(\varphi_0)}$, as repeated iteration from the top element will compute the greatest fixpoint.

The "soundness" of this approach requires showing that each world remaining in $W_f$ can be extended into a model using only other worlds in the fixpoint. In some logics this is immediate. In others, like **CTL**, the set must be "unwound" or otherwise manipulated to construct a model.

Thus generating the set of all worlds modulo the closure $cl(\varphi_0)$ of some formula $\varphi_0$ and computing the greatest fixpoint of a sound and complete semantics-inspired function gives a decision procedure where satisfiability and validity are determined by checking whether the intersection of those worlds claiming to satisfy or falsify $\varphi_0$ with the set of all worlds is empty.

We describe specifics of how BDDs are used and how the fixpoint construction works, using **K** as an example, first described by Pan et al. [5].

## 2 Implementation in BDDs

We now describe the BDD implementation at a high level.

**Constructing a Finite Set of Finite Worlds.** As we have seen, given some finite closure $cl(\varphi_0)$, the naive way to construct the finite set of all finite worlds is simply to use the set of all subsets of $cl(\varphi_0)$. We instead use only the "sensible subsets" following Pan et al. and Marrero.

Specifically, we construct $\mathit{Atoms}(\varphi_0)$ as the base set of atoms whose truth values guarantee that we can distinguish worlds. Typically this is the non-classical subset of the closure, from which the truth values of the rest of the closure can be computed using classical conjunction, disjunction and negation. We then define $\mathcal{W} = 2^{\mathit{Atoms}(\varphi_0)}$ to be the set of all subsets of these atoms. Any potential world will either satisfy or falsify each of these atoms, so we can associate a world $w$ with exactly the set of atoms that it satisfies, and hence view $w$ as a simple bi-valent valuation on $\mathit{Atoms}(\varphi_0)$. The set $\mathcal{W}$ is smaller than $2^{cl(\varphi_0)}$, and does not contains worlds which behave inappropriately with respect to classical conjunction and disjunction.

For the logic **K**, an acceptable closure $cl(\varphi_0)$ is the set of subformulae of $\varphi_0$ and their negations. The set of atoms however is defined as the smaller set:

$$\mathit{Atoms}(\varphi_0) = \{\Box\psi \mid \Box\psi \in cl(\varphi_0)\} \cup (\mathit{Prop} \cap cl(\varphi_0))$$

For example, $\mathit{Atoms}(\Box(p \Rightarrow q) \Rightarrow \Box p \Rightarrow \Box q) = \{p, q, \Box p, \Box q, \Box(p \Rightarrow q)\}$ and the set $\{p, \Box(p \Rightarrow q)\}$ corresponds to a world that makes $p$ and $\Box(p \Rightarrow q)$ true, and makes $q$, $\Box p$ and $\Box q$ false.

**BDDs as set of worlds.** We need an efficient way to represent potential worlds and (denotations of formulae as) sets of potential worlds.

A BDD over a set $V = \{v_1, \cdots, v_k\}$ of Boolean-valued variables represents a function mapping each Boolean valuation on these variables to one of $\{t, f\}$. If we associate each atom $a \in \mathit{Atoms}(\varphi_0)$ with a unique BDD variable $v_a$, then a BDD over these variables is a function mapping each valuation on $\mathit{Atoms}(\varphi_0)$ to one of $\{t, f\}$. If we view the valuations which the BDD maps to $t$ as being "selected", then a BDD represents a set of valuations, or a set of potential worlds. Thus a BDD is a function $f : 2^V \mapsto \{t, f\}$ that selects a subset from the powerset $2^V$ of $V$.

For example, in **K** with atoms as above, the set $\{p, \Box(p \Rightarrow q)\}$ corresponds to a valuation under which the BDD variables $v_p$ and $v_{\Box(p\Rightarrow q)}$ are true, while $v_q, v_{\Box p}$ and $v_{\Box q}$ are all false. The BDD which returns $t$ whenever $v_p$, $v_q$, and $v_{\Box p}$ are true corresponds to the set of worlds $\{\{p, q, \Box p\}, \{p, q, \Box p, \Box q\}, \{p, q, \Box p, \Box(p \Rightarrow q)\}, \{p, q, \Box p, \Box q, \Box(p \Rightarrow q)\}\}$.

In particular, the BDD $\top$, which returns $t$ for every valuation, represents the set $\mathcal{W}$ of *all* worlds/subsets over $\mathit{Atoms}(\varphi_0)$ in constant space and time!

The fpm method is usually considered to be naive because it must *"first construct the set of all subsets of $cl(\varphi_0)$, whose cardinality is exponential in the size of $cl(\varphi_0)$"*. The main reason why the fpm method can be implemented efficiently using BDDs is that they turn this "wisdom" on its head. Specifically, by using reduced ordered BDDs the BDD only branches on variables that would cause two valuations to give different results.

**Defining denotations.** For each $a \in \mathit{Atoms}(\varphi_0)$ we use $[\![a]\!]$ to refer to the BDD which is true exactly when the variable corresponding to $a$ is true. Equivalently, $[\![a]\!]$ is the set of worlds that make $a$ true. The denotations of non-atomic formulae in the closure $cl(\varphi_0)$ are computed inductively, usually in an obvious way. For example for **K**, $[\![\psi \wedge \phi]\!] = [\![\psi]\!] \wedge [\![\phi]\!]$, similarly for disjunction and negation, and $[\![\Diamond\psi]\!] = \neg[\![\Box\neg\psi]\!]$.

**Representing relations.** All the logics we consider have relational Kripke semantics so we must be able to represent and reason about these relations.

A BDD $f : 2^V \mapsto \{t, f\}$ over a finite set of variables $V$ corresponds to some subset (of worlds) of $2^V$. Consider a BDD $g(V \cup V')$, corresponding to some subset $S$ of $2^{V \cup V'}$. Any member of $S$, such as $\{v_1, \ldots, v_k\} \cup \{v'_1, \ldots, v'_k\}$, corresponds to a particular valuation on $V \cup V'$. If we conceptually split the valuation into its two components over $V$ and over $V'$, as above, then we can view the valuation as an ordered pair of sub-valuations. This allows us to think of $g(V \cup V')$ as a subset of $2^V \times 2^{V'}$ since there is a bijection from $2^{V \cup V'}$ onto $2^V \times 2^{V'}$ whenever $V$ and $V'$ are disjoint. If $Atoms(\varphi_0)$, $V$ and $V'$ have the same cardinality then $g(V \cup V')$ can be viewed as a subset of $\mathcal{W} \times \mathcal{W}$ using any bijection of $Atoms(\varphi_0)$ onto $V$ and $V'$.

When discussing such BDDs representing pairs, we can think of $g(V \cup V')$ as $g(V, V')$. We will often construct such BDDs by combining BDDs using variables in $V$ or $V'$. When we write $[\![\psi]\!]$, it is constructed from variables in $V$ and represents the first world of a pair, while when we write $[\![\psi]\!]'$ it is constructed from variables in $V'$ and represents the second world in a pair. There is some subtlety here: writing $[\![\psi']\!]$ does not make any sense since $\psi$ is from $cl(\varphi_0)$. Thus $[\![\psi]\!]'$ is a BDD defined over $V'$, which is obtained by making a "photocopy" of the BDD over $V$ for $[\![\psi]\!]$ and replacing each $v_i \in V$ with its clone $v'_i \in V'$.

**The case of K.** Constraints on the specific relation vary by logic, but we present the reasoning for **K** here. The semantics of **K** refer to a Kripke relation R. The relation itself is unrestricted, but its interactions with the modal formulae provide constraints such as the following:

$$\forall w.\mathcal{M}, w \Vdash \Box\psi \Rightarrow \forall v.R(w, v) \Rightarrow \mathcal{M}, v \Vdash \psi \tag{1}$$

Dropping quantifiers, we can rearrange this formula to state a restriction on $R$:

$$R(w, v) \Rightarrow \mathcal{M}, w \Vdash \Box\psi \Rightarrow \mathcal{M}, v \Vdash \psi \tag{2}$$

We treat this formula as an upper bound on $R$, and take the intersection of all the right hand sides given by all $\Box$-formulae in the closure $Atoms(\varphi_0)$ as the definition of a maximal $R$, where maximal means that it links any two worlds that are "allowed to be linked":

$$R(w, v) = \bigwedge_{\Box\psi \in Atoms(\varphi_0)} \mathcal{M}, w \Vdash \Box\psi \Rightarrow \mathcal{M}, v \Vdash \psi \tag{3}$$

The semantics of $\Box$-formulae are captured by this maximal $R$: if two worlds can be related by this $R$, and the first world $w$ claims to satisfy a $\Box\psi$, then the second world $v$ must satisfy $\psi$.

The specific BDD representation of this constraint is as follows, where we use $R(V, V')$ to represent a BDD parametrised by sets of variables $V$ and $V'$:

$$R(V, V') = \bigwedge_{\Box\psi \in Atoms(\varphi_0)} [\![\Box\psi]\!] \Rightarrow [\![\psi]\!]' \tag{4}$$

We have now represented both $\mathcal{W}$ and $R$ using BDDs over $Atoms(\varphi_0)$ and their copies. Recall that the general procedure requires us to refine $\mathcal{W}$ to exclude those worlds that do not obey the semantics of **K**. The remaining task is to construct and solve a fixpoint formula corresponding to the remaining semantics of the logic. We will construct a greatest-fixpoint formula which is monotonic decreasing, so by the Knaster-Tarski theorem we can repeatedly iterate the formula starting with the top element $\mathcal{W}_0 = \mathcal{W} = \top$ to compute the greatest fixpoint $\mathcal{W}_f$. Note that the set $\mathcal{W}$, despite representing $2^{|Atoms(\varphi_0)|}$ worlds, is represented very succinctly by the BDD $\top$, and in general the size of a BDD (and time taken to perform BDD operations) is not proportional to the size of the set it represents but instead depends upon the dependencies between the variables in the characteristic function of the set.

For **K**, the choice of atoms and definition of $[\![\cdot]\!]$ address the classical semantics of $\wedge, \vee$ and $\neg$, and the construction of $R$ enforces the semantics of the $\Box$-formulae. The only semantic conditions left to address are for $\Diamond$-formulae:

$$\forall w.\mathcal{M}, w \Vdash \Diamond\psi \Rightarrow \exists v.R(w,v) \wedge \mathcal{M}, v \Vdash \psi \tag{5}$$

This equation can be used almost exactly to enforce the constraint. One thing to note is that $v$ must be in $\mathcal{M}$, that is it must be in the set of "good" worlds being considered.

$$good(S) = S \wedge \bigwedge_{\Diamond\psi \in cl(\varphi_0)} [\![\Diamond\psi]\!] \Rightarrow \exists V'.R(V,V') \wedge S(V') \wedge [\![\psi]\!]' \tag{6}$$

By $S(V')$ here, we mean a "photocopy" of $S$, where each variable in $V$ is replaced by the variable in $V'$ that corresponds to the same atom.

Given a set of potential good worlds $S$, this function retains the worlds that have candidate $R$-successors in $S$ to witness each diamond they claim to satisfy. That is, it removes from $S$ all potential worlds which cannot "satisfy" their diamonds in the set $S$.

The existential appearing in this formula is QBF-style quantification over a set of variables. Intuitively, this is logically equivalent to the disjunction of all assignments to those variables. In practice, the BDD package we used provides such a function. We have not looked into better ways of doing it ourselves since this is beyond the scope of our research.

## 3 Potential Extensions to other Non-Classical Logics

We now show how the method for **K** [5] can be extended in various directions. Note that all logics considered in this section are known to be decidable (via the "fpm method"), so the main question is really just whether we can find an easy way to capture the method using BDDs.

### 3.1 Multimodal K, extra frame conditions and interacting relations

The huge diversity of propositional modal logics arises from the ability to modally characterise numerous first-order frame conditions on the underlying binary Kripke relation (s).

**Multimodal K.** Pan et al. do not need to consider extra frame conditions on the reachability relation since the modal logic **K** allows arbitrary frames. Extending from **K** to multimodal **K** (aka $\mathcal{ALC}$) is simple as the semantics of the modalities are independent of each other. Instead of constructing a single $R$ relation, there is now an $R_\pi$ relation for each action $\pi$. In the greatest fixpoint computation, instead of referring to the relation $R$, the appropriate $R_\pi$ is used for the $\langle\pi\rangle\psi$ formula at hand. Otherwise, everything follows as for **K** [5].

**Extra frame conditions.** Adding extra frame conditions is not quite so trivial. Marrero handles seriality in **CTL** [4], while our work on **Int** [3] shows how to handle reflexivity and transitivity. We revisit these conditions, and show how to handle euclideanness and symmetry.

Having computed a maximal base relation $R^0$, how can we enforce reflexivity? A naive way is to just take the reflexive closure of $R^0$. However, the $R^0$ we compute is maximally permissive, so if $(w,w) \notin R^0$ then this indicates that $w$ cannot be part of a reflexive model. Thus it is not sound to just add $(w,w)$ back, instead we must remove $w$ from the set of potential worlds by considering only the reflexive worlds from the start:

$$W_0 = \mathcal{W}_{refl} = R^0(V,V) \tag{7}$$

In a similar way, seriality can't be enforced by modifying a base maximal relation $R^0$. In addition, as the fixpoint procedure refines the set of worlds and thus the domain of $R$, the restricted relation may become non-serial, so seriality must be addressed as part of the fixpoint function. Marrero enforces seriality by modifying the *good* function as follows:

$$good(S) = (\exists V'.R(V,V') \wedge S(V')) \wedge \ldots \tag{8}$$

Both seriality and reflexivity are modular in that adding these constraints work for any context without knowledge of the maximal relation $R^0$, while transitivity requires extra knowledge of $R^0$. The important thing about transitivity is the concept of preserving constraints forwards: in the simple case of **K4** this is equivalent to "boxes persist", but with a more complicated relation or logic this may need to be re-evaluated:

$$R(V,V') = R^0(V,V') \wedge \bigwedge_{\Box\psi \in Atoms(\varphi_0)} [\![\Box\psi]\!] \Rightarrow [\![\Box\psi]\!]' \tag{9}$$

Euclideanness can be treated in a very similar way to transitivity. Instead of constraints persisting forwards, constraints must persist backwards: if some successor of $w$ has a constraint ($\Box$-formula) then $w$ itself must have that constraint. For **K5** this is encapsulated as follows:

$$R(V,V') = R^0(V,V') \wedge \bigwedge_{\Box\psi \in Atoms(\varphi_0)} [\![\Box\psi]\!]' \Rightarrow [\![\Box\psi]\!] \tag{10}$$

Symmetry can be handled in a modular way given a maximal relation $R^0$ by restricting it to the maximal symmetric sub-relation as follows:

$$R(w,v) = R^0(w,v) \wedge R^0(v,w) \qquad \equiv \qquad R(V,V') = R^0(V,V') \wedge R^0(V',V) \tag{11}$$

We can thus handle the basic modal logics **KT**, **KD**, **K4**, **K5** and **KB**. The modularity of most of these extensions, and the simplicity of transitivity and euclideanness means that we can also handle combinations of these, allowing us to deal with the 15 basic normal modal logics.

**Interacting relations.** Another direction to consider is interactions between relations. We showed that this approach extends to **BiKt** [3] which has two interacting modal relations. In that case, we were able to sidestep the complications by showing that we could work in a different frame without interaction conditions, and get equivalent answers.

Some interaction conditions are plausibly able to be handled directly however. Statements such as one relation $R_1$ contains $R_2$ result in constraints like so:

$$R_2^1(V,V') = R_2^0(V,V') \wedge R_1^0(V,V') \tag{12}$$

Thus, if $wR_2^1v$, then $wR_1^0v$. If there are multiple conditions, then these restrictions may have to be chained together. Also, such restrictions do not preserve transitivity. If $R_2$ is transitive and $R_1$ is not, then $R_2^1$ may not be transitive after this restriction even if $R_2^0$ is transitive.

Finally, one of the strengths of this method is its versatility. For example, there are two ways to obtain tense logic **Kt**. The first is to start with two modal relations $R_\Box$ and $R_\blacksquare$, and enforce $R_\Box = R_\blacksquare^{-1}$ by requiring that each relation is a subset of the other. The other is to use a single relation $R$ which is defined from semantics referring to both $\Box$-formulae and $\blacksquare$-formulae.

## 3.2 Description Logic

Lite description logics are deliberately weakened fragments of multimodal logics, which can of course be solved using the approaches described here. However, constructing formulae from BDDs is potentially an exponential operation in the number of atoms, and the fixpoints potentially take an exponential number of iterations to compute, so the decision procedure would be inherently exponential, not benefiting from the low computational complexity of Lite logics.

Additionally, a common use-case of description logics is in situations where constraints are relatively simple, but the number of concepts and individuals becomes very large. In these situations this method may not work well, as the number of atoms becomes large.

However, there are ways in which this approach may benefit the types of reasoning done in description logics. Specifically, classifying a TBox reduces to calculating the greatest fixpoint using the denotation of the TBox as an initial value instead of $\mathcal{W}$, then multiple simple queries can be made of the final set to determine whether $C \sqsubseteq D$ for each $C$ and $D$.

Not all the features of the more expressive description logics are feasible either, specifically it is not obvious to us how to handle cardinality constraints.

Functional properties may be possible by treating both $\exists R.C$ and $\forall R.C$ in the same way, as they must both refer to the single successor that an individual must have. The constructed $R$ relation may not itself be functional, but by choosing exactly one of the options at each world, a model where it is functional can be generated.

## 3.3 Hybrid logic

A fixed finite set of nominals is plausible, but binder causes problems both because the logic becomes undecidable, and it is not obvious how to allow arbitrary worlds to be named.

**K**-with-nominals can be represented by requiring that that if a nominal $i$ is true at some world, any other world in the same model claiming to be $i$ must be equivalent to that world.

To represent this, the set of atoms is not just those of **K** with additional atoms for each nominal $i^k$, but an additional $m \times |Atoms(\varphi_0)|$ new atoms for $m$ nominals $i^k$. For each of the "base" atoms $a$, the "additional" atom $@_i a$ is read as "In this model, the world $i$ makes $a$ true". These new atoms must be invariant over the modal relation as shown below at the left, and must interact with the base nominal atoms as shown below at the right:

$$R(V, V') \Rightarrow [\![@_i a]\!] \Leftrightarrow [\![@_i a]\!]' \qquad\qquad [\![i]\!] \Rightarrow [\![@_i a]\!] \Leftrightarrow [\![a]\!]$$

Now if there is a path along $R$ and its converse between two worlds (that is, they appear in the same model) that both claim to make $i$ true, they must be represented by the same set of atoms. Without complications such as irreflexivity or antisymmetry, we are able to treat the equivalence classes as worlds themselves, and thus we can construct a model where the nominal is true at exactly one world.

This choice of atoms works with non-atomic $@_i\psi$ as well by deconstructing $\psi$ into atoms, using $[\![@_i(\psi \wedge \varphi)]\!] = [\![@_i\psi]\!] \wedge [\![@_i\varphi]\!]$, similarly for $\vee$ and $\neg$, and $[\![@_i @_j \psi]\!] = [\![@_j\psi]\!]$.

Although the number of atoms is significantly larger than for **K** and thus performance may well suffer, the procedure remains EXPTIME . But because the $i^k_a$ atoms essentially partition the set of worlds into non-interacting components, the impact on performance may be reduced.

## 4 Using BDDs to decide the $\mu$-calculus

The $\mu$-calculus is infamously tricky to work with, both understanding what a particular formula "means" and deciding whether or not a given formula is satisfiable. The primary difficulty arises from the almost arbitrary fixpoint computations that can be expressed, and the complex interactions between nested fixpoints. Since Marrero [4] described a decision procedure making use of BDDs which involved explicitly calculating least-fixpoints for the temporal "until" eventualities, we consider whether this can be extended to the $\mu$-calculus. We present a procedure which we believe decides the $\mu$-calculus in EXPTIME.

One point to note in particular from Marrero is the way that $\mathbf{A}(\varphi\mathbf{U}\psi)$ was treated: not only did each **EX**-formula have to have a successor (like diamonds in **K**), but they had to have a successor which was consistent with the $\Box$-like nature of the least fixpoint being computed. This concept of considering otherwise-unrelated formulae together is also required for the $\mu$-calculus.

Another thing to note is that the traditional fpm-method does not work for the $\mu$-calculus, and instead automata are traditionally used. Unlike the other logics considered here, we attempt to give a more rigorous explanation, and also give the proofs we currently have. Specifically we believe that we have termination and soundness, but not yet completeness.

### 4.1 Syntax and semantics of the $\mu$-calculus

Formulae of the $\mu$-calculus are built from mutually disjoint sets of atomic formulae $Prop$, atomic actions $Act$ and atomic variables $Var$, where $p \in Prop$, $X \in Var$ and $\pi \in Act$ via:

$$\varphi ::= p \mid \neg p \mid X \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \mu X.\varphi \mid \nu X.\varphi \mid [\pi]\varphi \mid \langle\pi\rangle\varphi$$

Models of $\mu$-calculus formulae are structures $\mathcal{M} = (W, \{R_i\}, \rho)$. Given a valuation $\vartheta : Var \to 2^W$ on variables, denotations with respect to a model $(W, \{R_i\}, \rho)$ are defined via [1]:

$$[\![p]\!]_\vartheta = \rho(p) \qquad [\![\neg p]\!]_\vartheta = W \setminus \rho(p) \qquad [\![X]\!]_\vartheta = \vartheta(X)$$

$$[\![\varphi \wedge \psi]\!]_\vartheta = [\![\varphi]\!]_\vartheta \cap [\![\psi]\!]_\vartheta \qquad [\![\varphi \vee \psi]\!]_\vartheta = [\![\varphi]\!]_\vartheta \cup [\![\psi]\!]_\vartheta$$

$$[\![\mu X.\varphi]\!]_\vartheta = \bigcap\{S \subseteq W \mid S \supseteq [\![\varphi]\!]_{\vartheta[X:=S]}\} \qquad [\![\nu X.\varphi]\!]_\vartheta = \bigcup\{S \subseteq W \mid S \subseteq [\![\varphi]\!]_{\vartheta[X:=S]}\}$$

$$[\![[\pi]\varphi]\!]_\vartheta = \{w \in W \mid \forall v.wR_\pi v \Rightarrow v \in [\![\varphi]\!]_\vartheta\} \qquad [\![\langle\pi\rangle\varphi]\!]_\vartheta = \{w \in W \mid \exists v.wR_\pi v \wedge v \in [\![\varphi]\!]_\vartheta\}$$

Note that $[\![\mu X.\varphi]\!]_\vartheta$ ($[\![\nu X.\varphi]\!]_\vartheta$) can be expressed as least (greatest) fixpoints of $\lambda A.[\![\varphi]\!]_{\vartheta[X:=A]}$.

We will work with closed formulae, and variables are required to be uniquely bound, so for $X \in Var \cap cl(\varphi_0)$ there is exactly one $\xi X.\psi \in cl(\varphi_0)$ where $\xi \in \{\mu, \nu\}$. Thus a variable is uniquely associated with a single fixpoint expression.

The questions we seek to answer are whether there exists a model $\mathcal{M}$ with a world $w$ such that $w \in [\![\varphi]\!]_\emptyset$ (satisfiability), and whether there exists a model and world such that $w \notin [\![\varphi]\!]_\emptyset$ (falsifiability/validity). We solve both questions simultaneously by determining the set of all worlds "relevant to $\varphi_0$" in any model $\mathcal{M}$.

### 4.2 Defining denotations

In addition to the atoms used in multimodal **K**, we create atoms for fixpoints/variables of the $\mu$-calculus. That is, given the set $cl(\varphi_0)$ of all subformulae of $\varphi_0$ and their negations (ensuring to rename variables as necessary to maintain unique bindings), we define:

$$Atoms(\varphi_0) = \{[\pi]\psi \mid [\pi]\psi \in cl(\varphi_0)\} \cup (Prop \cap cl(\varphi_0)) \cup \{\mu X.\psi \mid \mu X.\psi \in cl(\varphi_0)\}$$

Because least and greatest fixpoints are negation duals, we only add one to the set of atoms similarly to how only $\Box$-formulae are made atoms and $\Diamond$-formulae are computed. Because each variable is uniquely bound by exactly one fixpoint formula, we consider the worlds where $X$ holds to be equivalent to the worlds where $\xi X.\psi$ holds. If computing $[\![\varphi_0]\!]_{\vartheta_0}$ eventually requires computing $[\![X]\!]_{\vartheta_n}$, then at some intermediate point it must be computing $[\![\xi X.\psi]\!]_{\vartheta_i}$. The value of $[\![\xi X.\psi]\!]_{\vartheta_i}$ is a fixed point $Z$ such that $Z = [\![\psi]\!]_{\vartheta_i[X:=Z]}$. Thus $[\![X]\!]_{\vartheta_n} = Z$, so $X$ and $\xi X.\psi$ have the same denotation, and we refer to the atom as $X$ or $\xi X.\psi$ interchangeably.

Thus, we define the following:

$$\begin{aligned}
[\![a]\!] &= \{w \in \mathcal{W} \mid w \in [\![a]\!]_\vartheta\} && \text{where } a \in Atoms(\varphi_0) \text{ and } \vartheta(X) = [\![\xi X.\psi]\!]_\vartheta \\
[\![X]\!] &= [\![\xi X.\psi]\!] && \text{where } X \text{ is uniquely bound by } \xi X.\psi \\
[\![\nu X.\psi]\!] &= \neg[\![\mu Y.\phi]\!] && \text{where } \mu Y.\phi \text{ is the negation dual} \\
[\![\neg p]\!] &= \neg[\![p]\!] \\
[\![\langle\pi\rangle\psi]\!] &= \neg[\![[\pi]\neg\psi]\!] \\
[\![\psi_1 \wedge \psi_2]\!] &= [\![\psi_1]\!] \wedge [\![\psi_2]\!] \\
[\![\psi_1 \vee \psi_2]\!] &= [\![\psi_1]\!] \vee [\![\psi_2]\!]
\end{aligned}$$

It is important to note that $[\![\psi]\!]$ and $[\![\psi]\!]_\vartheta$ have different meanings: $[\![\psi]\!]$ is something that we construct, and we eventually want it to correspond to the semantic notion of $[\![\psi]\!]_\vartheta$, but this is not the case yet.

The $R_\pi$ relations are constructed in the same manner as for multimodal **K**, but it is much more important to note that $R_\pi$ is an over-approximation here. Because we now have variables, the denotation for $[\pi]X$ is not fixed, so while the $R_\pi$ we construct here will be useful, it does not capture the entire semantics of $\Box$-formulae now.

## 4.3 Enforcing semantics

As with the other logics, we now want to construct a fixpoint formula that enforces the model-theoretic semantics. The component of the fixpoint formula dealing with $\langle\pi\rangle\psi$ formula is the same as for multimodal **K**, so the remaining consideration is the fixpoints.

Instead of a shallow "local" evaluation, such as used for the limited eventualities in **CTL**, because the fixpoint formulae expressible in the $\mu$-calculus are almost arbitrary, we inspect the formula deeply to compute the appropriate denotation.

For each least (greatest) fixpoint $\xi X.\psi$ in the closure we use the fixpoint semantics of the logic, rather than the infinite intersection / union, by calculating $\lambda A.[\![\psi]\!]_{[X:=A]}$: the denotation of $\psi$ given that the denotation of $X$ is $A$. This involves computing nested fixpoints and dealing with modalities as well. Diamond-formulae are simple, as the pre-image of the denotation of the successor world can be computed.

However, box-formulae in the fixpoint are not as simple as negating and treating as diamonds. A world $w$ satisfying $\langle\pi\rangle\psi$ at some intermediate fixpoint valuation is interpreted as "$w$ can have a successor satisfying $\psi$", which means that the negation or complement of this set is interpreted as "$w$ cannot have a successor satisfying $\psi$". At intermediate stages this interpretation can be overly restrictive, and we should instead consider "it is possible for $w$ to have 0 or more successors, all of which falsify $\psi$".

For example, consider the fixpoint $\mu X.[\pi]X$ in a closure $\{X, [\pi]X, \langle\pi\rangle q, q\}$. At some stage when computing worlds where this least fixpoint $X$ holds, we might consider worlds $w = \{X, [\pi]X, \langle\pi\rangle q\}$, $u = \{X, [\pi]X, [\pi]\neg q, q\}$, $v = \{X, [\pi]X, \langle\pi\rangle q, q\}$. According to $R_\pi$, we have $R_\pi(w, u), R_\pi(w, v), R_\pi(w, w), R_\pi(u, w), R_\pi(v, w), R_\pi(v, u)$ and $R_\pi(v, v)$. Suppose that at some iteration of the least fixpoint, $u$ is found to be in the fixpoint, but $w$ and $v$ are not. If we compute $\neg\exists V'.R_\pi(V, V') \wedge S(V') \wedge \neg X(V')$, as $[\pi]X \equiv \neg\langle\pi\rangle\neg X$, then $w$ and $v$ will be excluded, because $R_\pi(w, v)$ and $R_\pi(v, w)$. However, it is possible to construct a model with a world $w$ and without the edge $R_\pi(w, v)$, so this result is incorrect.

In order to correct this, we use something like the following formula:

$$[\![[\pi]\psi]\!] \wedge \bigwedge_{\langle\pi\rangle\chi \in cl(\varphi_0)} [\![\langle\pi\rangle\chi]\!] \Rightarrow \exists V'.R_\pi(V, V') \wedge \mathcal{V}(\chi)' \wedge G([\pi]\psi)$$

Here $\mathcal{V}(\chi)$ deeply expands $\chi$ according to the intuitions here and above, and $G([\pi]\psi)$ is a term to account for boxes being true simultaneously.

The intuition behind this formula is that if $[\pi]\psi$ holds at a world, for that world to be acceptable then all its existentials must be satisfiable in a way that is consistent with the box: If $\langle\pi\rangle\chi$ is true, then there is an $R_\pi$ successor where $\chi$ is true (the $\Diamond$-formula is satisfied) and this is consistent with the boxes that are true (the $G$ term).

Before going into specifics of what $G$ is, note that as-is the formula can have an infinite loop: when considering the formula $\langle\pi\rangle[\pi]X$, the $\Box$-formula will recurse on the $\Diamond$-formula which will refer once again to the $\Box$-formula. We resolve this by introducing another fixpoint formula, such that any fixpoint of the formula gives a consistent denotation for $[\pi]\psi$. Then the greatest fixpoint of this formula contains all fixpoints, and thus the greatest fixpoint of the formula gives a maximal denotation for $[\pi]\psi$. This requires some changes elsewhere, which we address after presenting the expansion as a whole.

The $G$ term in the formula is intended to capture the restriction of boxes, in much the same way as the constructed $R_\pi$ relation. The difference is that it once again deeply expands the formulae and considers the current set of assumed denotations, both for fixpoint variables and additionally for $\Box$-formulae.

To bring this all together, we define a function $\mathcal{V}(\psi, S, \sigma_{var}, \sigma_\Box)$ which performs the deep-analysis of $\psi$ given that all worlds must be in $S$, some variables have denotations given by $\sigma_{var}$, and some $\Box$-formulae have denotations given by $\sigma_\Box$, shown in Figure 1.

Note that when a new fixpoint is encountered, the assignments to $\Box$-formulae are forgotten during that calculation, since the assignments to variables changing can potentially change the denotation of a $\Box$. For example $[\pi]X$ may have some current denotation including worlds with successors satisfying $X$, but then $X$ is assigned the empty denotation, meaning that $[\pi]X$ can only be true at worlds with no $\pi$-successors.

Finally, we bring this all together for the greatest fixpoint formula as follows:

$$\begin{aligned} good(S) = S \wedge & \bigwedge_{\langle\pi\rangle\psi \in cl(\varphi_0)} [\![\langle\pi\rangle\psi]\!] \Rightarrow \mathcal{V}(\langle\pi\rangle\psi, S, \emptyset, \emptyset) \\ \wedge & \bigwedge_{\mu Z.\psi \in cl(\varphi_0)} [\![\mu Z.\psi]\!] \Rightarrow \mathcal{V}(\mu Z.\psi, S, \emptyset, \emptyset) \\ \wedge & \bigwedge_{\nu Z.\psi \in cl(\varphi_0)} [\![\nu Z.\psi]\!] \Rightarrow \mathcal{V}(\nu Z.\psi, S, \emptyset, \emptyset) \end{aligned}$$

$$
\begin{aligned}
\mathcal{V}(p, S, \sigma_{var}, \sigma_\Box) &= [\![p]\!] \wedge S \\
\mathcal{V}(\neg p, S, \sigma_{var}, \sigma_\Box) &= \neg[\![p]\!] \wedge S \\
\mathcal{V}(X, S, \sigma_{var}, \sigma_\Box) &= \begin{cases} \sigma_{var}(X) \wedge S & \text{if } X \in \sigma_{var} \\ [\![X]\!] \wedge S & \text{otherwise} \end{cases} \\
\mathcal{V}(\psi_1 \wedge \psi_2, S, \sigma_{var}, \sigma_\Box) &= \mathcal{V}(\psi_1, S, \sigma_{var}, \sigma_\Box) \wedge \mathcal{V}(\psi_2, S, \sigma_{var}, \sigma_\Box) \\
\mathcal{V}(\psi_1 \vee \psi_2, S, \sigma_{var}, \sigma_\Box) &= \mathcal{V}(\psi_1, S, \sigma_{var}, \sigma_\Box) \vee \mathcal{V}(\psi_2, S, \sigma_{var}, \sigma_\Box) \\
\mathcal{V}(\mu X.\psi, S, \sigma_{var}, \sigma_\Box) &= \begin{cases} \sigma_{var}(X) \wedge S & \text{if } X \in \sigma_{var} \\ LFP(\lambda A.\mathcal{V}(\psi, S, \sigma_{var}[X := A], \emptyset)) & \text{otherwise} \end{cases} \\
\mathcal{V}(\nu X.\psi, S, \sigma_{var}, \sigma_\Box) &= \begin{cases} \sigma_{var}(X) \wedge S & \text{if } X \in \sigma_{var} \\ GFP(\lambda A.\mathcal{V}(\psi, S, \sigma_{var}[X := A], \emptyset)) & \text{otherwise} \end{cases} \\
\mathcal{V}(\langle\pi\rangle\psi, S, \sigma_{var}, \sigma_\Box) &= S \wedge \exists V'.R_\pi(V, V') \wedge \mathcal{V}(\psi, S, \sigma_{var}, \sigma_\Box)' \\
\mathcal{V}([\pi]\psi, S, \sigma_{var}, \sigma_\Box) &= \begin{cases} \sigma_\Box([\pi]\psi) \wedge S & \text{if } [\pi]\psi \in \sigma_\Box \\ GFP(\lambda A.S \wedge [\![[\pi]\psi]\!] \wedge \bigwedge_{\langle\pi\rangle\chi \in cl(\varphi_0)} & \\ \quad [\![\langle\pi\rangle\chi]\!] \Rightarrow \exists V'.R_\pi(V, V') \wedge G(A) & \text{otherwise} \\ \qquad \wedge \mathcal{V}(\chi, S, \sigma_{var}, \sigma_\Box[[\pi]\psi := A])) & \end{cases}
\end{aligned}
$$

*where*

$$G(A) = \bigwedge_{[\pi]\phi \in Atoms(\varphi_0)} [\![[\pi]\phi]\!] \Rightarrow \mathcal{V}(\phi, S, \sigma_{var}, \sigma_\Box[[\pi]\psi := A])'$$

Figure 1: The function to compute the denotation of a $\mu$-calculus formula by deep-inspection.

In fact, the component dealing with $\Diamond$-formulae can also be written in the same manner as for **K**, but this more general statement is easier on the proofs.

## 4.4 Proofs

First we note that all fixpoints can be computed accurately by repeated iteration. This is a consequence of all the fixpoint formulae being monotone, and the Knaster-Tarski theorem.

We present a proof that the procedure described above is sound: If a formula is falsifiable, then the procedure will find a witness.

We aim to prove that given a subset $S$ of the filtration, any model $\mathcal{M} = (W, \{R_i\}, \rho)$ such that the filtration of $W$ is a subset of $S$, and a world $w \in W$, if $w \in [\![\psi]\!]_\emptyset$ then the representative of $w$ in the filtration is in $\mathcal{V}(\psi, S, \emptyset, \emptyset)$.

We do this by proving a stronger theorem:

**Theorem 1.** *Given a model $\mathcal{M} = (W, \{R_i\}, \rho)$, a subset $S$ of $\mathcal{W}$, a partial map $\sigma_{var}$ from fixpoint formulae to denotations, and a partial map $\sigma_\Box$ from $\Box$-formulae to denotations, if*

1. *the worlds of $W$ are all represented in $S$; and*
2. *for each fixpoint variable $Z \in dom(\sigma_{var})$, $[\![\xi Z.\varphi]\!]_{\sigma_{var}} = \sigma_{var}(Z)$;*

3. *for each fixpoint variable $Z \notin dom(\sigma_{var})$, when $\sigma_{var}$ is used as a valuation then $[\![Z]\!]_{\sigma_{var}} = \sigma_{var}(Z) = [\![\xi Z.\varphi]\!]_{\sigma_{var}}$; and*

4. *for each formula $[\pi]\varphi \in dom(\sigma_\Box)$, $[\![[\pi]\varphi]\!]_{\sigma_{var}} = \sigma_\Box([\pi]\varphi)$*

*then for all $w \in W$, if $w \in [\![\psi]\!]_{\sigma_{var}}$ then $w \in \mathcal{V}(\psi, S, \sigma_{var}, \sigma_\Box)$.*

We define an ordering on $(\psi, \sigma_{var}, \sigma_\Box) \in cl(\varphi_0) \times ((Var \cap cl(\varphi_0)) \times \mathcal{W}) \times (\{[\pi]\psi \mid cl(\varphi_0)\} \times \mathcal{W})$ as follows:

**Definition 2.** $(\psi^1, \sigma^1_{var}, \sigma^1_\Box) < (\psi^2, \sigma^2_{var}, \sigma^2_\Box)$ iff $\sigma^1_{var} \supset \sigma^2_{var}$ or ($\sigma^1_{var} = \sigma^2_{var}$ and ($\sigma^1_\Box \supset \sigma^2_\Box$ or ($\sigma^1_\Box = \sigma^2_\Box$ and $\psi^1$ is a strict subformula of $\psi^2$.)))

This ordering is well-founded because $\subset$ and strict subformulae are well-founded. The ordering also corresponds to the definition of the $\mathcal{V}$ function.

*Proof.* We make use of well-founded induction over this ordering

- $\mathcal{V}(p, S, \sigma_{var}, \sigma_\Box)$. From the definition of $[\![\cdot]\!]$ for atoms, if $w \in [\![p]\!]_{\sigma_{var}}$ then $w \in [\![p]\!]$. Also by assumption $w \in S$, so $w \in [\![p]\!] \cap S$.

- $\mathcal{V}(\neg p, S, \sigma_{var}, \sigma_\Box)$. From the semantics, if $w \in [\![\neg p]\!]_{\sigma_{var}}$ then $w \in \mathcal{M} \setminus [\![p]\!]_{\sigma_{var}} \subseteq \neg[\![p]\!]$. Also by assumption $w \in S$, so $w \in \neg[\![p]\!] \cap S$.

- $\mathcal{V}(Z, S, \sigma_{var}[Z := X], \sigma_\Box)$. By definition, $[\![Z]\!]_{\sigma_{var}[Z:=X]}$ must be $X$. Thus $w \in X$, and $w \in S$ by assumption, therefore $w \in X \cap S$.

- $\mathcal{V}([\pi]\psi, S, \sigma_{var}, \sigma_\Box[[\pi]\psi := X])$. By assumption 4, $w \in X$, and by assumption 1, $w \in S$. Thus $w \in X \cap S$.

- $\mathcal{V}(\mu Z.\psi, S, \sigma_{var}[Z := X], \sigma_\Box)$. By assumption 2, $w \in X$, and by assumption 1, $w \in S$. Thus $w \in X \cap S$

- $\mathcal{V}(\nu Z.\psi, S, \sigma_{var}[Z := X], \sigma_\Box)$. As above.

- $\mathcal{V}(X, S, \sigma_{var}, \sigma_\Box)$ when $X \notin dom(\sigma_{var})$. By assumption 3, since $w \in [\![X]\!]_{\sigma_{var}}$ we have that $w \in [\![\xi X.\psi]\!]_{\sigma_{var}}$. By the definition of $[\![\cdot]\!]$, this means that $w \in [\![\xi X.\psi]\!]$ or equivalently $w \in [\![X]\!]$. Since $w \in S$ by assumption, we therefore have $w \in [\![X]\!] \wedge S$ as required.

- $\mathcal{V}(\psi_1 \wedge \psi_2, S, \sigma_{var}, \sigma_\Box)$. If $w \in [\![\psi_1 \wedge \psi_2]\!]_{\sigma_{var}}$ then $w \in [\![\psi_1]\!]_{\sigma_{var}}$ and $w \in [\![\psi_2]\!]_{\sigma_{var}}$. By induction we therefore have $w \in \mathcal{V}(\psi_1, S, \sigma_{var}, \sigma_\Box)$ and $w \in \mathcal{V}(\psi_2, S, \sigma_{var}, \sigma_\Box)$, and thus $w$ is in the intersection as required.

- $\mathcal{V}(\psi_1 \vee \psi_2, S, \sigma_{var}, \sigma_\Box)$. As for $\psi_1 \wedge \psi_2$.

- $\mathcal{V}(\langle\pi\rangle\psi, S, \sigma_{var}, \sigma_\Box)$. If $w \in [\![\langle\pi\rangle\psi]\!]_{\sigma_{var}}$ then there exists a $v \in \mathcal{M}$ such that $wR_\pi v$, and $v \in [\![\psi]\!]_{\sigma_{var}}$. By induction, such a $v$ must be in $\mathcal{V}(\psi, S, \sigma_{var}, \sigma_\Box)$.

  Consider one component of the constructed $R_\pi$, say $[\![[\pi]\varphi]\!] \Rightarrow [\![\varphi]\!]'$. If $w \in [\![[\pi]\varphi]\!]_{\sigma_{var}}$ then $w \in [\![[\pi]\varphi]\!]$ by the definition of $[\![\cdot]\!]$. Additionally $v \in [\![\varphi]\!]_{\sigma_{var}}$ due to the semantics of $\Box$-formulae, so $v \in [\![\varphi]\!]$. Thus $(w, v)$ is in that component. If $w \notin [\![[\pi]\varphi]\!]_\vartheta$ then we have $w \in \neg[\![[\pi]\varphi]\!]$ and thus $(w, v)$ is in the component. Thus $(w, v)$ is in each component of $R_\pi$, so it is in their intersection and $(w, v) \in R_\pi$.

  Together with the assumed $w \in S$, this means that $w \in S \wedge \exists V'.R_\pi(V, V') \wedge \mathcal{V}(\psi, S, \sigma_{var}, \sigma_\Box)'$ as required.

- $\mathcal{V}(\nu X.\psi, S, \sigma_{var}, \sigma_\Box)$. Given that $w \in [\![\nu X.\psi]\!]_\vartheta$, the semantics require that $w \in [\![\psi]\!]_{\vartheta[X:=A]}$ for some $A \subseteq [\![\psi]\!]_{\vartheta[X:=A]}$. For such $A$, each $v \in A$, must of course satisfy $v \in [\![\psi]\!]_{\vartheta[X:=A]}$. Since formulae are restricted to being monotone with respect to variable assignments, $A$ is a subset of some fixpoint, and the greatest fixpoint $Z$ contains all fixpoints. Thus $w \in [\![\psi_{\vartheta[X:=Z]}]\!]$.

  By induction, for each $v \in Z$ we therefore have $v \in \mathcal{V}(\psi, S, \sigma_{var}[X := Z], \emptyset)$, and so $Z$ is a fixed point of $\lambda A.\mathcal{V}(\psi, S, \sigma_{var}[X := A], \emptyset)$. Since the greatest fixpoints contains all fixpoints, $w$ is in the greatest fixpoint, and thus $w \in \mathcal{V}(\nu X.\psi, S, \sigma_{var}, \sigma_\Box)$.

- $\mathcal{V}(\mu X.\psi, S, \sigma_{var}, \sigma_\Box)$. Given that $w \in [\![\mu X.\psi]\!]_{\sigma_{var}}$, the semantics require that $w \in \bigcap\{A \subseteq \mathcal{M} \mid [\![\psi]\!]_{\sigma_{var}[X:=A]} \subseteq A\}$.

  Take $Z = LFP(\lambda A.A \vee \mathcal{V}(\psi, S, \sigma_{var}[X := A], \emptyset))$. Because $Z$ is a fixed point, $Z = \mathcal{V}(\psi, S, \sigma_{var}[X := Z], \emptyset)$. Consider any $v \in [\![\psi]\!]_{\sigma_{var}[X:=Z]}$. By induction, $v \in \mathcal{V}(\psi, S, \sigma_{var}[X := Z], \emptyset)$. This means that $[\![\psi]\!]_{\sigma_{var}[X:=Z]} \subseteq Z$ holds, and thus $w \in Z$, and thus $w \in \mathcal{V}(\mu X.\psi, S, \sigma_{var}, \sigma_\Box)$ as required.

- $\mathcal{V}([\pi]\psi, S, \sigma_{var}, \sigma_\Box)$. when $[\pi]\psi \notin dom(\sigma_\Box)$.

  We first show that

  $$w \in ([\![\langle\pi\rangle\chi]\!] \Rightarrow \exists V'.R_\pi(V, V') \wedge G([\![[\pi]\psi]\!]_{\sigma_{var}}) \wedge \mathcal{V}(\chi, S, \sigma_{var}, \sigma_\Box[[\pi]\psi := [\![[\pi]\psi]\!]_{\sigma_{var}}]))$$

  for each $\langle\pi\rangle\chi$. If $w \notin [\![\langle\pi\rangle\chi]\!]_{\sigma_{var}}$ then $w \in [\![[\pi]\neg\psi_1]\!]_{\sigma_{var}}$ and so $w \in [\![[\pi]\neg\psi_1]\!] = \neg[\![\langle\pi\rangle\psi_1]\!]$.

  Otherwise, there must be some $v \in \mathcal{M}$ such that $wR_\pi v$ and $v \in [\![\chi]\!]_{\sigma_{var}}$. By induction, $v \in \mathcal{V})(\chi, S, \sigma_{var}, \sigma_\Box[[\pi]\psi := [\![[\pi]\psi]\!]_{\sigma_{var}}])$ since condition 4 holds by definition. Using the same method as we did for $\langle\pi\rangle\psi$ we have that $(w, v) \in R_\pi$.

  We must show that $(w, v)$ satisfies each of the conjuncts of $G([\![[\pi]\psi]\!]_{\sigma_{var}})$. If $w \notin [\![[\pi]\phi]\!]_{\sigma_{var}}$ then $w \in \neg[\![[\pi]\phi]\!]$, and thus the pair satisfies the conjunct. Otherwise $v \in [\![\phi]\!]_{\sigma_{var}}$, so by induction $v \in \mathcal{V}(\phi, S, \sigma_{var}, \sigma_\Box[[\pi]\psi := [\![[\pi]\psi]\!]_{\sigma_{var}}])$, so $(w, v)$ satisfies the conjunct.

  Thus we have shown that $(w, v)$ must satisfy the existentially quantified formula, and thus $w$ satisfies the existential quantification for each $\langle\pi\rangle\chi$ in the closure.

  Because $w \in [\![[\pi]\psi]\!]_{\sigma_{var}}$ we have $w \in [\![[\pi]\psi]\!]$, and by assumption we have $w \in S$. By generalising, we have $[\![[\pi]\psi]\!]_{\sigma_{var}} \subseteq f([\![[\pi]\psi]\!]_{\sigma_{var}})$ for the fixpoint expression we define, and thus it is a subset of the greatest fixpoint. Thus $w \in \mathcal{V}([\pi]\psi, S, \sigma_{var}, \sigma_\Box)$ as required.

$\dashv$

We can then apply this theorem to show that for any world $w$ of any model $\mathcal{M}$ using only worlds in $S$, if $w \in [\![\psi]\!]_\emptyset$, then $w \in \mathcal{V}(\psi, S, \emptyset, \emptyset)$. Condition 1 of Theorem 1 is explicitly enforced, and conditions 2 and 4 hold vacuously. Condition 3 potentially restricts the valuations we consider, but this has no impact on closed formulae.

To show that this method is complete, an additional step is required: If $w \in S$, then $w \in good(S)$ for any world $w$ in any model $\mathcal{M}$.

*Proof.* Given that $w \in S$, the first conjunct is trivially satisfied. For the remaining conjuncts, Suppose that $w \in [\![\psi]\!]_\emptyset$ for $\psi \in \{\langle\pi\rangle\psi_1, \mu X.\psi_1, \nu X.\psi_1\}$. By Theorem 1 we have $w \in \mathcal{V}(\psi, S, \emptyset, \emptyset)$ as required. Thus $w$ is in each of the conjuncts, and so $w \in good(S)$ as required. $\dashv$

We now show that this procedure remains in EXPTIME.

**Theorem 3.** *The procedure described above can be computed in $O(2^{O(n)})$ time.*

*Proof.* Computing a BDD over a set of variables $V$ takes time proportional to $2^{|V|}$. Since we have two variables per atom, and we have $O(N)$ atoms, each BDD formula can be computed in $O(2^{O(N)})$ time. Each fixpoint is computed by repeated iteration until the answer is unchanged. The result space of each fixpoint is $\mathcal{W}$ of size $2^{|Atoms(\varphi_0)|}$, so each fixpoint is computed in $O(2^{O(n)})$ iterations.

Consider the $\mathcal{V}$ function. Since $S$ is fixed, we can associate each instance of the function with a tuple of formula, fixpoint denotations, and $\Box$-formula denotations. The number of formulae is polynomial in the size of the initial formula, and there are $O(N2^{|Atoms(\varphi_0)|})$ possible ways of assigning fixpoint denotations and $\Box$-formula denotations. This means that there are $O(2^{O(N)})$ different calls to $\mathcal{V}$ given $S$. There is the possibility that $\mathcal{V}$ may be called with the same arguments multiple times. However, since it is pure functional, if this is the case then the results can be cached and the exponential bound retained.

To complete the proof, observe that each call to $\mathcal{V}$ does at most an exponential amount of work, computing a BDD or calling $\mathcal{V}$ at most exponentially many times, and the outermost greatest fixpoint formula calls interpret a linear number of times each iteration, thus the procedure takes at most $O(2^N)$ time. $\dashv$

## 5 Further work

**$\mu$-calculus.** We have yet to prove that the method we describe for the $\mu$-calculus is sound: in theory the fixpoint computed may contain representatives for potential worlds which do not appear in any model, and thus may find false counterexamples. One solution is to construct a model for each representative in the final fixpoint, and we are currently working on this.

**Methodology for semantic constraints.** Another area worth considering is whether it is possible to algorithmically construct BDDs to represent certain classes of first-order-definable conditions. Currently we have tried to explain our insights for particular frame conditions of interest (Section 3), but we do not have a mechanical translation from semantics to BDD conditions. This kind of construction is possible in tableau methods [6], so similar methods may allow for less human intuition in these BDD-based methods as well.

**Substructural logics.** When moving to substructural logics, instead of a binary Kripke relation, there is often a ternary relation of some sort. In much the same way as we represented a binary relation by having a single copy $a'$ of each atom $a$, we can represent a ternary relation by having 2 copies $a'$ and $a''$ of each atom $a$. The question then is whether we construct maximal ternary relations and use them in the same way that we used the maximal binary relations.

Many substructural logics are undecidable, and thus we won't be able to make a decision procedure. Nonetheless we do believe that some decidable substructural are amenable to this approach. We have started looking at a decidable fragment of separation logic, and believe it to be feasible, but do not have any results yet.

**First order logic.** One area that we have yet to consider is extending to first order logics. Once again many such logics are undecidable, but perhaps we can construct a semidecison procedure, or perhaps this approach could work for a decidable fragment of first order logic.

One of the first issues to consider is what set of atoms should be used. As soon as function symbols are introduced, there are potentially an infinite number of distinct objects, distinguishable by how many times the function symbol is applied. If we cannot set a fixed finite space to care about in the first place, then significant changes must follow. So far we have yet to find a way of handling this without essentially using a different automated reasoning technique.

**Bernays–Schönfinkel class.** This class of first order logic requires that in prenex normal form, all existential quantifiers occur before any universal quantifier, and there are no function symbols. Equivalently, the skolem form of the formulae contains only nullary functors/constants. This restricted setting is known to be decidable.

We considered treating the constants as nominals, and predicates as relations or propositions true of a world. However what should the closure be? If the closure includes the negation of the input formula, then the closure includes formulae which are not in the Bernays-Schönfinkel class, and will in general include existential statements.

**BDDs for other methods.** Given that the BDD-based decision procedures for **CTL** and **Int** were competitive, we are also considering whether other automated reasoning methods could benefit from using BDDs. In particular, we are implementing a tableau procedure using BDDs. Potential benefits include fast equality checks; simple unsat caching by constructing a BDD of known-bad formulae and restricting the tableau nodes considered to the complement of that; and fast saturation phases.

## References

[1] Julian Bradfield and Colin Stirling. Modal mu-calculi. In *The Handbook of Modal Logic*, pages 721–756. Elsevier, 2006.

[2] R. Goré, J. Thomson, and F. Widmann. An experimental comparison of theorem provers for CTL. In *Temporal Representation and Reasoning (TIME), 2011 Eighteenth International Symposium on*, pages 49 –56, sept. 2011. doi: 10.1109/TIME.2011.16.

[3] Rajeev Goré and Jimmy Thomson. BDD-based automated reasoning for propositional bi-intuitionistic tense logics. In *IJCAR*, 2012, to appear.

[4] Will Marrero. Using BDDs to decide CTL. *Lecture Notes in Computer Science*, 3440/2005: 222–236, 2005.

[5] Guoqiang Pan, Ulrike Sattler, and Moshe Y. Vardi. BDD-based decision procedures for the modal logic K. *Journal of Applied Non-classical Logics*, 49, 2005.

[6] R. A. Schmidt and D. Tishkovsky. Automated synthesis of tableau calculi. *Logical Methods in Computer Science*, 7(2):1–32, 2011. doi: http://dx.doi.org/10.2168/LMCS-7(2:6)2011.

[7] Alfred Tarski. A lattice-theoretical fixpoint theorem and its applications. *Pacific journal of Mathematics*, 5(4):285–309, 1955.

# A One-Pass Tableau-Based Workflow Verification Framework

Md Zahidul Islam and Wendy MacCaull

Centre for Logic and Information
St. Francis Xavier University, Antigonish, Canada
{x2010mce, wmaccaul}@stfx.ca

**Abstract**

Workflow management systems (WfMSs) are useful tools for supporting enterprise information systems. Such systems must ensure compliance with guidelines and regulations. While formal verification techniques can be used in the development stages to help ensure behavioral properties of many systems, these techniques are generally not available in workflow tools. We present a framework which models workflows using Petri nets and translates the model to a tableau style model checker. The model checker uses the recently introduced one-pass tableau algorithm and delivers enhanced performance over traditional two-pass strategies in practical applications. A failed tableau will generate a counter model which can aid in debugging. We present a case study involving a health services delivery program, and verify properties written in Computation Tree Logic (CTL). The tableau method can be modified to accommodate other specification languages such as timed CTL, logics of beliefs, desires and intentions, temporal description logic, first order logic, and others.

## 1 Introduction

It is a common practice to analyze a system's behaviour before its actual implementation. Established analysis approaches like test beds allow for rigorous, transparent, and replicable testing of software, hardware, and networking systems. However, they are difficult to set up, are usually very costly, and require experts. Another approach, simulation, involves providing certain inputs and observing their corresponding outputs. Providing all possible inputs and observing their outputs is tedious and usually impractical. These shortcomings led researchers to the application of formal verification in system analysis and development. This involves modelling systems using an adequate level of abstraction which decreases analysis cost and gives a rigorous view of the system. Models are relatively easy to modify and errors found before implementation can greatly decrease cost. Properly verified models ensure better processes. We present a framework for applying formal verification to workflow models.

There are two formal verification approaches: theorem proving and model checking. Theorem proving is a logic based proof theoretic approach which typically uses a very expressive language for describing systems and property specifications. The system is expressed as a set of axioms and the specifications are expressed as formulae; a proof system is used to determine if the formulae are valid. In model checking, the description of a system (also known as a model) is given by the specification language of a model checker and the model checker determines if a (usually) temporal logic (such as Computation Tree Logic (CTL), or Linear Temporal Logic (LTL)) formula holds for the model. Applying a formal verification technique by modelling a system using a formal specification language is generally a difficult and tedious task. Our framework starts with a human-friendly specification language, Petri nets, which has a graphical representation easily modelled with the Coloured Petri net (CPN) tool [23]; the Petri net models are automatically transformed to Kripke structures for formal analysis.

The system does not have to be represented in the formal specification language of an existing model checker.

The tableau method is a popular proof procedure applicable to a wide range of logics including temporal logics. The traditional tableau method for CTL is a two-pass procedure [2] which applies a set of tableau rules to construct a tableau in the first pass and determines the inconsistent nodes (the nodes that cannot be a part of a valid model for the given CTL formula) in the second pass. A formula is satisfiable if and only if a model for the formula can be found in the tableau. Recently an improvement over the two-pass procedure, known as the one-pass tableau procedure [1], was developed. A comparison between the one-pass and two-pass procedure in [8] showed that the one-pass procedure consistently, and sometimes dramatically, outperformed the two-pass procedure. The one-pass procedure requires a single pass through the tree to determine the satisfiability of a formula. This procedure can be used for model checking where the tableau is constructed from a given CTL property and a system model. Here we propose a workflow verification framework based on this technique. Our framework includes an automatic translator to translate a Petri net workflow model to the corresponding Kripke structure, and uses the one-pass tableau procedure for model checking. One-pass tableau-based decision procedures have been used for various logics, but to the best of our knowledge we are the first to use the one-pass tableau algorithm for model checking. We show the usefulness of our framework with a case study involving health service delivery.

The rest of this paper is organized as follows: Section 2 presents some related work; Section 3 presents some background topics; Section 4 discusses various components of the one-pass tableau-based workflow verification framework; Section 5 describes a case study, and Section 6 concludes the paper and offers some directions for future work.

## 2 Motivation and Related Work

There is no foundational, well recognized, or universally accepted formalism for workflow verification [7]. In [19], the authors discussed an automatic translation of workflow models into DVE, the specification language of the DiVinE model checker. The end result of their work is the NOVA WorkFlow [6] tool which uses the Compensable Workflow Modelling Language (CWML) for workflow modelling. Use of the DiVinE model checker limited the NOVA WorkFlow tool to LTL property specifications. An application of the SPIN model checker to workflow verification can be found in [22], and another approach using UPPAAL is available in [10]. Spin supports LTL, and UPPAAL supports Timed Computation Tree Logic. A tableau-based model checker for Temporal Description Logic (ALCT) can be found in [4] and a similar approach for Timed BDI$_{CTL}$ can be found in [15]. Both the ALCT and Timed BDI$_{CTL}$ model checkers use Petri nets to design workflows, but the Petri net models were translated manually to generate the state space for model checking.

Among the other workflow management systems, FlowMake [20] can identify structural conflicts in process models, YAWL [24] has some verification facilities mainly with respect to structural properties, AgentWork [16] uses dynamic rules to allow users to identify errors in the execution logic of the workflow while PLM$_{flow}$ [26] can generate workflow from business rules and can detect deadlocks.

Existing tools involve the usually difficult task of writing workflow models in the specification language of existing model checkers, or lack temporal verification capabilities, or are restricted to one property specification language. Tools and techniques to support workflow modelling and automatic verification in a single but flexible framework are needed.

According to [9], with suitable optimization techniques, tableau-based methods are potentially more flexible and efficient than other model checking approaches. The one-pass tableau strategy has been developed recently [1], and a naïve implementation of the one-pass tableau-based decision procedure for various logics [1] is available at [17]. While the worst-case complexity of the one-pass algorithm is 2EXPTIME which is worse than the EXPTIME complexity of the two-pass algorithm, in most practical situations, the worst case rarely arises; indeed the one-pass algorithm consistently outperforms the two-pass algorithm [8].

## 3 Preliminaries

In our framework, we use Petri nets to formally describe a workflow model, CTL to describe properties of the system, and the one-pass tableau-based model checking algorithm.

### 3.1 Workflow modelling using Petri nets

Worflow management systems (WfMSs) such as YAWL [24] provide users without any programming experience a relatively easy way to organize and/or describe complex processes in a visual format. Financial institutions, healthcare organizations, etc., involving complex processes, information and communication systems are adopting WfMSs to orchestrate the various activities. The main objective of workflow modelling is to provide an abstract view of a system to support analysis. Petri nets provide a graphical interface along with a strong mathematical foundation to accomplish this. A Petri net is a graph with two types of nodes—*places* and *transitions*. Arcs connect the two types, and no two nodes of the same type can be directly connected.

**Definition 1** (Petri net)**.** *A Petri net is a triple* $(P, T, F)$*, where:* $P$ *is a finite set of places,* $T$ *is a finite set of transitions* $(P \cap T = \emptyset)$*, and* $F \subseteq (P \times T) \cup (T \times P)$ *is a set of arcs (flow relation).*

Researchers developed various workflow patterns to facilitate the development of process-oriented applications. For this paper, we consider only the basic control flow patterns, namely sequential flow, parallel flow, conditional flow, and iterative flow [14]. These suffice to define many complex workflows. Our framework can handle Petri nets workflows containing all the four basic control flow patterns. Among the four basic patterns, conditional flow and iterative flow require special care. For example, in conditional flow, the token will follow one of the paths; in the iterative flow, the token will follow the same path multiple times. We used a variable called *guard* to restrict the token passing along one of the paths or along the same path. We also need to determine when to terminate an iterative flow which require updating a variable with each iteration. We call this updating step an *action*.

In Petri nets, the state space is given implicitly, but for formal verification, we need to generate the explicit states or markings. A marking $M$ of a Petri net is a function from the set of places $P$ to the non negative integers. Firing a transition $t$ in a Petri net with marking $M$, results in a new marking $M'$. A Petri net can be represented as a Kripke structure: the states are markings and there is a transition in the Kripke structure from $M$ to $M'$ whenever a transition in the Petri net creates a marking $M'$ from $M$.

[1]PDL, CTL, LTL, Modal Logic KD, KD45, K4, CK, K, S4, KLM Logic P, Intuitionistic Logic G4IP, and Propositional Classical Logic

**Definition 2** (Kripke Structure)**.** *A Kripke structure, over a set $AP$ of atomic propositions, is a 4-tuple $\mathcal{M} = \big(S, \rightarrow, L, I\big)$, where $S$ is a finite set of states, $\rightarrow \subseteq S \times S$ is a transition relation, $L : S \rightarrow 2^{AP}$ is an interpretation function, and $I \subseteq S$ is a set of initial states. $L(s)$ is the set of atomic propositions satisfied by $s$.*

## 3.2 Property specifications

We chose CTL as the property specification language. The syntax and semantics of CTL are available in [1]. In addition to propositional operators, CTL has path quantifiers, $A$ (all paths), $E$ (some paths), and temporal modalities, $G$ (all future states), $F$ (some future state), $X$ (the next state), $U$ (until) and $B$ (before). In a formula, a temporal operator must be preceded by a path quantifier. The inductive definition of CTL formulae in Backus Näur Form is given below:

$$\begin{aligned}\varphi \ ::= & \ \bot \mid \top \mid p \mid \neg\varphi \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid EX\ \varphi \mid \ AX\ \varphi \mid EF\ \varphi \mid AF\ \varphi \mid \\ & EG\ \varphi \mid AG\ \varphi \mid E[\varphi U \varphi] \mid A[\varphi U \varphi] \mid E[\varphi B \varphi] \mid A[\varphi B \varphi]\end{aligned}$$

where $p$ ranges over a set of atomic propositions. Given a Kripke structure $\mathcal{M}$ and a CTL formula $\varphi$, the semantics of CTL is defined as follows:

1. $\mathcal{M}, s \models \top$ and $\mathcal{M}, s \not\models \bot$.
2. $\mathcal{M}, s \models p$ if and only if $p \in L(s)$.
3. $\mathcal{M}, s \models \neg\varphi$ if and only if $\mathcal{M}, s \not\models \varphi$.
4. $\mathcal{M}, s \models \varphi_1 \wedge \varphi_2$ if and only if $\mathcal{M}, s \models \varphi_1$ and $\mathcal{M}, s \models \varphi_2$.
5. $\mathcal{M}, s \models \varphi_1 \vee \varphi_2$ if and only if $\mathcal{M}, s \models \varphi_1$ or $\mathcal{M}, s \models \varphi_2$.
6. $\mathcal{M}, s \models EX\ \varphi$ if and only if $\exists s' \in S,$ such that $s \rightarrow s'$ and $\mathcal{M}, s \models s'$.
7. $\mathcal{M}, s \models AX\ \varphi$ if and only if $\forall s' \in S,$ if $s \rightarrow s'$ then $\mathcal{M}, s \models s'$.
8. $\mathcal{M}, s \models EG\ \varphi$ if and only if there is a path $s_1 \rightarrow s_2 \rightarrow s_3 \rightarrow \cdots$, where $s_1 = s$, and for all $s_i$ along the path, we have $\mathcal{M}, s_i \models \varphi$.
9. $\mathcal{M}, s \models AG\ \varphi$ if and only if for all paths $s_1 \rightarrow s_2 \rightarrow s_3 \rightarrow \cdots$, where $s_1 = s$, and for all $s_i$ along the path, we have $\mathcal{M}, s_i \models \varphi$.
10. $\mathcal{M}, s \models EF\ \varphi$ if and only if there is a path $s_1 \rightarrow s_2 \rightarrow s_3 \rightarrow \cdots$, where $s_1 = s$ and for some $s_i$ along the path we have $\mathcal{M}, s_i \models \varphi$.
11. $\mathcal{M}, s \models AF\ \varphi$ if and only if for all paths $s_1 \rightarrow s_2 \rightarrow s_3 \rightarrow \cdots$, where $s_1 = s$ there is some $s_i$ such that $\mathcal{M}, s_i \models \varphi$.
12. $\mathcal{M}, s \models E[\varphi_1 U \varphi_2]$ if and only if there exists a path $s_1 \rightarrow s_2 \rightarrow s_3 \rightarrow \cdots$, where $s_1 = s$ and for some $s_i$ along the path $\mathcal{M}, s_i \models \varphi_2$ and $\forall j, j < i\ \ \mathcal{M}, s_j \models \varphi_1$.
13. $\mathcal{M}, s \models A[\varphi_1 U \varphi_2]$ if and only if for all paths $s_1 \rightarrow s_2 \rightarrow s_3 \rightarrow \cdots$, where $s_1 = s$ there exists $s_i$ along the path such that $\mathcal{M}, s_i \models \varphi_2$ and $\forall j, j < i\ \ \mathcal{M}, s_j \models \varphi_1$.
14. $\mathcal{M}, s \models E[\varphi_1 B \varphi_2]$ if and only if there exists a path $s_1 \rightarrow s_2 \rightarrow s_3 \rightarrow \cdots$, where $s_1 = s$ and for some $s_i$ along the path $\mathcal{M}, s_i \models \varphi_2$ implies $\exists j, j < i\ \ \mathcal{M}, s_j \models \varphi_1$.

15. $\mathcal{M}, s \models A[\varphi_1 B \varphi_2]$ if and only if for all paths $s_1 \rightarrow s_2 \rightarrow s_3 \rightarrow \cdots$, where $s_1 = s$ there exists $s_i$ along the path such that $\mathcal{M}, s_i \models \varphi_2$ implies $\exists j, j < i \;\; \mathcal{M}, s_j \models \varphi_1$.

We say an *elementary formula* is a formula of the form $p$, $\neg p$, $EX\varphi$ or $AX\varphi$ where $p$ is an atomic proposition and $\varphi$ is a CTL formula. The classification of non-elementary formulae is shown in Table 1 using Smullyan's $\alpha$- and $\beta$-notation.

Table 1: Smullyan's $\alpha$- & $\beta$-notation for CTL

| $\alpha$ | $\alpha_1$ | $\alpha_2$ |
|---|---|---|
| $\varphi \wedge \psi$ | $\varphi$ | $\psi$ |
| $EG\ \varphi$ | $\varphi$ | $EX\ EG\ \varphi$ |
| $AG\ \varphi$ | $\varphi$ | $AX\ AG\ \varphi$ |
| $E[\varphi B \psi]$ | $\neg\psi$ | $\varphi \vee EX\ E[\varphi B \psi]$ |
| $A[\varphi B \psi]$ | $\neg\psi$ | $\varphi \vee AX\ A[\varphi B \psi]$ |

| $\beta$ | $\beta_1$ | $\beta_2$ |
|---|---|---|
| $\varphi \vee \psi$ | $\varphi$ | $\psi$ |
| $EF\ \varphi$ | $\varphi$ | $EX\ EF\varphi$ |
| $AF\ \varphi$ | $\varphi$ | $AX\ AF\ \varphi$ |
| $E[\varphi U \psi]$ | $\psi$ | $\varphi \wedge EX\ E[\varphi U \psi]$ |
| $A[\varphi U \psi]$ | $\psi$ | $\varphi \wedge AX\ A[\varphi U \psi]$ |

## 3.3 Tableau-based satisfiability checking for CTL

Tableau systems obey the subformula principle - all formulae occurring in a tableau proof are subformulae of the formula being proved. Subformulae are obtained by applying a set of rules based on the semantics of the particular logic. Applications of these rules forms a tree, called the tableau. For propositional logic, tableau-based procedures include the following steps: to show a formula $\varphi$ is valid we try to show its negation $\neg\varphi$ is not satisfiable, i.e., there is no assignment of truth values to propositional variables in $\neg\varphi$ to make it *true.* The expression $\neg\varphi$ is decomposed into subformulae by applying tableau expansion rules. If a branch of the tableau contains a pair of contradictory formulae (i.e., $\psi$ and $\neg\psi$), then this branch is marked as *closed.* The tree construction stops when all the branches close or there is no other formula to which a tableau rule can be applied. An open branch in a completed tableau (rules have been applied to all the formulae) gives a counter example, i.e, an assignment which satisfies $\neg\varphi$. If all branches close, the tableau is said to be *closed* and $\varphi$ is declared valid.

In CTL tableau, Boolean connectives are handled the same way as in propositional logic, and temporal connectives are handled by decomposing them into a requirement on the "current state" and a requirement on "the rest of the sequence" [25]. Implementing the tableau rules will cause some branches of the tableau to loop forever. However, the tableau can be made finite by identifying nodes containing the same set of formulae. A formula $\varphi$ of the form $EF\varphi, AF\varphi, E[\varphi\ U\ \psi]$, or $A[\varphi\ U\ \psi]$ is called an *eventuality formula* [1]. Eventuality formulae state "something will happen eventually in the future". To guarantee validity of a CTL formula $\varphi$, in addition to checking for propositional inconsistencies, we need to check for unsatisfiability of all the eventuality formulae. The tableau rules for CTL can be categorized into four categories: the $\alpha$ rules, the $\beta$ rules, the $X$ rule, and the terminal rules. The $\alpha$ rules are for the conjunctive operators (i.e., $\wedge$, $EG$, $AG$, $EB$, and $AB$) and each creates one child (e.g., the rule for $EG\varphi$ creates a child with the formulas $\varphi$ and $EXEG\varphi$). The $\beta$ rules are for the disjunctive operators (i.e., $\vee$, $EF$, $AF$, $EU$ and $AU$) and each creates two children (e.g., the rule for $AF\varphi$ creats a child with $\varphi$ and a child with $AXAF\varphi$). The $X$ rule is for the $X$ operator and the number of children created is dependent on the number of $EX$ formulae in a node. The $X$ rule states that if there is $\{EX\phi_1, \cdots, EX\phi_n, AX\psi_1, \cdots, AX\psi_m\}$ in a node then there are $n$ children of the node, with $\{\phi_1, \psi_1, \cdots, \psi_m\}$ at child 1, $\{\phi_2, \psi_1, \cdots, \psi_m\}$ at child 2, and so on.

There are two terminal rules, one (known as the *id* rule) is applied when there is a contradiction on a branch and the other (known as the *block* rule) is applied when expanding a branch causes a loop. If a node $m$ is about to be created as a child of a node $n$ and there is an ancestor $n_0$ of $n$ having the same set of formulae in $m$, then $m$ is not created, and $n$ and $n_0$ are connected with a feedback edge (these situations causes loops in the tableau).

A Hintikka structure for $\varphi$ is a finite partial representation of a model for $\varphi$. A formal discussion on Hintikka structures for CTL formulae can be found in [1], [2]. The tableau procedure is a systematic search for a Hintikka structure; to determine the satisfiability of $\varphi$ the tableau shows there is no Hintikka structure for $\neg\varphi$.

The two-pass tableau-based decision procedures [2], [3] test the satisfiability of a CTL formula $\varphi$ in two steps or "passes". In the first step, it constructs a tableau $\mathcal{T}_\varphi$, for $\varphi$, by applying tableau construction rules. If any Hintikka structure satisfies $\varphi$, then there is at least one represented by $\mathcal{T}_\varphi$ [8]. In the second step, inconsistent nodes (nodes that cannot be a part of any Hintikka structure for $\varphi$) are identified. The second pass uses an algorithm known as the marking algorithm to mark the inconsistent nodes; the marking algorithm marks the root of the tableau if there is no Hintikka structure for the input formula [2]. Termination of the one-pass tableau procedure is guaranteed [11].

The one-pass tableau algorithm for CTL [1] uses a single pass to determine the satisfiability of a formula. Instead of constructing the tableau first and then finding the Hintikka structure, the one-pass tableau determines the existence of a Hintikka structure while constructing the tableau, through the use of a *history* and a *variable* associated with each tableau node. In the one-pass tableau, loops are determined by looking at the *history* of the current node; the *history* is passed from parent to child. The *variable* propagates information about the unsatisfiable eventualities from a child to its parent. The history of a node is calculated while applying a tableau rule. The variable of a terminal node is calculated according to the terminal rule when a branch of the tableau terminates, and propagated upward.

The tableau rules for the one-pass tableau for satisfiability checking are available in [1]. A tableau node, in a one-pass tableau, contains three components - a set of formulae $\Gamma$, a history *Fev* and *Br*, and a variable *uev*, the three are $\Gamma$ :: *Fev*, *Br* :: *uev*, where the symbol "::" separates the three components. Here, *Fev* keeps track of the satisfiable eventualities, *Br* keeps track of the formulae that may create loops and is used by one of the terminal rules to identify loops, and *uev* keeps track of the unsatisfiable eventualities. The *uev* of a node is set to $\{(\mathit{false}, m)\}$ if both branches created by a $\beta$ rule are *closed* due to propositional inconsistencies; the *uev* is set to the empty set if there are no unsatisfiable eventualities or propositional inconsistencies in any of the children; the *uev* is set to the set of all the unsatisfiable eventualities of the children if both children have unsatisfiable eventualities; and finally the *uev* is set to the set of all unsatisfiable eventualities of a child if either of the two children has unsatisfiable eventualities.

The one-pass procedure starts with the negation of the formula of interest (the input formula) in negation normal form (NNF) as the root and the rules are applied on the root to construct the tableau. If, after the construction, the *uev* of the root is not empty, we call it a closed tableau. A closed tableau means the negation of the input formula is not satisfiable, i.e., the input formula is valid. On the other hand, if the uev of the root is the empty set we say the tableau is open, meaning the negation of the input formula is satisfiable, and the input formula is not valid. We discuss one of the rules, namely the *AF* rule below. A detailed discussion of the one-pass tableau rules may be found in [11].

$$(AF) \qquad \frac{AF\varphi; \Gamma :: Fev, Br :: uev}{\varphi; \Gamma :: \{AF\varphi\} \cup Fev, Br :: uev_1 \mid AXAF\varphi; \Gamma :: Fev, Br :: uev_2}$$

The *uev* of the parent is calculated from the *uev₁* and *uev₂* of the children as follows:

$$uev = \begin{cases} uev_1 & \text{if } uev_2 = \{(\mathit{false}, m)\} \\ uev_2 & \text{if } uev_1 = \{(\mathit{false}, m)\} \\ \{(AF\varphi, n)\} & \text{otherwise} \end{cases}$$

Here, $n = max\big(f(AF\varphi, uev_1) \cup f(AF\varphi, uev_2)\big)$. The function $f(AF\varphi, uev')$ returns the index of $AF\varphi$ in *uev'*. We show an example of the $AF$ rule in Fig. 1. In node $n_6$, *Fev* and *Br* came from its predecessor (not shown in the figure). An application of the $AF$ rule on $n_6$ creates nodes $n_7$ and $n_8$. The *id* rule is applied on $n_7$ and the *block* rule is applied on $n_8$. We show $n_6$ with the value of *uev* calculated from its children. For $n_6$, $uev_1 = \{(\mathit{false}, 1)\}$ and $uev_2 = \{(AF\varphi, 0)\}$. As $uev_1 = \{(\mathit{false}, 1)\}$, the *uev* of $n_6$ is set to $uev_2$ which is $\{(AF\varphi, 0)\}$.

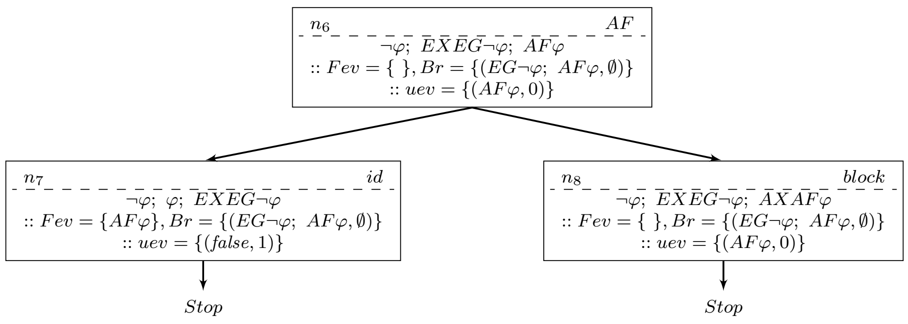

$n_6$ $AF$
$\neg\varphi;\ EXEG\neg\varphi;\ AF\varphi$
$:: Fev = \{\ \}, Br = \{(EG\neg\varphi;\ AF\varphi, \emptyset)\}$
$:: uev = \{(AF\varphi, 0)\}$
$n_7$ $id$
$\neg\varphi;\ \varphi;\ EXEG\neg\varphi$
$:: Fev = \{AF\varphi\}, Br = \{(EG\neg\varphi;\ AF\varphi, \emptyset)\}$
$:: uev = \{(\mathit{false}, 1)\}$
$n_8$ $block$
$\neg\varphi;\ EXEG\neg\varphi;\ AXAF\varphi$
$:: Fev = \{\ \}, Br = \{(EG\neg\varphi;\ AF\varphi, \emptyset)\}$
$:: uev = \{(AF\varphi, 0)\}$
*Stop*
*Stop*

Figure 1: Illustration of the AF rule.

## 4 The One-Pass Tableau-Based Model Checking

A generic framework using the tableau method for model checking is described in [9]. The main idea of tableau-based model checking is that given a Kripke structure $\mathcal{M}$ with initial state $s_0$ and a property $\varphi$, the algorithm simulates the construction of the tableau in $\mathcal{M}$. The construction starts from $s_0$ and moves forward along the transitions in $\mathcal{M}$. More specifically,

1. The tableau construction starts with $(\neg\varphi, s_0)$ where $s_0 \in S$.
2. The tableau construction is similar to the tableau for satisfiability. However, the construction rules must be modified so that the tableau nodes are properly associated with the states of $\mathcal{M}$.
3. When the tableau construction is completed, $\mathcal{M}$ satisfies $\varphi$ iff the final tableau with $(\neg\varphi, s_0)$ is a closed tableau.

We show the modifications required to use the one-pass tableau procedure to perform model checking.

**Definition 3** (Closure of a formula $cl(\varphi)$ in CTL)**.** *The closure $cl(\varphi)$ of a formula $\varphi$ is the least set of formulae such that:*

*1.* $\top, \varphi \in cl(\varphi)$*;*

2. *$cl(\varphi)$ is closed under taking subformulae;*

3. *if $\psi \in cl(\varphi)$ and $\psi$ does not begin with $\neg$ then $\neg\psi \in cl(\varphi)$;*

4. *$cl(\varphi)$ is closed under taking all components of $\alpha-$formulae and $\beta-$formulae.*

Given a Kripke structure $\mathcal{M}$, and a CTL formula $\varphi$, $L[cl(\varphi), s] := \big(L(s) \cup \{\neg p \mid p \in AP \setminus L(s)\}\big) \cap cl(\varphi)$. It can be shown that $L[cl(\varphi), s]$ consists of a set of atomic propositions and the negation of atomic propositions only. For example, given the Kripke structure in Figure 2, and a CTL formula $\varphi = AG(p \rightarrow AFq)$, $L[cl(\varphi), s] = \{p, q\}$.

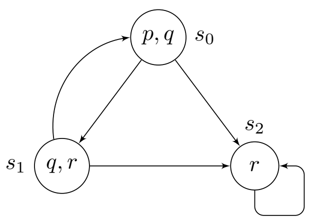


Figure 2: A simple Kripke structure with three states.

In the one-pass tableau-based model checking, the tableau construction starts with $(s_0, \neg\varphi \cup L[cl(\neg\varphi), s_0])$. Here, $s_0$ is the initial state of the model and $\varphi$ is the CTL property to be verified. In model checking, all the tableau rules are same as the tableau for satisfiability checking except for the $X$ rule. The $X$ rule which deals with formulae referring to the "next state", is modified to denote a transition from one state to another. The next states of a state can be identified from the given transition relation of the Kripke structure. The one-pass tableau model checking algorithm given in Algorithm 1. The tableau construction starts from the initial states and to reduce the branching, the $\alpha$ rules are applied before the $\beta$ rules. The $\alpha$ and $\beta$ rules do not make transitions from one state to another. The $X$ rule is applied when a tableau node has only $EX$ and $AX$ formulae to which to apply tableau rules. In the tableau-based satisfiability checking, the number of children of a node having $EX$ and $AX$ formulae depends on the number of $EX$ formulae in the node. In model checking, the number of children is determined from the number of $EX$ formulae and the number of states adjacent to the current state. If the constructed tableau is closed then the property $\varphi$ is declared *true*; otherwise, an open branch of the tableau shows a counter example.

## 4.1 System description

We implemented the tableau-based model checking framework using the C++ programming language. We used the CPN tool to design the Petri net workflow models. The CPN tool stores the Petri net in the Petri net markup language (PNML) which is an XML-based interchange format for Petri nets. Our framework could be modified to deal with any Petri net graphical editor with an XML based interchange format. The top-level structure of the tableau-based model checking framework is shown in Fig. 3 and a brief discussion of each component is given below.

**The XML Parser:** Most of the information in the XML file generated by the CPN tool is editor specific information, such as: the position of the nodes, the size of the nodes, and colours of different parts, etc. Our XML Parser reads the input PNML file from the beginning to the

**Algorithm 1** Given a *Kripke structure* and a *property*, this algorithm generates a tableau using the one-pass tableau procedure

```
root ← tableauNode(property ∪ $L[cl(property), state]$ , state) {Here, tableauNode($\varphi$, $s_0$) is a constructor that creates a tableauNode with $\varphi$ in the formulaList and $s_0$ in the stateSpaceID}
nodeStack.push(root)
while one-pass tableau rules have not been applied to all the nodes in nodeStack do
  tempTableauNode ← nodeStack.pop()
  if the id rule is applicable to tempTableauNode then
    apply the id rule
  else if a linear rule is applicable to tempTableauNode then
    newTableauNode ← tableauNode($\{\alpha_1, \alpha_2\} \cup \Gamma$, $s_i$) {Here, $\Gamma$ is the set of formulae in tempTableauNode and $s_i$ is the stateSpaceID in tempTableauNode}
    nodeStack.push(newTableauNode)
  else if a universal branching rule is applicable to tempTableauNode then
    newTableauNode1 ← tableauNode($\{\beta_1\} \cup \Gamma$, $s_i$)
    nodeStack.push(newTableauNode1)
    newTableauNode2 ← tableauNode($\{\beta_2\} \cup \Gamma$, $s_i$)
    nodeStack.push(newTableauNode2)
  else if an existential branching rule is applicable to tempTableauNode then
    adjList ← all the states adjacent to $s_i$ in the Kripke structure
    for all $s_j \in$ adjList do
      newTableauNode ← tableauNode($\Delta \cup \psi \cup L[cl(property), s_j]$, $s_j$) {Here, tempTableauNode has a formula of the form $EX\psi; AX\Delta$}
      nodeStack.push(newTableauNode)
    end for
  else
    apply the block rule
  end if
end while
```

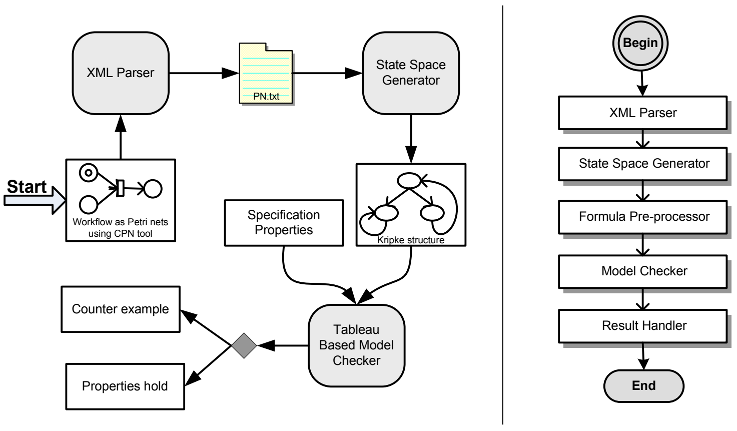


Figure 3: The components of the tableau-based model checking framework.

end and extracts the information related to the Petri net places and transitions and stores, and stores it in a simple text file.

**The State Space Generator:** The State Space Generator loads the Petri net model from the simple text file generated by the XML Parser. The Petri net is represented in the memory as a list of transitions. Two types of information control a transition firing, a guard and an action. If we have a loop in the workflow model then a set of transitions are fired until the guard becomes *false*. A guard represents a condition of the from *variable op value*, where $op \in \{==, !=, <, >, \leq, \geq\}$. An action represents an assignment of the form $variable = exp$, where *exp* is a *variable* or of the form *variable* $op_1$ *value*, and $op_1 \in \{+, -, *\}$. A transition can have an action and/or a guard, whereas a place can have only an action. Actions associated with transitions can change the value of a variable in the output places. After loading the Petri net model from the text file, the Kripke structure is generated by simulating the Petri net[11].

**The Formula Pre-processor:** The Formula Pre-processor module applies four pre-processing steps before applying the tableau model checking algorithm. The first step is to rewrite the $U$ and $B$ operators in the given property. For example, a formula of the form $E[\varphi\ U\ \psi]$ is written as $(\varphi\ EU\ \psi)$, this makes it easier to generate a parse tree for the formula. The second pre-processing step is to generate a parse tree for the property. Using parse trees provides two benefits: we can easily identify the first operator to apply a tableau rule to (it is the root of the parse tree) and easily identify the subformulae. The third pre-processing step is to change the input property $\varphi$ to $\neg\varphi$. The final pre-processing step is to convert $\neg\varphi$ to NNF.

**The Model Checker:** The Model Checker module uses two functions − a state space handler to manage the Kripke structure and a property handler to manage the property parse tree. The Model Checker module implements the one-pass tableau model checking algorithm according to the previous discussion.

**The Result Handler:** The purpose of the Result Handler module is to show the output of the model checker in a readable format. Currently, our model checker can show the output as a list of tableau nodes which can be transformed into a tree structure by hand. If a property does not hold, we can get a counter example by investigating an open branch.

First, we implemented the one-pass tableau algorithm for satisfiability checking and tested our implementation with a comprehensive set (41 in total) of CTL formulae available at [17]; see the results in [11]. Then we modified the one-pass tableau algorithm for model checking. We used the Mahone cluster (a parallel cluster of 134 nodes with 64GB RAM per node) of ACEnet [2] to run our experiments.

## 5 Case Study

Our research is part of a collaboration among academic researchers, an industry partner and the local health authority to develop innovative workflow tools for health services delivery [13], [6]. Healthcare workflows are developed from guidelines or best practises defined by healthcare professionals. Such guidelines are processes describing the activities for providing treatment to a patient. Using the CPN tool, we modeled a workflow following the national Hospice Palliative Care (HPC) guideline [3] and used our tool model check some properties, some of which are listed

[2]ACEnet: http://www.ace-net.ca
[3]Canadian HPC association: http://www.chpca.net/

below.

HPC refers to the comfort care that reduces the severity of a disease rather than providing a cure. For example, if surgery cannot be performed to remove a tumour, radiation treatment might be tried to reduce its rate of growth, and pain management could help the patient manage physical symptoms. The HPC guideline containing 51 tasks is depicted in Fig. 4; the Petri net model contains 55 places and 51 transitions, and the corresponding workflow state space graph consists of 67 states. We verified the following properties of the model:

**Property 1:** $AF\,end\_of\_workflow$
The `end_of_workflow` will always be reached.

**Property 2:** $AG(error\_in\_therapy \rightarrow EF\,report\_to\_supervisor)$
Any error in therapy is always reported to the supervisor.

**Property 3:** $AG(prepare\_care\_plan \rightarrow AF\,present\_care\_plan)$
After a care plan is prepared, it is always presented to the patient.

**Property 4:** $AG\neg(\neg define\_limits\_of\_conf. \wedge share\_accurate\_info)$
Limits of confidentiality are always defined before information is shared.

The verification results are summarized in Table 2. Further experiments may be found in [11], including experiments on verification of large models, and an example of a smaller model with a failed property and the output of a counter model. In our current implementation the output of counter models for large Petri nets is difficult for the human eyes to read.

Table 2: Property verification results of the one-pass tableau model checker

| **Property** | **Time** (in sec) | **No. of tableau nodes** | **Memory** (in MB) | **Valid** |
|---|---|---|---|---|
| Property 1 | 3.887 | 2266 | 8.1 | Yes |
| Property 2 | 13.931 | 4968 | 16.5 | Yes |
| Property 3 | 12.389 | 4778 | 15.9 | Yes |
| Property 4 | 11.399 | 4920 | 16.4 | Yes |

## 6 Conclusion and Future work

In this paper we presented a model checking framework for workflow model verification. We used the one-pass tableau method for model checking which can efficiently verify properties for a large workflow model. The framework can be improved and extended easily as the architecture can be adapted to support different workflow modelling languages (e.g., YAWL [24]) as well as a variety of property specification languages, such as LTL, timed CTL, other modal logics such as BDI logics (logics for beliefs, desires, and intentions) as needed to verify various aspects of large scale enterprise information systems. The latter modifications in general simply require the addition and/or replacement of one-pass tableau rules. While there are other approaches to model checking Petri nets (e.g., LoLa [18] and Fast [5]) these in general lack flexibility as they use a fixed specification language for property specifications. We presented an implementation of the framework, but a number of improvements are possible. The tree structure, generated

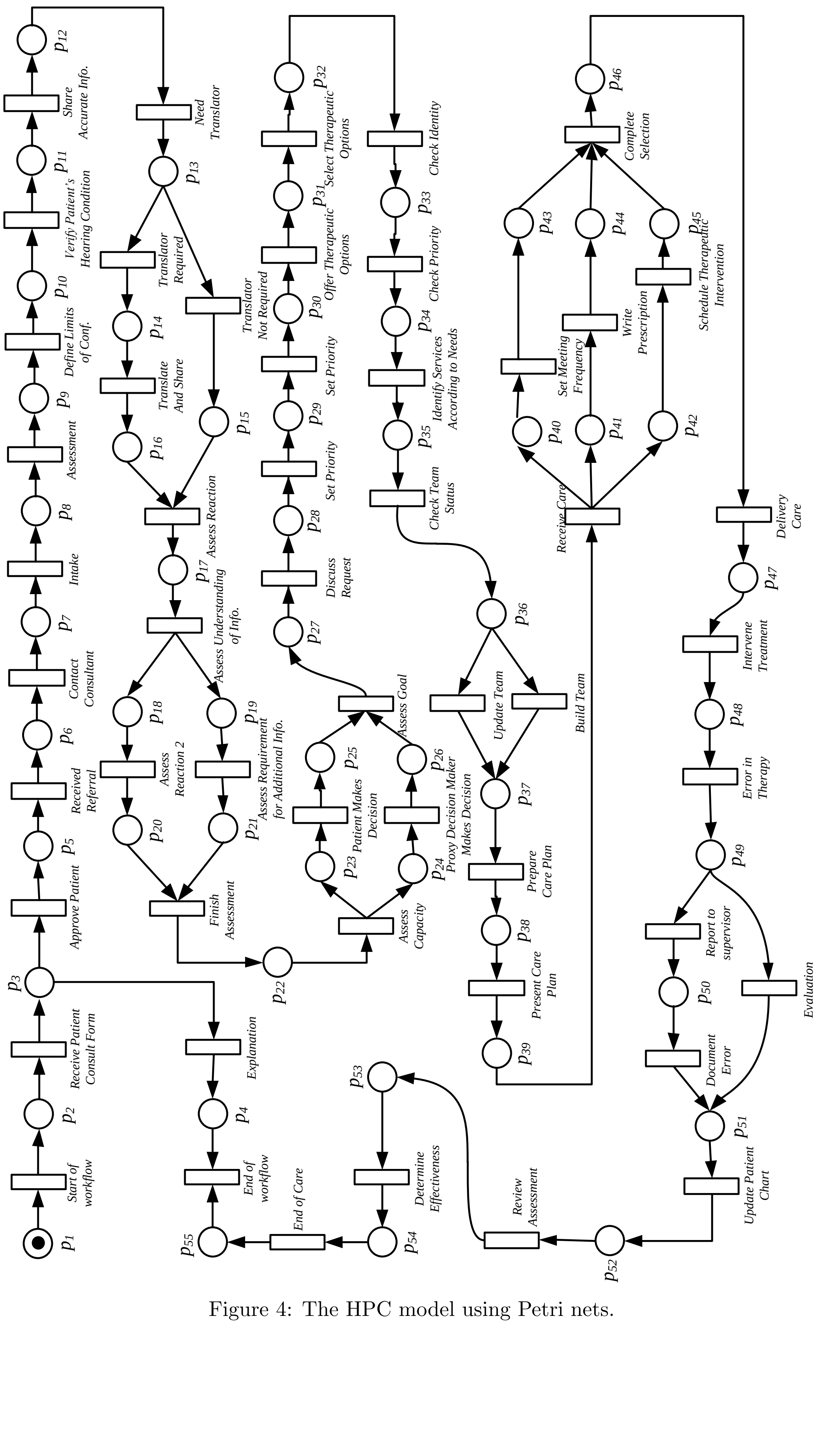


Figure 4: The HPC model using Petri nets.

by the tableau model checking algorithm, is not possible to show on the console output. A graphical user interface showing the tableau would help the user better analyse the counter models. Another improvement would be to apply high performance computing techniques to increase the efficiency of the model checking algorithm. In the one-pass tableau method, only one branch of the derivation tree needs to be considered at any stage, making it suitable to implement on a bank of parallel processors [1]. In [21], [12], the authors discussed two approaches to parallel temporal tableau for LTL using the two-pass tableau procedure; these need investigation for the one-pass method. The first approach applies parallelism by dividing the sequential algorithm into separate sub-problems and distributing the sub-problems to different processors. In this approach communication between the processes are maintained using two shared queues. The second approach does not use any shared queue, hence the inter-process communication increases. However, dividing into sub-problems does not ensure equal load distribution across the processors, because one sub-problem may take less time than the others, i.e., some processors will remain idle while the others are working. The inter-process communication increases in the second approach. In literature, experiments show that the second approach performs better than the first approach for LTL [21]. Many optimization techniques including unit propagation, simplification, and backjumping have been developed for tableau-based modal logic systems which can be applied to the one-pass tableau-based model checking algorithm to enhance performance. Extensions of the method to include timing information are fairly straightforward [15].

**Acknowledgements:** The authors were supported by an NSERC Discovery Grant, by the Atlantic Canada Opportunities Agency, and by an ACEnet Graduate Research Fellowship and benefited from many discussions with clinicians from the Guysborough Antigonish Strait Health Authority (GASHA).

## References

[1] P. Abate, R. Goré, and F. Widmann. One-Pass Tableaux for Computation Tree Logic. In *Proc. of the 14th Int. Conference on Logic for Programming, Artificial Intelligence and Reasoning*, LPAR'07, pages 32–46, 2007.

[2] M. Ben-Ari, A. Pnueli, and Z. Manna. The Temporal Logic of Branching Time. In *Proc. of the 8th ACM SIGPLAN-SIGACT symposium on Principles of programming languages(POPL '81)*, volume 20, pages 164–176, 1981.

[3] E. M. Clarke and E. A. Emerson. Design and Synthesis of Synchronization Skeletons Using Branching-Time Temporal Logic. In *Proc. of Logic of Programs, Workshop*, volume 131, pages 52–71, 1982.

[4] J. Dallien, W. MacCaull, and A. Tien. Initial Work in the Design and Development of Verifiable Workflow Management Systems and Some Applications to Health Care. In *Proc. of the 5th Int. Workshop on Model-based Methodologies for Pervasive and Embedded Software, Co-located with ETAPS 2008*, pages 78–91, 2008.

[5] Laboratoire Spécification et Vérification (LSV). `http://www.lsv.ens-cachan.fr/Software/fast/`.

[6] Centre for Logic and Informatoin (CLI). `http://logic.stfx.ca/`.

[7] S. Giro. Workflow Verification: a New Tower of Babel. In *Proc. of the Int. Modelling and Simulation Multiconference (IMSM 2007)*, 2007.

[8] V. Goranko, A. Kyrilov, and D. Shkatov. Tableau Tool for Testing Satisfiability in LTL: Implementation and Experimental Analysis. *Electronic Notes in Theoretical Computer Science (ENTCS)*, 262, 2010.

[9] Valentin Goranko. Temporal Logics for Specification and Verification. In *Proc. of the European Summer School in Logic, Language and Information (ESSLI'09)*, 2009.

[10] V. Gruhn and R. Laue. Using Timed Model Checking for Verifying Workflows. In *Proc. of Computer Supported Activity Coordination (2005)*, pages 75–88, 2005.

[11] M. Z. Islam. A tableau-based workflow verification framework for computation Tree Logic (CTL). Master's thesis, St. Francis Xavier University, Antigonish, Canada, 2012.

[12] R. Jonathon. A Blackboard Approach to Parallel Temporal Tableaux. In *In Proc. of Artificial Intelligence, Methodologies, Systems and Applications*, 1994.

[13] W. MacCaull, H. Jewers, and M. Latzel. Using an Interdisciplinary Approach to Develop a Knowledge-driven Careflow Management System for Collaborative Patient-centred Palliative Care. In *In Proc. of the 1st ACM Int. Conference on Health Informatics*, 2010.

[14] Workflow Management Coalition. Workflow Management Coalition Terminology & Glossary, 1999. Document Number WFMC-TC-1011.

[15] K. Miller and W. MacCaull. Verification of Careflow Management Systems with Timed BDI$_{CTL}$ Logic. In *Proc. of Int. Workshops on Business Process Management (BPM 2009)*, volume 43, pages 623–634, 2010.

[16] R. Müller, U. Greiner, and E. Rahm. AgentWork: a workflow system supporting rule-based workflow adaptation. *Data & Knowledge Engineering*, 51:223–256, 2004.

[17] Research School of Information Sciences and Engineering (RSISE). The Tableau Workbench. `http://twb.rsise.anu.edu.au/demolist/`.

[18] Theory of Programming Languages and Programming. `http://www.informatik.uni-rostock.de/tpp/lola/`.

[19] F. Rabbi, H. Wang, and W. MacCaull. YAWL2DVE: An Automated Translator for Workflow Verification. In *Proc. of the 4th IEEE Int. Conference on Secure Software Integration and Reliability Improvement (SSIRI '10)*, pages 53–59, 2010.

[20] W. Sadiq and M. E. Orlowska. Applying Graph Reduction Techniques for Identifying Structural Conflicts in Process Models. In *Proc. of the 11th Int. Conference on Advanced Information Systems Engineering (CAiSE '99)*, pages 195–209, 1999.

[21] R. I. Scott, M. D. Fisher, and J. A. Keane. Parallel Temporal Tableaux. In *In Proc. of the 4th Int. Euro-Par Conference on Parallel Processing*, 1998.

[22] M. C. Stolz. Verification of Workflow Control-Flow Patterns with the SPIN Model Checker. Master's thesis, University of Bern, Switzerland, 2010.

[23] The CPN tool website. `http://www.cpntools.org/`.

[24] W. M. P. van der Aalst and A. ter Hofstede. YAWL: yet another workflow language. *Journal of Information Systems*, 30(4):245–275, 2005.

[25] P. Wolper. The Tableau Method for Temporal Logic: An Overview. *Logique et Analyse, 28(110–111)*, pages 119–136, 1985.

[26] L. Zeng, D. Flaxer, H. Chang, and J. J. Jeng. PLM$_{flow}$-Dynamic Business Process Composition and Execution by Rule Inference. In *Proc. of the 3rd Int. Workshop on Technologies for E-Services (TES '02)*, pages 141–150, 2002.

# Initial Experiments with External Provers and Premise Selection on HOL Light Corpora

Cezary Kaliszyk
University of Innsbruck
Innsbruck, Austria
cezarykaliszyk@gmail.com

Josef Urban*
Radboud University Nijmegen
Nijmegen, The Netherlands
Josef.Urban@gmail.com

**Abstract**

This paper reports our initial experiments with using external ATP and premise selection methods on some corpora built with the HOL Light system. The testing is done in three different settings, corresponding to those used earlier for evaluating such methods on the Mizar/MML corpus. This is intended to provide the first estimate about the usefulness of such external reasoning and AI systems for solving problems over HOL Light and its libraries.

## 1 Motivation

Usage of external first-order ATPs like Vampire [22], E [24], SPASS [30], and recently also SMT solvers like Z3 [6] for ITP-based (large-theory) formalization has been developed quite significantly in the recent decade. Particularly in the Isabelle community, the Isabelle/Sledgehammer [3, 2] bridge to such external tools is getting increasingly popular. This helps to further develop various parts of the technology involved. ATPs have recently gained the ability to quickly load large theories over large signatures and work with them. Methods for automated selection of relevant knowledge and for proof guidance are actively developed, together with specialized automated systems targeted at particular mathematical domains. Formats and translation methods handling more formalization-friendly foundations are being defined, and metasystems that decide which ATP, translation method, strategy, parallelization, and premises to use to solve a given problem with limited resources are being designed. Cooperation of humans and computers over large corpora of formal knowledge is an interesting field, allowing exploration of new AI systems and combinations of different AI techniques that can attempt to encode concepts like analogy and intuition, and rigorously evaluate their usefulness. Perhaps not only Hilbert and Turing, but also the formality-opposing and intuition-oriented Poincaré[1] [21] would have been interested to learn about the new “semantic AI paradise” of such large corpora of formal and computer-understandable mathematics (from which we do not intend to be expelled).

The HOL Light [10] system is probably the first among the existing well-known ITPs which has integrated and extensively used a general ATP procedure, the MESON tactic [11]. Hurd has developed and benchmarked early bridges [12, 13] between HOL and external systems, and his Metis system [14] has also become a significant part of the Isabelle/Sledgehammer bridge to ATPs [20]. Using the very detailed Otter/Ivy [18] proof objects, Harrison also later implemented a bridge from HOL Light to Prover9 [17].

HOL Light however does not yet have a general bridge to large-theory ATP/AI methods, similar to Isabelle/Sledgehammer or Miz$\mathbb{AR}$ [27, 28], which would attempt to automatically

*Supported by the NWO projects “MathWiki” and “Learning2Reason”

[1]2012 is not just the year of Turing [9], but also of Poincaré, whose ideas about creativity and invention involving random, intuition-guided exploration confirmed by critical evaluation quite correspond to what AI metasystems like MaLARea [29] try to emulate in the large-theory formal setting.

solve a new goal by selecting the relevant knowledge from the large library and running (possibly several) external ATPs on such (possibly several alternative) premise selections. HOL Light seems to be a natural candidate for adopting such methods, because of the amount of work already done in this direction mentioned above, and also thanks to HOL Light's foundational closeness to Isabelle/HOL. Also, it seems that thanks to the Flyspeck project [8], HOL Light is becoming less of only a "single, very knowledgable formalizer" tool, and also getting increasingly used as a "tool for interested mathematicians" (particularly Vietnamese[2]) that know the large libraries much less and have much less experience with crafting their own targeted proof tactics. For such ITP users it is good to provide a small number of strong methods that allow fast progress.

The work reported here consists of several experiments intended to give an initial information about the usefulness of building such a bridge for HOL Light. The evaluation tries to follow the pattern introduced for Mizar/MML in [25, 26]:

1. Evaluate the ATP efficiency on simple ITP steps ("by" in Mizar, MESON in HOL Light).
2. Evaluate the ATP efficiency on re-proving whole theorems in the libraries from their (as exact as possible) proof dependencies.
3. Evaluate the ATP efficiency on proving whole theorems when the premises are chosen from the large library by AI (heuristic, learning) methods.

In general, the work is much less complete and polished than the similar work done for Mizar and Isabelle, and also in much rawer state than the finished work by Harrison and Hurd mentioned above. The first issues are now efficiency and encoding of the export from HOL Light to FOL, and also the compatibility and alignment of the data that we use for re-proving of whole theorems from their dependencies, and from trained advice. But we hope that obtaining the initial results and reporting them and the problems encountered could attract some interest and expert advice with such technical issues, so that the bridges are finished (not necessarily by us) sooner, and the HOL Light and Flyspeck large mathematical corpora become available to ATP and AI research in the same way as the Mizar and Isabelle corpora.

Apart from the attempt to inspire in this section, the rest of the paper is organized as follows. Section 2 explains how problems in the above categories were prepared, building on the work of Harrison and Adams. Section 3 reports the experiments and results, and Section 4 concludes.

## 2 Problem Exports

All three kinds of export described below initially rely on using parts of the MESON tactic for exporting the problems to the TPTP FOF format. MESON is based on the model elimination method invented by Loveland [15] and later combined with Prolog-like search tree [16]. The implementation of MESON in HOL-Light first applies a number of tactics that transform the HOL goal to a FOL goal (or multiple goals). The FOL goal is then passed to an ML procedure that returns a proof which is later replayed using HOL Light proof steps. This often means that multiple ATP problems are created from one such MESON call (due to pre-processing), and the formulas are already skolemized. The MESON-based export can become very slow for larger problems, probably depending on the use of higher-order features in the problems. However the export still provides a sufficient number of problems for the first evaluation. Our plan is

[2]`http://weyl.math.pitt.edu/hanoi2009/Participants/`

to later switch to TFF1-based (extension of FOF with types and polymorphism) export [4], for which for example Why3 [7] already has a usable translation tool to FOF by Andrei Paskevich.

Even though MESON uses CNF, we encode it as FOF to get around some syntactic issues with TPTP, and also because large-theory systems have a longer tradition of working on the FOF level. In each problem, apart from the TPTP formulas themselves, we keep as a comment also the original HOL Light goal and assumptions, and their first-order encoding used internally by the MESON tactic. This is intended to help debugging the translation, and we invite interested readers to check that we proceed (at least for most problems) correctly. For the problems created from the full theorems, we additionally keep the name of the theorem inside the problem.

## 2.1 Exporting problems from the original MESON calls

We have first hooked the exporting code into the MESON (and ASM_MESON) tactic itself, to get a large number of TPTP problems corresponding to the HOL Light problems on which the MESON tactic is used. There does not seem to be any issue running this export, so we ran it fully on the core HOL Light, HOL Multivariate, and Flyspeck corpora. The problems created are available online.[3] From core HOL Light this yields 2057 problems, from HOL Multivariate 12428, and from Flyspeck 19634. Their average, minimum, and maximum sizes are shown in Table 1

Table 1: MESON problems

| Corpus | Problems | Average size | Minimum size | Maximum size |
|---|---|---|---|---|
| Core HOL Light | 2057 | 8.1 | 2 | 64 |
| HOL Multivariate | 12428 | 17.7 | 2 | 226 |
| Flyspeck | 19634 | 12.7 | 2 | 132 |
| Total | 34119 | 14.3 | 2 | 226 |

These problem sizes (which should additionally be considered as the CNF sizes) are lower than in the problems corresponding to the "by" (atomic justifications) in Mizar. There the dependent types with Horn-like adjective mechanisms are a very significant part of the automation, and particularly in more advanced theories their TPTP encoding can produce tens of formulas.

The HOL Light type system also does not have the additional features like type classes that complicate the problems for Isabelle, and it seems that explicit encoding of the type system in the MESON export is entirely avoided. One guard against type-related unsoundness in MESON seems to be exhaustive instantiation of different polymorphic variants into different (untyped first-order) symbols, including equality (this is probably not difficult in a tableau-based system like MESON). Our export to TPTP currently merges all polymorphic instances of equality into the one standard FOL equality, which can make some TPTP problems unsound.[4]

If this all is correct, then it is a bit surprising that the problem sizes (which on our corpora often correlates with first-order ATP difficulty) of the "full-scale ATP" MESON problems that actually appear in the HOL Light corpora look comparable to the problems originating from

[3]`http://mizar.cs.ualberta.ca/~mptp/hh/tptp.tgz`

[4]We became aware of this issue thanks to the PAAR workshop reviews, and so far we have not tried to measure the influence of such unsoundness in the problems. Some related quantification is available in the work of Hurd and Meng and Paulson.

the Mizar's "limited-by-design" and "obvious-inference-only" [5, 23] atomic justifications. This also hints that HOL Light users might be able to do bigger steps if using external ATPs.

## 2.2 Exporting theorem problems

The second interesting set of problems is on the "theorem" level of ITP libraries. This level seems to be quite similar in the major ITPs: "theorem" is typically not corresponding to what mathematicians call a theorem, but it is rather a self-sufficient lemma with a formal proof of tens to hundreds lines that can be useful in other formal proofs and hence should be named and exported. Since the ITP proofs can be longer, proving such theorems fully automatically is typically a challenge, which makes such problems suitable for ATP benchmarks, challenges, and competitions.

In Mizar and in Isabelle (done by Blanchette in so far unpublished work) the corresponding ATP problems for theorems can be produced by collecting the dependencies (premises) from the proofs (by suitable tracking mechanisms), and then translating the $Premises \vdash Theorem$ statement to first-order logic. It seems that HOL Light does not provide (at least not out-of-the box) such high-level tracking of dependencies, however there is recent work by Adams in exporting HOL Light to HOL Zero [1] (with cross-verification as the main motivation) that does (also) high-level dependency tracking. We have used these data (dependency table) as follows:

1. First we attempted to synchronize the theorem names used by Adams with our work (there can be different naming conventions). This was an iterative process that we ended when there were 55 remaining differences out of 1782 theorems (possibly caused also by small version difference).

2. Then for each HOL Light theorem, we have replaced the default "prove" function with a function that first looks up the dependencies (in the external dependency table), filters out those that (for whatever reason) do not exist in the current environment, and calls the MESON exporting code described above for the problem $Dependencies \vdash Theorem$

This is problematic not just because of the possible dependency incompatibilities. The MESON export of some problems can take very long time (one problem that we left overnight took more than five hours), and create very large files (several megabytes). Thousands of formulas are no longer a problem for existing large-theory ATP techniques, but the processing time inside HOL Light makes experiments impractical. This was the main reason why the re-proving experiments based on the dependency information about whole proofs were limited to 1178 HOL Light theorems for which we get the TPTP translation before we encounter such slowdowns. This inefficiency seems to be caused by MESON's exhaustive treatment of polymorphism, which is not a problem for normal solving of small HOL Light tasks with MESON, but does not scale well to large numbers of premises. We either need to a more efficient implementation of this code in HOL Light,[5] or as already mentioned, we might just try to entirely switch from the MESON export to the TFF1 export which will likely avoid major optimizations (those can be done while translating from TFF1 to FOF).

The 1178 theorems give rise to 1993 problems after the MESON export, also available online.[6] The average number of formulas in them is 46, the maximum is 2469, which means that some of these problem should benefit from premise-selection methods.

[5]After reading the first version of this paper, John Harrison started to look at this issue.

[6]`http://mizar.cs.ualberta.ca/~mptp/hh/theorems.tar.gz`

## 2.3 Exporting theorem problems with premise selection

Given the large libraries that have been built with HOL Light, the interesting ATP/AI task is to prove new theorems without having to manually select the relevant premises. ATP problems of this kind are created for Mizar/MML by consistent translation of the whole MML to TPTP, and then letting premise selection algorithms find the most relevant premises for a given theorem $t$ from the large set of $t$-allowed premises (typically those theorems and definitions that were already available when $t$ was being proved, expressed, e.g., as TPTP include files).

Currently, the MESON export that we use invents specific symbols for each problem, and it is not clear to us if it can be easily modified to translate each HOL Light theorem separately to FOF, so that such separate theorem translations could be later consistently combined into large ATP problems. Again, the TFF1 layer should make this possible.

So in order to do the initial test of how good ATP/AI methods can be when using the whole available HOL Light theory at each point, we put the premise selection directly inside HOL Light. This is done as follows:

1. First we train (using the SNoW system in naive Bayes mode) a premise selector on the proof dependency data by Adams. Similar to such training on MML data, HOL Light symbols are used for the input-feature representation of the proved theorems, and the necessary proof dependencies are used as the output features (labels) for the learning. The resulting standalone premise selector is now also accessible online.[7] The 10-fold cross-validation (i.e.: training on 9/10 and testing on 1/10 of the data) gives so far on average about 43% cover of the needed dependencies in the first 100 hits. For Mizar/MML this is about 70% [25]. Some of this difference can be caused by bugs in the data processing, but it is also possible that just using symbols for characterizing formulas is weaker on the HOL Light corpora, and using other features (e.g., all formula (sub)terms) will be useful.

2. Then for each HOL Light theorem, we have replaced the default "prove" function with a function that extracts the symbols from the theorem[8] and sends them as a query to the premise selector. The premise selector replies with a list of theorem names ordered by their expected relevance for the goal, from which we again filter out those that do not exist in the current environment. Then we take the $N$ most relevant of the remaining recommended premises, and call the MESON exporting code for the problem $\mathit{RecommendedPremises} \vdash \mathit{Theorem}$.

This solution is a bit similar to how Isabelle/Sledgehammer selects premises, which is also done internally on Isabelle terms rather than externally on their TPTP representation.[9] However, Isabelle/Sledgehammer now uses a manually taylored relevance filter [19].

The MESON translation with a higher number of premises is however again a bottleneck, so we limit the number of advised premises to 60, and do the export to TPTP only for $N$ equal to 10, 20, 30, 40, 50, and 60. For the same efficiency reasons (and to have the same set of theorems for all $N$) we also stop the export after producing problems for 964 theorems, resulting in about 1700 TPTP problems. The problem numbers can slightly differ for different $N$, for example for `NUMPAIR_INJ_LEMMA` only one problem is created by the MESON export when using 20 premises,

[7] `http://mws.cs.ru.nl/~urban/holdata/isab_adv_demo.html`

[8] We could have used Adams' data for this too, but this way we can use the premise selector also on conjectures that are not in Adams' data.

[9] The Isabelle TPTP export now uses consistent symbol naming, so external premise selectors can be already tested on Isabelle data. Some initial (so far unpublished) experiments with the MaLARea system have been started by Blanchette and Urban.

but two problems are created when using 60 premises. Table 2 summarizes the six datasets, which are also available online.[10]

Table 2: Theorem problems (after MESON export) with advised premises

| Premises | Theorems | Problems | Avg. size/theorem | Avg. size/problem | Max size |
|---|---|---|---|---|---|
| 10 | 964 | 1649 | 86.32 | 50.46 | 882 |
| 20 | 964 | 1662 | 136.84 | 79.37 | 922 |
| 30 | 964 | 1680 | 196.44 | 112.72 | 1097 |
| 40 | 964 | 1683 | 250.80 | 143.65 | 1470 |
| 50 | 964 | 1687 | 339.06 | 193.75 | 1781 |
| 60 | 964 | 1687 | 462.47 | 264.27 | 2181 |

For the higher values of $N$ the average problem sizes reach values that can further benefit from internal ATP large-theory methods developed in the past years. We should also note that in some cases there is currently a total mismatch between the symbols on which the advisor was trained, and the symbols that we extract from the theorem that is to be advised (this can be due to various omissions in the processing and synchronizing of the symbol names with Adams' data). So again, these problems should be considered as an initial experiment rather than the best of what the current premise selection techniques can achieve.

## 3 Experiments

We use Vampire 1.8 and E 1.4 on the problems. All ATPs are run with 5s time limit on an Intel Xeon X5650 2.67GHz server with 24GB RAM and 12MB CPU cache. Each problem is always assigned one CPU.

### 3.1 Using external ATPs to prove the calls to MESON

Table 3 shows the results of running Vampire and E on the MESON problems. The solutions are also online.[11] The problems turn out to be very easy, and the average number of needed Vampire premises is quite low in comparison to the average problem size. One problem (tptp19150.p) has been found countersatisfiable by E. After manually adding an extensionality axiom for functions, both E and Vampire can prove it. It is possible that some knowledge about extensionality of basic constants of HOL is hard-coded in MESON. So far we have not decided to add this axiom to the translation of every problem to FOL, because it is explicitly present in some of the problems. The low number of premises that are actually needed for the proof is also a bit suspicious, but some problems seem to be really trivial (for example, asking to prove that $c \neq c$), which might be partially due to the MESON preprocessing and splitting into multiple problems. There are some harder problems, for example multivariate/tptp13687.p took E to generate 152441 clauses and process 15889 of them.

[10] `http://mizar.cs.ualberta.ca/~mptp/hh/advised_theorems10.tar.gz`, and so on for 20 to 60.

[11] `http://mizar.cs.ualberta.ca/~mptp/hh/meson_results.tar.gz`

Table 3: ATP results on problems created from the MESON calls

| Corpus | Problems | Avg. size | V-proved (%) | E-proved (%) | Avg. V-premises |
|---|---|---|---|---|---|
| Core HOL Light | 2057 | 8.1 | 2055 (99.9%) | 2057 (100%) | 2.91 |
| HOL Multivariate | 12428 | 17.7 | 12422 (100%) | 12393 (99.7%) | 3.40 |
| Flyspeck | 19634 | 12.7 | 19592 (99.8%) | 19621 (99.9%) | 2.15 |
| Total | 34119 | 14.3 | 34069 (99.9%) | 34071 (99.9%) | 2.60 |

## 3.2 Using external ATPs to prove theorems

Table 4 shows the results of running Vampire and E on the 1993 theorem problems. The solutions are again online.[12] Many problems are reported countersatisfiable by E, which can mean that we are missing some proof dependencies, processing them in a wrong way, or that the completeness of the MESON export is limited (note that for the MESON problems in the previous section we are only using those which succeeded in HOL Light). The average number of premises that Vampire needed for the 519 proofs went a bit higher than in the previous section, but it is still suspiciously low. The overall success rate is 26%, however as mentioned above, these are theorems from the beginning of the core HOL Light corpus, and the export (and thus also likely the problems) get harder later.

Table 4: ATP results on the 1993 theorem problems

| Problems | Avg. size | V-proved (%) | E-proved (%) | Avg. V-premises | E-CounterSat (%) |
|---|---|---|---|---|---|
| 1993 | 46 | 519 (26%) | 517 (26%) | 4.6 | 876 (44%) |

## 3.3 Using external ATPs to prove theorems with premise selection

Table 5 shows the results of running Vampire and E on the six differently advised batches of theorem problems. The solutions are again online.[13] Advising more premises helps quite a lot, and particularly Vampire is good in handling the larger problems. The advised theorem problems are a subset of those from previous section, so the result of 455 proved by Vampire in the 60-advised batch compares quite well to the 519 proved in the previous section. This seems encouraging, but again, modulo all the possible bugs and imperfections that might be involved.

Table 5: ATP results on the advised theorem problems

| Premises | Problems | Avg. size | V-proved (%) | E-proved (%) | Avg. V-premises | E-ContrSat (%) |
|---|---|---|---|---|---|---|
| 10 | 1649 | 50.46 | 225 (13.6%) | 221 (13.4%) | 3.33 | 352 (21.3%) |
| 20 | 1662 | 79.37 | 294 (17.7%) | 288 (17.3%) | 4.22 | 175 (10.5%) |
| 30 | 1680 | 112.72 | 350 (20.1%) | 340 (20.2%) | 4.55 | 95 (5.7%) |
| 40 | 1683 | 143.65 | 387 (23%) | 354 (21%) | 4.86 | 41 (2.4%) |
| 50 | 1687 | 193.75 | 427 (25.3%) | 362 (21.5%) | 5.03 | 29 (1.7%) |
| 60 | 1687 | 264.27 | 455 (27%) | 367 (21.7%) | 5.13 | 29 (1.7%) |

[12] `http://mizar.cs.ualberta.ca/~mptp/hh/theorems_results.tar.gz`

[13] `http://mizar.cs.ualberta.ca/~mptp/hh/advised_theorems_results.tar.gz`

## 4 Conclusion and Future Work

What we did seems straightforward. Inside HOL Light we encoded the translation to TPTP using MESON, introduced some bookkeeping to keep track of available theorems, implemented the calls to the premise advisor, and hooked these functions to suitable places. Outside HOL Light, most of the work was in researching how to use and synchronize Adams' dependency data. Training the basic naive Bayes premise selection and providing the trained advisor is now a standard technology done already many times.

We might have made mistakes in tweaking the HOL Light data and functions for our purpose, and one reason for this workshop paper is to expose any serious bugs to better-informed eyes. However even if there were serious issues in exporting the problems in the TPTP format, it still seems that doing what we are attempting to do is quite well-researched today, and the large-theory ATP/AI is out there, ready to be applied to the HOL Light corpora. We have not done any major sanity checking yet w.r.t. the proofs that we obtain. One issue that we became aware of (after the reviews of the first version of the paper) is our use of one global equality predicate, which together with MESON's removal of type guards can lead to unsound translations. We have not measured the influence of such unsoundness yet. However, HOL Light has methods that import MESON and Ivy proofs, and a lot of relevant work has been done on proof import with Isabelle/Sledgehammer using Metis.

Future work has been mentioned several times. Probably the lowest hanging fruit is to export all theorems from the corpora in the TFF1 format. This could solve the efficiency and symbol-consistency problems, and allow us to use premise selection externally rather than internally.

## 5 Acknowledgments

Mark Adams gave us his HOL Light proof export data for HOL Zero, which made it possible to attempt the reproving and learning experiments for larger proofs. Tom Hales helped to start the MESON exporting work at CICM 2011, and his interest as a leader of Flyspeck motivates us. Piotr Rudnicki has made his Mizar workstation available for the experiments. Thanks to John Harrison for providing MESON (and HOL Light) and discussing some topics. Obviously, none of these people have any responsibility for possible bugs and errors that we might have committed when exploring their work and attempting to use it. Thanks also to the PAAR 2012 referees, who provided a number of helpful comments and suggestions.

## References

[1] Mark Adams. Introducing HOL Zero - (extended abstract). In Komei Fukuda, Joris van der Hoeven, Michael Joswig, and Nobuki Takayama, editors, *ICMS*, volume 6327 of *Lecture Notes in Computer Science*, pages 142–143. Springer, 2010.

[2] Jasmin Christian Blanchette, Sascha Böhme, and Lawrence C. Paulson. Extending Sledgehammer with SMT solvers. In Nikolaj Bjørner and Viorica Sofronie-Stokkermans, editors, *CADE*, volume 6803 of *Lecture Notes in Computer Science*, pages 116–130. Springer, 2011.

[3] Jasmin Christian Blanchette, Lukas Bulwahn, and Tobias Nipkow. Automatic proof and disproof in Isabelle/HOL. In Cesare Tinelli and Viorica Sofronie-Stokkermans, editors, *FroCoS*, volume 6989 of *Lecture Notes in Computer Science*, pages 12–27. Springer, 2011.

[4] Jasmin Christian Blanchette and Andrei Paskevich. TFF1: The TPTP typed first-order form with rank-1 polymorphism. Available online at `http://www4.in.tum.de/~blanchet/tff1spec.pdf`.

[5] Martin Davis. Obvious logical inferences. In Patrick J. Hayes, editor, *IJCAI*, pages 530–531. William Kaufmann, 1981.

[6] Leonardo Mendonça de Moura and Nikolaj Bjørner. Z3: An Efficient SMT Solver. In C. R. Ramakrishnan and Jakob Rehof, editors, *TACAS*, volume 4963 of *Lecture Notes in Computer Science*, pages 337–340. Springer, 2008.

[7] Jean-Christophe Filliâtre and Claude Marché. The Why/Krakatoa/Caduceus platform for deductive program verification. In Werner Damm and Holger Hermanns, editors, *CAV*, volume 4590 of *Lecture Notes in Computer Science*, pages 173–177. Springer, 2007.

[8] Thomas C. Hales. Introduction to the Flyspeck project. In Thierry Coquand, Henri Lombardi, and Marie-Françoise Roy, editors, *Mathematics, Algorithms, Proofs*, volume 05021 of *Dagstuhl Seminar Proceedings*. Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany, 2005.

[9] Thomas C. Hales. Mathematics in the age of the Turing machine. *Lecture Notes in Logic*, 2012. to appear; `http://www.math.pitt.edu/~thales/papers/turing.pdf`.

[10] John Harrison. HOL Light: A tutorial introduction. In Mandayam K. Srivas and Albert John Camilleri, editors, *FMCAD*, volume 1166 of *Lecture Notes in Computer Science*, pages 265–269. Springer, 1996.

[11] John Harrison. Optimizing Proof Search in Model Elimination. In M. McRobbie and J.K. Slaney, editors, *Proceedings of the 13th International Conference on Automated Deduction*, number 1104 in Lecture Notes in Artificial Intelligence, pages 313–327. Springer-Verlag, 1996.

[12] Joe Hurd. Integrating Gandalf and HOL. In Yves Bertot, Gilles Dowek, André Hirschowitz, C. Paulin, and Laurent Théry, editors, *TPHOLs*, volume 1690 of *Lecture Notes in Computer Science*, pages 311–322. Springer, 1999.

[13] Joe Hurd. An LCF-style interface between HOL and first-order logic. In Andrei Voronkov, editor, *CADE*, volume 2392 of *Lecture Notes in Computer Science*, pages 134–138. Springer, 2002.

[14] Joe Hurd. First-order proof tactics in higher-order logic theorem provers. In Myla Archer, Ben Di Vito, and César Muñoz, editors, *Design and Application of Strategies/Tactics in Higher Order Logics (STRATA 2003)*, number NASA/CP-2003-212448 in NASA Technical Reports, pages 56–68, September 2003.

[15] Donald W. Loveland. Mechanical theorem proving by model elimination. *Journal of the ACM*, 15(2):236–251, April 1968.

[16] Donald W. Loveland. *Automated Theorem Proving: A Logical Basis.* North-Holland, Amsterdam, 1978.

[17] William McCune. Prover9 and Mace4. `http://www.cs.unm.edu/~mccune/prover9/`, 2005–2010.

[18] William McCune and Olga Shumsky Matlin. Ivy: A Preprocessor and Proof Checker for First-Order Logic. In M. Kaufmann, P. Manolios, and J. Strother Moore, editors, *Computer-Aided Reasoning: ACL2 Case Studies*, number 4 in Advances in Formal Methods, pages 265–282. Kluwer Academic Publishers, 2000.

[19] Jia Meng and Lawrence C. Paulson. Lightweight relevance filtering for machine-generated resolution problems. *J. Applied Logic*, 7(1):41–57, 2009.

[20] Lawrence C. Paulson and Kong Woei Susanto. Source-level proof reconstruction for interactive theorem proving. In Klaus Schneider and Jens Brandt, editors, *TPHOLs*, volume 4732 of *Lecture Notes in Computer Science*, pages 232–245. Springer, 2007.

[21] Henri Poincaré. *The foundations of science: Science and hypothesis, The value of science, Science and method.* The Science Press, New York, 1913.

[22] Alexandre Riazanov and Andrei Voronkov. The design and implementation of VAMPIRE. *AI Commun.*, 15(2-3):91–110, 2002.

[23] Piotr Rudnicki. Obvious Inferences. *Journal of Automated Reasoning*, 3(4):383–393, 1987.

[24] Stephan Schulz. E - A Brainiac Theorem Prover. *AI Commun.*, 15(2-3):111–126, 2002.

[25] Josef Urban. MPTP - Motivation, Implementation, First Experiments. *Journal of Automated Reasoning*, 33(3-4):319–339, 2004.

[26] Josef Urban. MPTP 0.2: Design, implementation, and initial experiments. *J. Autom. Reasoning*, 37(1-2):21–43, 2006.

[27] Josef Urban, Piotr Rudnicki, and Geoff Sutcliffe. ATP and presentation service for Mizar formalizations. *CoRR*, abs/1109.0616, 2011.

[28] Josef Urban and Geoff Sutcliffe. Automated reasoning and presentation support for formalizing mathematics in Mizar. In Serge Autexier, Jacques Calmet, David Delahaye, Patrick D. F. Ion, Laurence Rideau, Renaud Rioboo, and Alan P. Sexton, editors, *AISC/MKM/Calculemus*, volume 6167 of *Lecture Notes in Computer Science*, pages 132–146. Springer, 2010.

[29] Josef Urban, Geoff Sutcliffe, Petr Pudlák, and Jirí Vyskocil. MaLARea SG1- Machine Learner for Automated Reasoning with Semantic Guidance. In Alessandro Armando, Peter Baumgartner, and Gilles Dowek, editors, *IJCAR*, volume 5195 of *Lecture Notes in Computer Science*, pages 441–456. Springer, 2008.

[30] Christoph Weidenbach, Dilyana Dimova, Arnaud Fietzke, Rohit Kumar, Martin Suda, and Patrick Wischnewski. SPASS Version 3.5. In Renate A. Schmidt, editor, *CADE*, volume 5663 of *Lecture Notes in Computer Science*, pages 140–145. Springer, 2009.

# Learning from Multiple Proofs: First Experiments

Daniel Kühlwein and Josef Urban*
Radboud University Nijmegen
Nijmegen, Netherlands
`firstname.lastname@gmail.com`

**Abstract**

Mathematical textbooks typically present only one proof for most of the theorems. However, there are infinitely many proofs for each theorem in first-order logic, and mathematicians are often aware of (and even invent new) important alternative proofs and use such knowledge for (lateral) thinking about new problems.

In this paper we start exploring how the explicit knowledge of multiple (human and ATP) proofs of the same theorem can be used in learning-based premise selection algorithms in large-theory mathematics. Several methods and their combinations are defined, and their effect on the ATP performance is evaluated on the MPTP2078 large-theory benchmark. Our first findings are that the proofs used for learning significantly influence the number of problems solved, and that the quality of the proofs is more important than the quantity.

## 1 Introduction

Automated Theorem Provers (ATPs) struggle when a problem has a large number of unnecessary premises [9]. Premise selection in large theories seems to be an important instance of the general *proof guidance* problem. It has been shown that proper design and choice of knowledge selection heuristics can change the overall success of large-theory ATP techniques by tens of percents [1]. This paper continues our work on machine learning algorithms for premise selection [1, 5]. We investigate how the knowledge of different proofs can be integrated in the machine learning algorithms for premise selection, and how it influences the performance of the ATPs.

In our earlier experiments [5] we tested and evaluated several premise selection algorithms on a subset of the Mizar Mathematical Library (MML), the MPTP2078 large-theory benchmark,[1] using the (human) proofs from the MML as training data for the learning algorithms. We have found that (i) learning from such human proofs helps a lot, but (ii) alternative proofs can quite often be successfully constructed by ATPs, making heuristic methods like SInE surprisingly strong and orthogonal to learning methods. Thanks to these experiments we now also have (possibly several) ATP proofs for most of the problems. This gives us an opportunity to learn from different (and in particular not just human) proofs and their combinations, and observe the influence on the ATP performance.

The rest of the paper is organized as follows. Section 2 introduces the necessary machine learning terminology and explains how different proofs can be used in the algorithms. In Section 3, we define several possible ways to use the additional knowledge given by the different proofs. The different proof combinations are evaluated and discussed in Section 4, and Section 5 concludes.

*The authors were supported by the NWO projects "MathWiki a Web-based Collaborative Authoring Environment for Formal Proofs" and "Learning2Reason".

[1]Available at `http://wiki.mizar.org/twiki/bin/view/Mizar/MpTP2078`.

## 2 The Machine Learning Framework and the Data

We start with the setting introduced in [1, 5]. $\Gamma$ denotes the set of all first order formulas (usually axioms, definitions and theorems) that appear in a given (fixed) large mathematical corpus (MPTP2078 in this paper). The corpus is assumed to use notation (symbols) and formula names consistently, since they are used to define the features and labels for the machine learning algorithms. Given a conjecture $c$ and a large number of *allowed premises* $\mathcal{A}_c \subset \Gamma$, the *premise selection problem* is to predict those premises from $\mathcal{A}_c$ that are likely to be useful for automatically constructing a proof of $c$. We typically experiment with proving theorems from $\Gamma$, so not all formulas in $\Gamma$ can be allowed as premises for a given $c \in \Gamma$. In practice, $\Gamma$ is typically ordered by the chronological growth of ITP libraries, and we use this ordering to define which premises are allowed for a given conjecture.

We say that a *proof $P$ is a proof over* $\Gamma$ if the conjecture and all premises used in $P$ are elements of $\Gamma$. Given a set of proofs $\Delta$ over $\Gamma$ in which every formula has at most one proof, the ($\Delta$-based) *proof matrix* $\mu_\Delta : \Gamma \times \Gamma \to \{0,1\}$ is defined as

$$\mu_\Delta(c,p) := \begin{cases} 1 & \text{if } p \text{ is used to prove } c \text{ in } \Delta, \\ 0 & \text{otherwise.} \end{cases}$$

In other words, $\mu_\Delta$ is the adjacency matrix of the graph of the direct proof dependencies from $\Delta$. This proof matrix, together with the formula features, is used for training machine learning algorithms in [1, 5], using MML proofs as $\Delta$. [1] and [5] also contain more details on using machine learning for premise selection.

In [5] we compared several different premise selection algorithms on the MPTP2078 dataset. Thanks to this comparison we now have ATP proofs for 1328 of the 2078 problems, found by Vampire 0.6 [7]. For some problems we found several different proofs, meaning that the sets of premises used in the proofs differ. Figure 1 shows the number of different ATP proofs we have for each problem. The maximum number is 49. We found 6.71 proofs per solvable problem on average.

This database of proofs allows us to start considering in this paper multiple proofs for a $c \in \Gamma$. For each conjecture $c$, let $\Theta_c$ be the *set of all ATP proofs of* $c$ in our dataset, and let $n_c$ denote the *cardinality of* $\Theta_c$. Due to the nature of our learning algorithms, we still use the (generalized) proof matrix setting, where the multiple proofs of $c$ are all in one row of $\mu$ represented using *weights*. Therefore $\mu$ will rather be called the (premise) *weight matrix* in the rest of the paper, and the general interpretation of $\mu_X(c,p)$ is the relevance (*weight*) of a premise $p$ for a proof of $c$ determined by $X$, where $X$ can either be a set of proofs defining $\mu$ as above, or a particular algorithm (typically in conjunction with the data to which it is applied) defining the values of $\mu$. For a single proof $\sigma$, let $\mu_\sigma := \mu_{\{\sigma\}}$, i.e.,

$$\mu_\sigma(c,p) := \begin{cases} 1 & \text{if } \sigma \in \Theta_c \text{ and } p \text{ is used to prove } c \text{ in } \sigma, \\ 0 & \text{otherwise.} \end{cases}$$

We end this section by introducing the (so far a bit fuzzy) concept of (premise) *redundancy*, which seems to be at the heart of the problem that we are exploring. Let $c$ be a conjecture and $\sigma_1, \sigma_2$ be proofs for $c$ ($\sigma_1, \sigma_2 \in \Theta_c$) with used premises $\{p_1, p_2\}$ and $\{p_1, p_2, p_3\}$ respectively. In this case, premise $p_3$ can be called *redundant* since we know a proof of $c$ that does not use $p_3$.[2]

[2]For this we assume some similarity between the efficiency of the proofs in $\Theta_c$, which is the case for our experiments based on the 5-second time limit.

Redundant premises appear quite frequently in ATP proofs, for example, due to exhaustive equational normalization that can turn out to be unnecessary for the proof. Now imagine we have a third proof of $c$, $\sigma_3$ with used premises $\{p_1, p_3\}$. With this knowledge, $p_2$ could also be called redundant (or at least unnecessary). But one could also argue that at least one of $p_2$ and $p_3$ is not redundant. In such cases, it is not yet clear what a meaningful definition of redundancy should be (see also [6] for related topics in machine learning). Hence for the remainder of the paper, we use the term *redundancy* as a (partially) vague concept to talk about premises that might not be necessary for a proof.

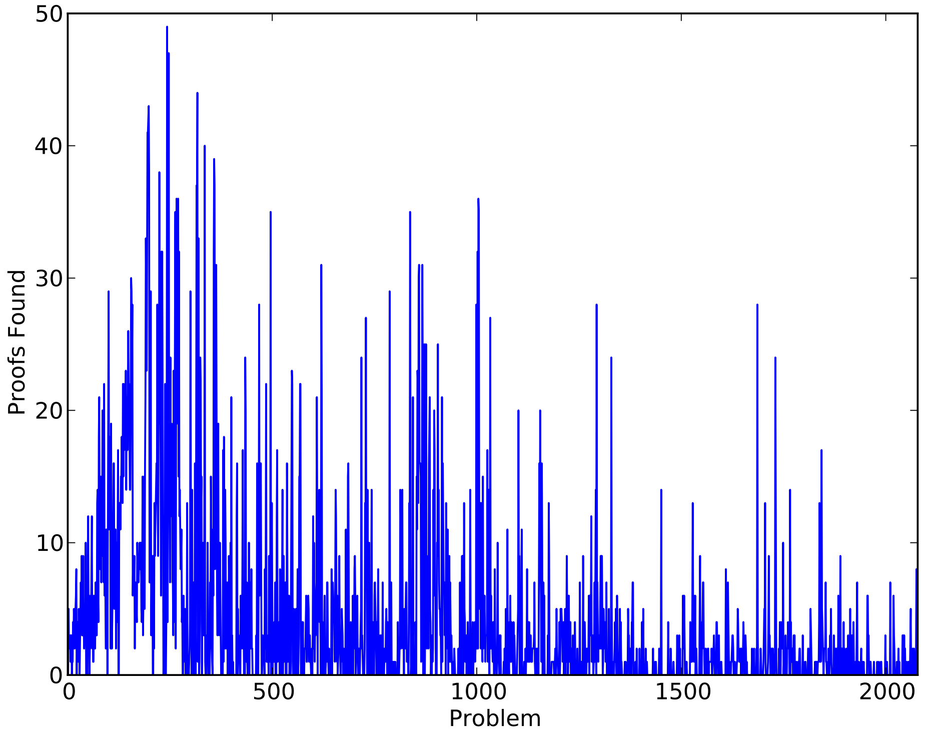


Figure 1: Number of different ATP proofs for each of the 2078 problems. The problems are ordered by their appearance in the MML.

## 3 Using Multiple Proofs

We define several different combinations of MML and ATP proofs and their respective premise weight matrices. Keep in mind that there are many problems for which we do not have any ATP proofs. For those problems, we will always just use the MML proof. I.e., for all premise weight matrices $\mu_X$ defined below, if there is no ATP proof of a conjecture $c$, then $\mu_X(c, p) = \mu_{\mathrm{MML}}(c, p)$.

## 3.1 Substitutions and Unions

The simplest way to combine different proofs is to either only consider the used premises of one proof, or take the union of all used premises. We consider five different combinations.

**Definition 3.1.1** (MML Proofs).

$$\mu_{\mathrm{MML}}(c,p) := \begin{cases} 1 & \textit{if } p \textit{ is used to prove } c \textit{ in the MML proof,} \\ 0 & \textit{otherwise.} \end{cases}$$

This dataset will be used as baseline throughout all experiments. It uses the human proofs from the Mizar library.

**Definition 3.1.2** (Random ATP Proof). *For each conjecture $c$ for which we have ATP proofs, pick a (pseudo)random ATP proof $\sigma_c \in \Theta_c$. Then we define*

$$\mu_{\mathrm{Random}}(c,p) := \begin{cases} 1 & \textit{if } p \textit{ is a used premises in } \sigma_c, \\ 0 & \textit{otherwise.} \end{cases}$$

**Definition 3.1.3** (Best ATP Proof). *For each conjecture $c$ for which we have ATP proofs, pick an(y) ATP proof with the least number of used premises $\sigma_c^{min} \in \Theta_c$. We set*

$$\mu_{\mathrm{Best}}(c,p) := \begin{cases} 1 & \textit{if } p \textit{ is a used premises in } \sigma_c^{min}, \\ 0 & \textit{otherwise.} \end{cases}$$

**Definition 3.1.4** (Random Union). *For each conjecture $c$ for which we have ATP proofs, pick a random ATP proof $\sigma_c \in \Theta_c$. $\mu_{mathrmrandomUnion}$ is defined as*

$$\mu_{\mathrm{RandomUnion}}(c,p) := \begin{cases} 1 & \textit{if } p \textit{ is a premise used in } \sigma_c \textit{ or in the MML proof of } c, \\ 0 & \textit{otherwise.} \end{cases}$$

**Definition 3.1.5** (Union). *For each conjecture $c$ for which we have ATP proofs, we define*

$$\mu_{\mathrm{Union}}(c,p) := \begin{cases} 1 & \textit{if } p \textit{ is a premise used in any ATP or MML proof of } c, \\ 0 & \textit{otherwise.} \end{cases}$$

## 3.2 Premise Averaging

A more advanced way to combine proofs is to consider some kind of *average* of the used premises. We consider three options, the standard average, a biased average and a scaled average.

**Definition 3.2.1** (Average). *The average gives equal weight to each proof.*

$$\mu_{\mathrm{Average}}(c,p) = \frac{1}{n_c+1} \sum_{\sigma \in \Theta_c} \mu_\sigma(c,p) + \mu_{\mathrm{MML}}(c,p)$$

The intuition is that the average gives a better idea of how necessary a premise really is. When there are very different proofs, such average will give a very low weight to every premise. That is why we also tried scaling as follows:

**Definition 3.2.2** (Scaled Average). *The scaled average ensures that there is at least one premise with weight 1.*

$$\mu_{\text{ScaledAverage}}(c,p) = \frac{\sum_{\sigma \in \Theta_c} \mu_\sigma(c,p) + \mu_{\text{MML}}(c,p)}{\max_{q\in\Gamma} \sum_{\sigma\in\Theta_c} \mu_\sigma(c,q) + \mu_{\text{MML}}(c,q)}$$

Another experiment is to make the weight of all the ATP proofs equal to the weight of the MML proof:

**Definition 3.2.3** (Biased Average).

$$\mu_{\text{BiasedAverage}}(c,p) = \frac{1}{2}\left(\frac{\sum_{\sigma\in\Theta_c} \mu_\sigma(c,p)}{n_c} + \mu_{\text{MML}}(c,p)\right)$$

## 3.3 Premise Expansion

Consider a situation where $a \vdash b$ and $b \vdash c$. Obviously, not only $b$, but also $a$ proves $c$. When we consider the used premises in a proof, we only use the information about the *direct* premises ($b$ in the example), but nothing about the *indirect* premises ($a$ in the example), the premises of the direct premises. Using this additional information might help the learning algorithms. We call this *premise expansion*. We define three different weight functions that try to capture this indirect information. All three penalize the weight of the indirect premises with a factor of $\frac{1}{2}$.

**Definition 3.3.1** (MML Expansion). *For the MML expansion, we only consider the MML proofs and their one-step expansions:*

$$\mu_{\text{MMLExp}}(c,p) = \mu_{\text{MML}}(c,p) + \frac{\sum_{q\in\Gamma} \mu_{\text{MML}}(c,q)\mu_{\text{MML}}(q,p)}{2}$$

*Note that since $\mu_{\text{MML}}(c,p)$ is either 0 or 1, the sum $\sum_{q\in\Gamma} \mu_{\text{MML}}(c,q)\mu_{\text{MML}}(q,p)$ just counts how often $p$ is a grandparent premise of $c$.*

**Definition 3.3.2** (Average Expansion). *The average expansion takes $\mu_{\text{Average}}$ instead of $\mu_{\text{MML}}$:*

$$\mu_{\text{AverageExp}}(c,p) = \mu_{\text{Average}}(c,p) + \frac{\sum_{q\in\Gamma} \mu_{\text{Average}}(c,q)\mu_{\text{Average}}(q,p)}{2}$$

**Definition 3.3.3** (Scaled Expansion). *And finally, we consider an expansion of $\mu_{\text{ScaledAverage}}$.*

$$\mu_{\text{ScaledAverageExp}}(c,p) = \mu_{\text{ScaledAverage}}(c,p) + \frac{\sum_{q\in\Gamma} \mu_{\text{ScaledAverage}}(c,q)\mu_{\text{ScaledAverage}}(q,p)}{2}$$

Deeper expansions and different penalization factors are possible, but given the performance of these initial tests shown in the next section we decided to not investigate further.

## 4 Results

### 4.1 Experimental Setup

All experiments were done on the MPTP2078 dataset. Because of its good performance in earlier evaluations, we used the Multi-Output-Ranking (MOR) learning algorithm [1] for the experiments. For each conjecture, MOR is allowed to train on all proofs that were (in the chronological order of MML) done up to that conjecture. In particular, this means that the algorithms do not train on the data they were asked to predict. Three-fold cross validation on the training data was used to find the optimal parameters. For the combinations in 3.1, the AUC measure was used to estimate the performance. The other combinations used the square-loss error. For each of the 2078 problems, MOR predicts a ranking of the premises.

We again use Vampire 0.6 for evaluating the predictions. Vampire is run with 5s time limit on an Intel Xeon E5520 2.27GHz server with 24GB RAM and 8MB CPU cache. Each problem is always assigned one CPU. [3] For each MPTP2078 problem, we created 20 new problems, containing the $10, 20, ..., 200$ highest ranked premises and ran Vampire on each of them. The graphs show how many problems were solved using the $10, 20, ..., 200$ highest ranked premises. As a performance baseline, Vampire 0.6 in CASC mode (that means also using SInE with different parameters on large problems) can solve 548 problems in 10 seconds [1].

## 4.2 Substitutions and Unions

Figure 2 shows the performance of the simple proof combinations introduced in 3.1. It can be seen that using ATP instead of MML proofs can improve the performance considerably, in particular when only few premises are provided. One can also see the difference that the quality of the proof makes. The best ATP proof predictions always solved more problems than the random ATP proof predictions. Taking the union of two or more proofs decreases the performance. This can be due to the redundancy introduced by considering many different premises. This would suggest that the ATP search profits most from a simple and clear (one-directional) advice, rather than from a combination of ideas.

## 4.3 Premise Averaging

Taking the average of the used premises could be a good way to combat the redundant premises. The idea is that premises that are actually important should appear in almost every proof, whereas premises that are redundant should only be present in a few proofs. Hereby, important premises should get a high weight and unimportant premises a low weight. The results of the averaging combinations can be seen in Figure 3.2.

Apart from the scaled average, it seems that taking the average does perform better than taking the union. However, the baseline of only the MML premises is better or almost as good as the average predictions.

## 4.4 Premise Expansions

Finally, we compare how expanding the premises effects the ATP performance in Figure 3.3. While expanding the premises does add additional redundancy, it also adds further potentially useful information.

However, all expansions perform considerably worse than the MML proof baseline. It seems that the additional redundancy outweighs the usefulness.

[3]We use Vampire as our default ATP because of its good performance in the CASC competitions, and because of its good performance on MML reported in [9]. Version 0.6 was chosen to make the experiments comparable with the results reported in [1] and [5].

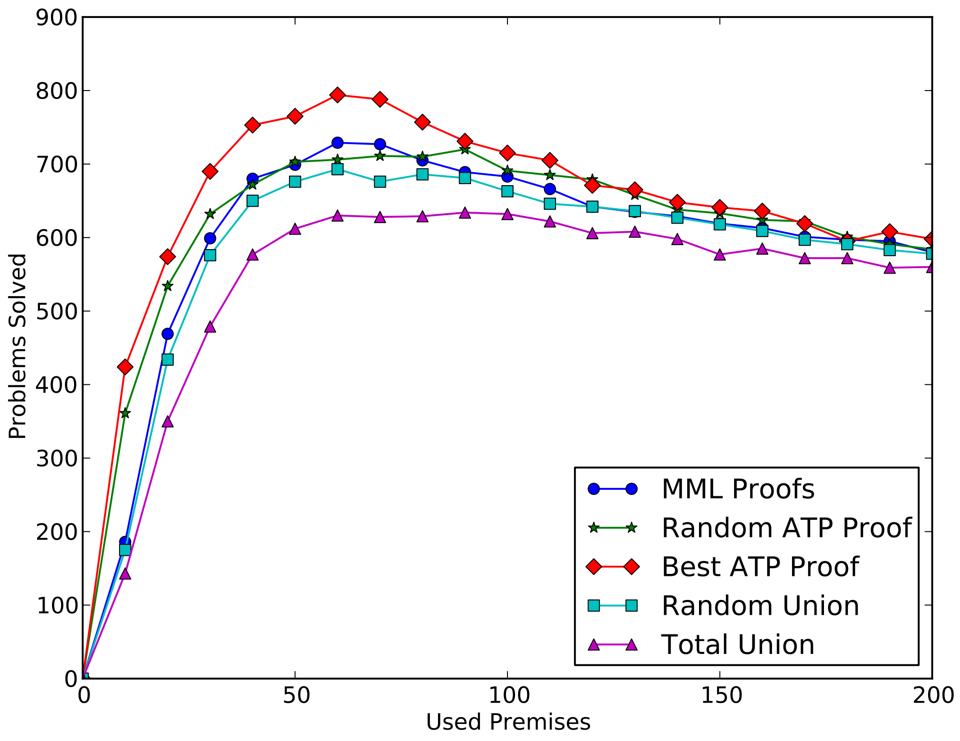


Figure 2: Comparison of the combinations presented in 3.1.

## 4.5 Other ATPs

We also investigated how learning from Vampire proofs affects other provers, by running E 1.4 [8] and Z3 3.2[3] on some of the learned predictions.[4] Figure 5 shows the results. The predictions learned from the MML premises serve as a baseline.

E using the predictions based on the best Vampire proofs is not so much improved over the MML-based predictions as Vampire is. This would suggest that the ATPs really profit most from “their own” best proofs. However for Z3 the situation is opposite: the improvement by learning from the best Vampire proofs is at some points even slightly better than for Vampire itself, and this helps Z3 to reach the maximum performance earlier than before. Also, learning from the averaged proofs behaves differently for the ATPs. For E, the MML and the averaged proofs give practically the same performance, for Vampire the MML proofs are better, but for Z3 the averaged proofs are quite visibly better. These effects are probably not so significant to be immediately investigated, but for example a comparison done on the whole MML (or its different parts), together with analysis of the strategies that the ATPs use, could provide more understanding. Another possible next experiment could be to swap the ATPs, and learn from E’s proofs (or even Z3 proofs, which seem to be coming soon), and see how they improve the other ATPs.

[4]With the same settings as Vampire.

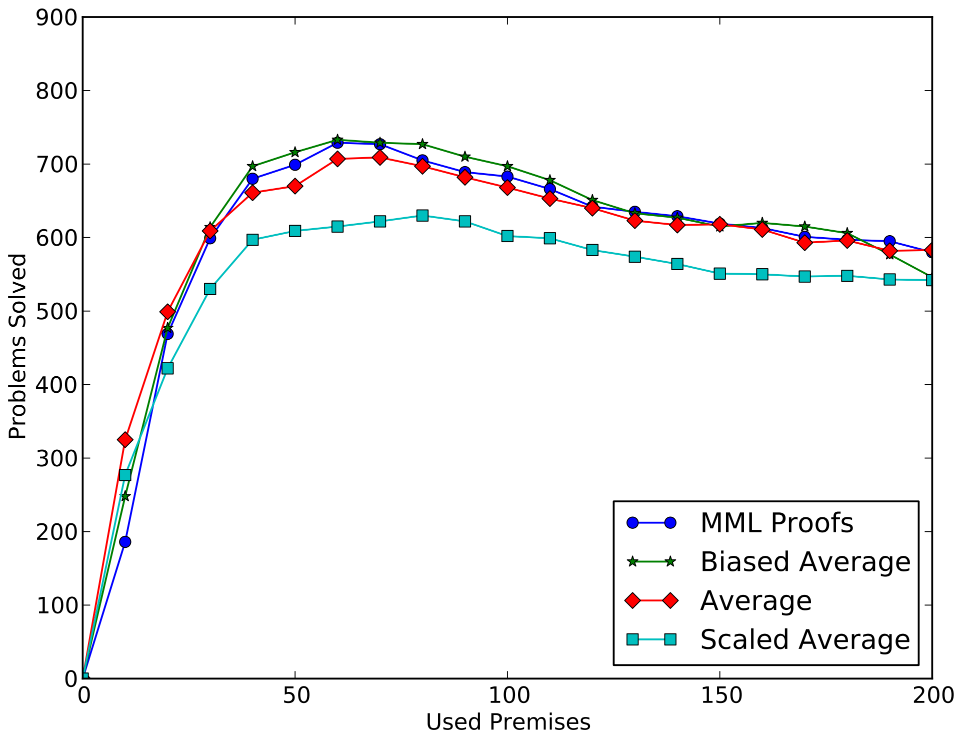


Figure 3: Comparison of the combinations presented in 3.2.

## 4.6 Comparison With the Best Results Obtained so far

In [5], we found that the a combination of SInE [4] and the MOR algorithm (trained on the MML proofs) has so far best performance on the MPTP2078 dataset. Figure 6 compares the new results from this paper with this combination. Furthermore we also try combining SInE with MOR trained on ATP proofs. For comparison we also include our baseline, the MML Proof predictions, and the results obtained from the SInE predictions.

While learning from the best ATP proofs leads to more problems solved than learning from the MML proofs, the combination of SInE and learning from MML proofs still beats both. However, combining the SInE predictions with the best ATP proof predictions gives even better results with a maximum of 823 problem solved (a 3.3% increase over the former maximum) when given the top 70 premises.

## 4.7 Machine Learning Evaluation

Machine learning has several methods to measure how good a learned classifier is without having to run an ATP. In general, this is done by comparing the learned prediction with the training data. The 100%Recall measure is an example. It tells us how many premises (starting from the highest ranked one) we need to give to the ATP to ensure that all necessary premises (according to the training data) are included.

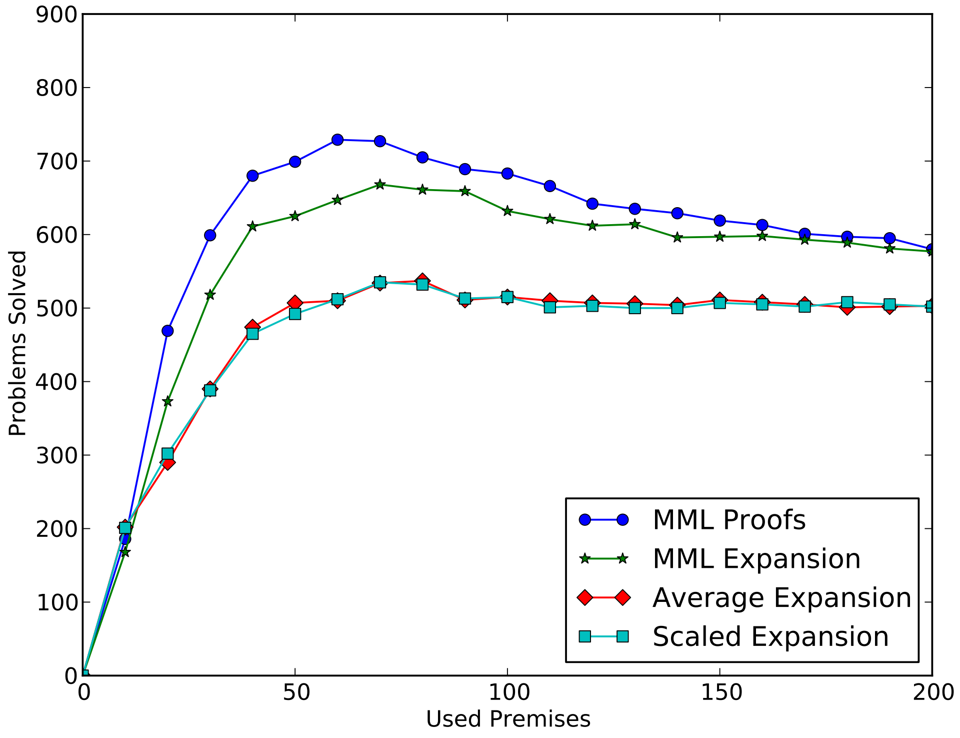


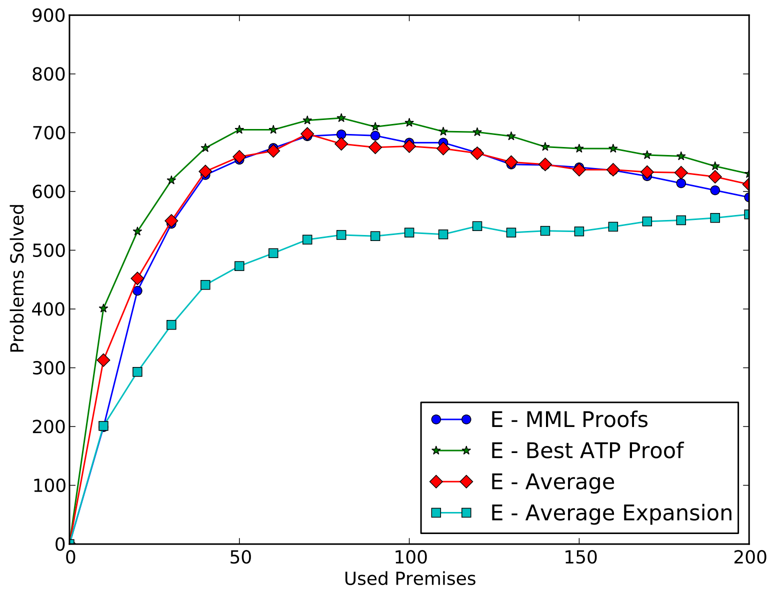
Figure 4: Comparison of the combinations presented in 3.3.

In [5] we found that the machine learning evaluation did not correspond to the ATP evaluation. For example, SInE performed worse than BiLi on the machine learning evaluation but better than BiLi on the ATP evaluation. Our explanation was that we are training from (and therefore measuring) the wrong data. With SInE the ATP found proofs that were very different from the MML proofs.

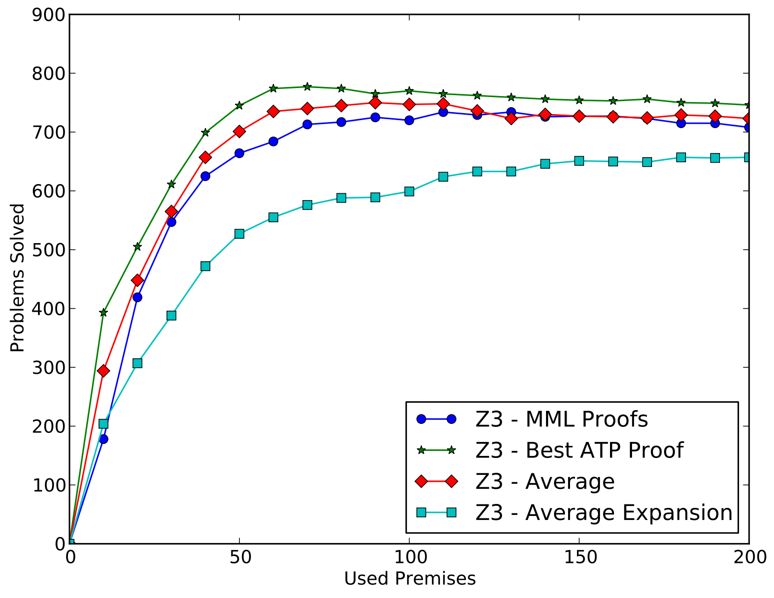
In Figure 7 we see a comparison between a machine learning evaluation (the 100%Recall measure ) depending on whether we evaluate on the MML proofs or on the best ATP proofs. Ideally we would like to have that the machine learning performance of the algorithms corresponds to the ATP performance (see Figure 6). This is clearly not the case for the 100%Recall on the MML proofs graph. The best ATP predictions are better than the MML proof predictions, and SInE solves more than 200 problems. With the new evaluation, the 100%Recall on the best ATP proofs graph, the performance is more similar to the actual ATP performance but there is still room for improvement.

## 5 Conclusion and Future Work

The fact that there is never only one proof makes premise selection an interesting machine learning problem. Since it is in general undecidable to know the "best prediction", the domain



(a) E



(b) Z3

Figure 5: Performance of other ATPs when learning from Vampire proofs.

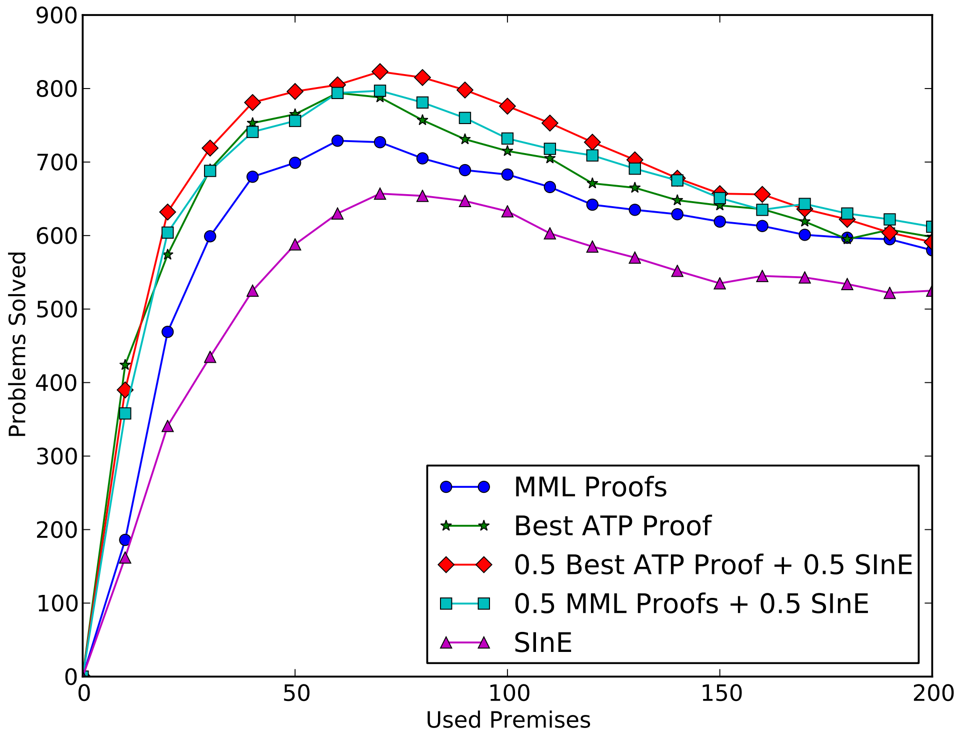


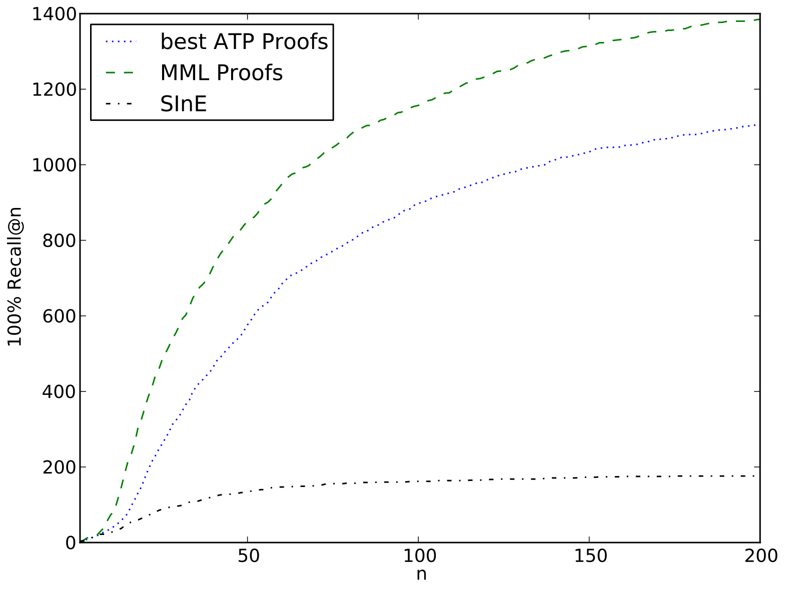
Figure 6: Comparison of the best performing algorithms.

has a randomness aspect that is quite unusual (Chaitin-like [2]) in AI.

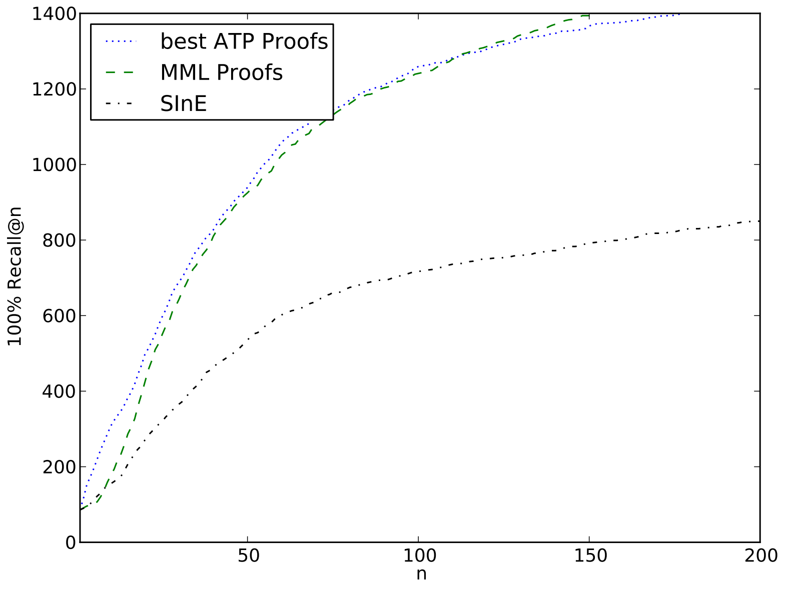
In this paper we started to explore how to combine different proofs to obtain better information for high-level proof guidance by premise selection. We found that it is easy to introduce so much redundancy that the predictions created by the learning algorithms are not good for existing ATPs. On the other hand we saw that learning from proofs with few premises (and hence probably less redundancy) increases the ATP performance. It seems that we should look for a measure of how 'good' or 'simple' a proof is, and only learn from the best proofs (A similar problem appears in machine learning under the name *instance selection* [6]). Such measures could be for example the number of inference steps done by the ATP during the proof search, or the total CPU time needed to find the proof.

Another question that was (at least initially) answered in this paper is to what extent learning from human proofs can help an ATP, in comparison to learning from ATP proofs. We saw that while not optimal, learning from human proofs seems to be approximately equivalent to learning from suboptimal (for example random, or averaged) ATP proofs. Learning from the best ATP proof is about as good as combining SInE with learning from the MML proofs. Combining SInE with learning from the best ATP proof still outperforms all methods, but the improvement is not as strong as in [5].

In the future we want to explore better methods for combining heuristics like SInE with machine learning methods. Such methods could operate (e.g., using SInE-like symbol expan-



(a) 100%Recall on the MML proofs.



(b) 100%Recall on the best ATP proofs.

Figure 7: 100%Recall comparison between evaluating on the MML and the best ATP proofs. The graphs show how many problems have all necessary premises (according to the training data) within the $n$ highest ranked premises.

sion) on the graph of symbol definitions (and similarities) in an analogous way to how the proof graph expansion was used in Section 3, but enriching the set of learning features instead of the set of learning labels. This, together with learning from simpler proofs could lead to even better results. We are also interested in using thresholds (instead of just rankings) in our algorithms.

## 6 Acknowledgments

Tom Heskes, Evgeni Tsivtsivadze, and Elena Marchiori helped us with their machine learning knowledge and pointed us to literature that might be relevant for this task. We thank the PAAR 2012 referees for their thorough reviews and comments that helped to improve the final presentation of this paper.

## References

[1] Jesse Alama, Tom Heskes, Daniel Kühlwein, Evgeni Tsivtsivadze, and Josef Urban. Premise Selection for Mathematics by Corpus Analysis and Kernel Methods. *CoRR*, abs/1108.3446, 2011.

[2] Gregory J. Chaitin. The Omega Number: Irreducible Complexity in Pure Math. In Jonathan M. Borwein and William M. Farmer, editors, *MKM*, volume 4108 of *Lecture Notes in Computer Science*, page 1. Springer, 2006.

[3] Leonardo Mendonça de Moura and Nikolaj Bjørner. Z3: An Efficient SMT Solver. In C. R. Ramakrishnan and Jakob Rehof, editors, *TACAS*, volume 4963 of *Lecture Notes in Computer Science*, pages 337–340. Springer, 2008.

[4] Krystof Hoder and Andrei Voronkov. Sine Qua Non for Large Theory Reasoning. In Nikolaj Bjørner and Viorica Sofronie-Stokkermans, editors, *CADE*, volume 6803 of *Lecture Notes in Computer Science*, pages 299–314. Springer, 2011.

[5] Daniel Kühlwein, Twan van Laarhoven, Evgeni Tsivtsivadze, Josef Urban, and Tom Heskes. Overview and Evaluation of Premise Selection Techniques for Large Theory Mathematics. In *IJCAR*, 2012. To appear.

[6] Huan Liu and Hiroshi Motoda. *Instance Selection and Construction for Data Mining.* Kluwer Academic Publishers, Norwell, MA, USA, 2001.

[7] Alexandre Riazanov and Andrei Voronkov. The Design and Implementation of Vampire. *AI Commun.*, 15(2-3):91–110, 2002.

[8] Stephan Schulz. E - A Brainiac Theorem Prover. *AI Commun.*, 15(2-3):111–126, 2002.

[9] Josef Urban, Krystof Hoder, and Andrei Voronkov. Evaluation of Automated Theorem Proving on the Mizar Mathematical Library. In *ICMS*, pages 155–166, 2010.

# A Resolution Calculus for Second-order Logic with Eager Unification

Alexander Leitsch Tomer Líbal
Vienna University of Technology
Vienna, Austria
{leitsch,shaolin}@logic.at

**Abstract**

The Effectiveness of the first-order resolution calculus is impaired when lifting it to higher-order logic. The main reason for that is the semi-decidability and infinitary nature of higher-order unification problems, which requires the integration of unification within the calculus and results in a non-effective search for refutations. We present a modification of the constrained resolution calculus (Huet'72) which uses an eager unification algorithm while retaining relative-completeness with regard to bounded unifiers. The first modification is the replacement of the unification rules with that of the bounded unification algorithm in (Líbal'12). This algorithm computes either pre-unifiers, or smaller unification problems which have terms containing Kleene stars. Following a result about an upper bound for these problems (Schmidt-Scauß,Schulz'98), the Kleene stars can effectively be replaced by natural numbers if we are interested in minimal unifiers only and the algorithm then decides the unifiability problem. Since computing minimal unifiers is not enough for the completeness of the calculus, we make a second modification and allow increases of these natural numbers. By applying a semi-eager strategy, we can always eagerly answer the unifiability question of a set of unification constraints while non-minimal unifiers are obtained via back-tracking and the increase of these numbers.

# Exploiting parallelism in the $\mathcal{ME}$ calculus

Tianyi Liang and Cesare Tinelli

Department of Computer Science
The University of Iowa

**Abstract**

We present some parallelization techniques for the Model Evolution ($\mathcal{ME}$) calculus, an instantiation-based calculus that lifts the DPLL procedure to first-order clause logic. Specifically, we consider a restriction of $\mathcal{ME}$ to the EPR fragment of clause logic for which the calculus is a decision procedure. The main operations in $\mathcal{ME}$'s proof procedures, namely clause instantiation and candidate literal generation, offer opportunities for MapReduce-style parallelization. This term/clause-level parallelization is largely orthogonal to the sort of search-level parallelization performed by portfolio approaches. We describe a hybrid parallel proof procedure for the restricted calculus that exploits parallelism at both levels to synergistic effect. The calculus and the proof procedure have been implemented in a new solver for EPR formulas. Our initial experimental results show that our term/clause-level parallelization alone is effective in reducing runtime and can be combined with a portfolio-based approach to maximize performance.

## 1 Introduction

The $\mathcal{ME}$ calculus [3] is an instantiation-based calculus that lifts to first-order logic without equality the popular DPLL procedure for propositional logic [6]. Like DPLL, it works with formulas in clause form, maintains at all times a *candidate model* for the input clause set, and keeps modifying that model until it finds one that satisfies all input clauses, or it determines that the clause set has no models. The main difference with DPLL is that the input clauses need not be ground and the candidate model is a Herbrand structure, represented finitely by a set of literals, called a *context*, instead of simple truth assignment. The calculus combines DPLL-style rules, such as unit propagation and splitting, with unification operations that generate instances of input clauses potentially not satisfied by the current candidate model. Such instances provide literals that can be added to the current context to obtain a new candidate model incrementally from the old one as needed. Implementations of $\mathcal{ME}$ benefit from enhancements similar to those developed for CDCL SAT solvers—the modern descendant of the DPLL procedure—such as conflict driven backjumping, lemma learning, and so on.

Roughly speaking, and ignoring those enhancements, a typical proof procedure for $\mathcal{ME}$ is a backtracking procedure relying on the following main data structures: a *context* $M$, a set of literals; a clause set $F$; a set $R$ of candidate *decision (or split) literals.* The procedure starts with $F$ consisting of the input clauses, $R$ empty, and $M = \{\neg v\}$ denoting a Herbrand structure in which every ground atom is false. Then, it repeatedly performs the following. By unifying clauses in $F$ with literals in $M$, it generates new *propagation* literals and adds them to $M$.[1] Then it identifies instances of clauses in $F$ that are possibly falsified (by the structure denoted) by the context $M$. A simple syntactic check is used to determine if the context is *unrepairable*, i.e., both $M$ and any of its enlargements definitely falsify one of those instances. If the context is instead repairable, the proof procedure uses the generated clause instances to compute new possible decision literals, and adds them to the candidate set $R$. Finally, it picks a literal from $R$

[1] Intuitively, these are literals all of whose ground instances *must* be satisfied from that point on; their addition prevents future extensions of $M$ that would break this requirement.

according to some selection heuristic, creates a new decision point, and adds the chosen literal or its complement to the context. When the current context is not repairable, the procedure backtracks to a previous decision point, if any, and replaces the corresponding decision literal $l$, and all literals added after that, by the complement of $l$. The process ends when no more instances of $F$ are falsified by $M$, which means that $F$ is satisfiable, or when $M$ is unrepairable but there are no decision points to backtrack to, which indicates that $F$ is unsatisfiable.

## 1.1 Parallelizing $\mathcal{ME}$

In sequential implementations of $\mathcal{ME}$'s proof procedure, computing propagation and decision literals takes a considerable portion of the runtime. This computation, however, presents several opportunities for large scale parallelization of some of the term-level and clause-level operations involved. The work presented here was motivated by the conjecture that such parallelization would be effective in reducing execution times.

The proof procedure's exploration of the search space is determined in large part by a heuristic selection of decision literals—the other main factor being the backtracking heuristics. Decision literal selection offers *its own* parallelization opportunities that can be exploited for instance with a portfolio-style approach using concurrent proof procedures with different selection heuristics. The success of portfolio approaches in CDCL SAT solvers suggests that $\mathcal{ME}$ proof procedures could benefit from them as well. However, in our case it was not obvious how term/clause-level parallelization might interact with a portfolio-style one.

To investigate these opportunities and their interactions we designed and implemented a parallel proof procedure for $\mathcal{ME}$. For simplicity, with started with a restriction of $\mathcal{ME}$ to *EPR clauses*, (universal) clauses whose literals may contain variables, constants but no function symbols. While an extension of this work to the whole clause logic fragment is left to future work the restriction to the EPR case is interesting in its own right because (*i*) $\mathcal{ME}$ yields (practical) decision procedure for the satisfiability EPR formulas and (*ii*) many interesting problems can be recast as EPR satisfiability problems [16, e.g.]. Our initial experimental results show that both term/clause-level and portfolio-style parallelizations produce significant speed ups for $\mathcal{ME}$. Furthermore, the two have largely orthogonal effects and so combine nicely to produce better runtime results than either of them alone.

## 1.2 Related Work

Parallel approaches in first-order theorem proving have been categorized at three different levels: term, clause and search level [5]. In term-level approaches parallelize operations such as term matching or unification; clause-level approaches parallelize operations such as deduction of new clauses or backward subsumption; search-leval approaches parallelize the exploration of the search space. Because first-order calculi can generate thousands of clauses, each with tens of literals, parallelism at term or clause level requires sophisticated data structure and scheduling algorithms. Usually, the overhead of thread scheduling overcomes the benefit of concurrency. To our knowledge, with one exception [10], there has been no new work on term/clause level parallelism in first-order theorem proving after a number discouraging attempts (see [5] again) done in the 1990s.

Search level parallelism has received most of the attention, especially in SAT and SMT solving, with two approaches: Guiding Path and Portfolio. The Guiding Path approach, introduced in PSATO [20], follows a Master/Slave architecture. A Master process initiates several sequential sub-solvers with partial models which partition the whole search space into several disjoint parts. If any sub-solver gets a satisfiable model, the problem is satisfiable; otherwise, it is unsatisfiable if all fail. This concept was further extended with a job stealing heuristics in ySAT[8]. By encoding QBF into SAT, QMiraXT became the first published parallel QBF solver[14]. A similar QBF solver can be found in [15, 9].

In the portfolio approach, introduced in ManySAT[11], a master process initializes several sub-solvers with different heuristics and makes them compete, of the same search space. The approach requires the various heuristics to be diverse enough. Complementary heuristics often generate super-linear speedups. This approach was also described as using different random seeds in [4]. Due to its success of SAT, the portfolio concept was also lifted to SMT solvers, with similar levels of success [19].

Lemma sharing is also an important factor in parallel SAT solvers. The portfolio approach benefits not only from the competition between subsolvers, but also from their cooperation through the exchange of lemmas [11]. There are two main shortcomings in lemma sharing. First, the set of shared lemmas may grow too large and possibly contain lemmas that are irrelevant to most subsolvers. Second, lemma communication can be a major source of overhead. Some SAT solvers minimize these problems by sharing only unit lemmas [4].

MapReduce is a software framework for processing large sets of data in a distributed system [7]. Very generally speaking, its main idea is to divide a large but highly distributable problem into small parts that can be processed completely independently (map step), and then compute the final result by combining the processed parts (reduce step). The sort of term/clause level parallelization that we describe in this paper could be seen as example of MapReduce.

## 1.3 Technical Preliminaries

The version of the $\mathcal{ME}$ calculus we consider here works in the *EPR fragment* of first-order clause logic without equality, which is restricted to clauses with no function symbols of positive arity. We use two disjoint sets of *variables*: a set $\mathcal{X}$ of *universal variables* and a set $\mathcal{P}$ of *parametric variables*.[2] We also use two disjoint sets of *constants*: a set $\mathcal{A}$ of *input constants* and a set $\mathcal{SK}$ of *Skolem constants*, with $\mathcal{C} = \mathcal{A} \cup \mathcal{SK}$. A *term* is either a constant or a variable. *Atoms*, *literals* (denoted by $k, l, l_0, k_0, \ldots$) and *clauses* ($C, C_0, \ldots$) over the set of terms above are defined as usual. We write $\neg l$ to denote the complement of a literal $l$; $l_0 \vee l_1 \vee \cdots \vee l_n$ to denote a clause $C$ modulo AC of $\vee$; $|C|$ to denote the number of literals in $C$; and $\Box$ to denote the empty clause. A *Skolemization* of a literal $l$, denoted by $l^{\mathrm{sko}}$ is any literal obtained by replacing each *universal* variable in $l$ by a fresh Skolem constant. A *substitution* $\sigma$ is an idempotent function from variables $\mathcal{X} \cup \mathcal{P}$ to terms $\mathcal{X} \cup \mathcal{P} \cup \mathcal{C}$ such that the set $\mathcal{D}om(\sigma) = \{z \in \mathcal{X} \cup \mathcal{P} \mid z\sigma \neq z\}$ is finite. Substitutions extend to terms and clauses as usual. We use the standard notions of unifier and most general unifier. The *join* $\sigma \bowtie \rho$ of two substitutions $\sigma$ and $\rho$ is the most general simultaneous unifier of the set $\{\{z, z\sigma\} \mid z \in \mathcal{D}om(\sigma)\} \cup \{\{z, z\rho\} \mid z \in \mathcal{D}om(\rho)\}$ when such a unifier exists—otherwise it is undefined. For notational convenience, we will treat the join operator as left associative. A substitution $\sigma$ is *p-preserving* if its restriction to $\mathcal{P}$ is a bijection onto $\mathcal{P}$. It is a *p-renaming* if it is p-preserving and its restriction to $\mathcal{X}$ is a bijection onto $\mathcal{X}$. A literal $l'$ is a *p-variant* of a literal $l$ if $l\sigma = l'$ for some p-renaming $\sigma$. For any literals $l_0, l_1$, we write $l_0 \geq l_1$ if $l_0\sigma = l_1$ for some p-preserving substitution $\sigma$; we call $l_1$ a *p-instance* of $l_0$. If $L$ is a set of literals, we write $L \geq l_1$ if $l_0 \geq l_1$ for some $l_0 \in L$. We denote respectively by $Par(l)$ and $Var(l)$ the set of all parametric and all universal variables occurring in literal $l$. A literal $l$ is *ground* if $Var(l) = Par(l) = \emptyset$; *universal* if $Par(l) = \emptyset$; and *pure* if either $Var(l) = \emptyset$ or $Par(l) = \emptyset$, or both.

[2]Parametric variables were called *parameters* in earlier papers on $\mathcal{ME}$.

## 2 A transition system for the $\mathcal{ME}$ calculus

To make the paper more self contained, we provide in this section a more formal description of the variant of $\mathcal{ME}$ used in this work. We refer the reader to [3] for more details on $\mathcal{ME}$.

As with formal and abstract treatments of the DPLL procedure and its extensions to Satisfiability Modulo Theories [17, 13, e.g.], one can formalize general classes of proof procedures for $\mathcal{ME}$ in a way that makes it easy to model and analyze operational features like backtracking and learning. An $\mathcal{ME}$ proof procedure can be described abstractly as a transition system over *states* of the form $\mathsf{unsat}$, a distinguished fail state, or the form $\langle \mathsf{M}, \mathsf{F}, \mathsf{R}, \mathsf{A} \rangle$ where $\mathsf{F}$ is a clause set, $\mathsf{M}$ is a *context*, $\mathsf{R}$ is a set of *remainders*, and $\mathsf{A}$ is a set of *propagation* literals (see below). We model generic $\mathcal{ME}$ proof procedures by means of a set of states of the kind above together with a binary *transition relation* over these states defined by means of *transition rules*. For a given state $S$, a transition rule precisely defines whether there is a transition from $S$ by this rule and, if so, to which state $S'$. A proof procedure can then be abstracted by a *transition system*, a set of transition rules defined over states, together with a strategy to generate executions in the system. We introduce a basic transition system for $\mathcal{ME}$ in the following.

**Contexts and context unifiers** A context $M$ is a finite sequence of *decision points* ($\bullet$), and pure literals. A literal of $M$ is *decision literal* if it immediately follows a decision point. Every maximal decision-point-free subsequence $M_i$ of a context $M = M_0 \bullet M_1 \bullet \cdots \bullet M_n$ is a *decision level* of $M$. When convenient, we will treat a context as a set. A literal $l$ is *contradictory* with a context $M$, written $l \perp M$, if $l\sigma = \neg k\sigma$ for some p-preserving substitution $\sigma$ and p-variant $k$ of a literal in $M$. We write $l \not\perp M$ if $l$ is not contradictory with $M$.

**Definition 1.** *Let $M$ be a context and $C = l_0 \vee \ldots \vee l_{m-1} \vee l_m \vee \ldots \vee l_n$ be a clause with $0 \leq m \leq n$. A substitution $\sigma$ is a context unifier of $C$ against $M$ with remainder $r = l_m\sigma \vee \ldots \vee l_n\sigma$ if the following hold for some fresh p-variants $k_0, k_1, \ldots, k_n$ of literals in $M$:*

1. *(i) $\sigma$ is a simultaneous most general unifier of $\{\{k_0, \neg l_0\}, \ldots, \{k_n, \neg l_n\}\}$,*
2. *(ii) $Par(k_i)\sigma \subseteq \mathcal{P}$ for $i = 0 \ldots m-1$,*
3. *(iii) $Par(k_i)\sigma \not\subseteq \mathcal{P}$ for $i = m \ldots n$.*

*The context unifier $\sigma$ is also admissible if the literals in $r$ are pure and do not share universal variables. We write $M|_\sigma C$ to denote the remainder of an admissible context unifier $\sigma$ of $C$ against $M$.*

We observe that any context unifier can be turned into an admissible one by renaming selected universal variables to parametric ones.

A clause $C$ *conflicts* with a context $M$ *via* a context unifier $\sigma$, written $C \perp_\sigma M$, if $\sigma$ is an admissible context unifier of $C$ against $M$ with an empty remainder. We write $C \perp M$ (resp., $C \not\perp M$) if $C \perp_\sigma M$ for some (resp., no) $\sigma$.

### 2.1 The transition rules

The transition rules of the system are listed in Figure 1. Each rule operates on a current state of the form $\langle \mathsf{M}, \mathsf{F}, \mathsf{R}, \mathsf{A} \rangle$, and modifies some of its components. The $\mathsf{A\text{-}Add}$ rule identifies a propagation (or *assert*) literal and adds it to the set $\mathsf{A}$ of propagation literals, while removing from $\mathsf{A}$ all p-instances of the new literal. The $\mathsf{A\text{-}Remove}$ rule removes from $\mathsf{A}$ any literal that has become contradictory with or a p-instance of the current context $\mathsf{M}$. The $\mathsf{R\text{-}Add}$ rule generates

$$\textsf{A-Add}\ \frac{C \vee l \in \mathsf{F} \quad C \perp_\sigma \mathsf{M} \quad Par(l\sigma) = \emptyset \quad \mathsf{A} \not\geq l\sigma}{\mathsf{A} := \{a \in \mathsf{A} \mid l\sigma \not\geq a\} \cup \{l\sigma\}}$$

$$\textsf{A-Remove}\ \frac{\mathsf{A} = \{l\} \cup A \quad l \perp \mathsf{M} \text{ or } \mathsf{M} \geq l}{\mathsf{A} := A} \qquad \textsf{R-Add}\ \frac{C \in \mathsf{F} \quad |C| > 1 \quad r = \mathsf{M}|_\sigma C \quad r \notin \mathsf{R}}{\mathsf{R} := \mathsf{R} \cup \{r\}}$$

$$\textsf{Assert}\ \frac{\mathsf{A} = \{l\} \cup A \quad l \not\perp \mathsf{M} \quad \mathsf{M} \not\geq l}{\mathsf{A} := A \quad \mathsf{M} := \mathsf{M}\, l} \qquad \textsf{Decide}\ \frac{\mathsf{R} = \{l \vee C\} \cup R \quad l \not\perp \mathsf{M} \quad \neg l^{\text{sko}} \not\perp \mathsf{M} \quad \mathsf{A} = \emptyset}{\mathsf{R} := R \quad \mathsf{M} := \mathsf{M} \bullet l}$$

$$\textsf{Backjump}\ \frac{\mathsf{M} = M' \bullet l\, M'' \quad \mathsf{F} \models C \quad C \perp \mathsf{M} \quad C \not\perp M'}{\mathsf{M} := M' \neg l^{\text{sko}} \quad \mathsf{A} := \emptyset} \qquad \textsf{Fail}\ \frac{\bullet \notin \mathsf{M} \quad C \in \mathsf{F} \quad C \perp \mathsf{M}}{\textsf{unsat}}$$

Figure 1: Transition Rules

a new remainder from a non-unit clause $C$ and adds it to the set $\mathsf{R}$ of remainders. The Assert rule moves a propagation literal to the current context provided that the literal is neither contradictory with nor a p-instance of the context. The Decide rule selects a literal $l$ from the available remainders in $\mathsf{R}$ and adds it as a decision literal to context, provided that neither $l$ nor its Skolemized complement is contradictory with the context.

The Backjump rule removes one or more decision levels from the current context and replaces the oldest of the removed decision literals by its Skolemized complement. A *backjump clause* $C$ entailed by the clause set is used to determine how far to backjump. In actual implementations this clause can be computed from an input clause that conflicts with the current context, by using a conflict resolution mechanism similar to the one used in CDCL SAT solvers. The Fail rule applies, producing the distinguished state unsat, if an input clause conflicts with the current context and the context contains no decision points (to backtrack to).

An actual implementation of the transition system would remove from the set $\mathsf{R}$ when backjumping all remainders computed using literals no longer in the context $\mathsf{M}$. We do not model this here just to simplify the description of the rules and because keeping stale remainders in $\mathsf{R}$ does effect correctness. Also for simplicity, we hardcode into the rules the heuristics that choses to process all current propagation literals before applying Decide. This is unnecessary for correctness if Backjump is modified to remove from $\mathsf{A}$ propagation literals computed using context literals no longer in $\mathsf{M}$ after the backjump.

Although we will not show it here, any fair execution of the transition system above starting with a state where $\mathsf{M} = \{\neg v\}$ and all the other fields except $\mathsf{F}$ are empty terminates in the unsat state if and only if the clauses in $\mathsf{F}$ are jointly unsatisfiable.

## 3 A Sequential Proof Procedure

In this section, we describe in some detail a sequential proof procedure we have implemented for the transition system in the previous section. The procedure mimics closely the behavior of the Darwin theorem prover [1], a full implementation of $\mathcal{ME}$ for clause logic without equality, when run on an EPR problem. The procedure uses these data structures for the main components of a state: a context, a priority queue of propagation literals, a priority queue of remainders, a set of clauses. Its main loop consists of the following steps:

1. **Propagation.** The highest-priority literal from the propagation queue, if any, is removed

from the queue. If it is neither p-subsumed by nor contradictory with the context, it is added to the context; otherwise, it is discarded and this step repeats.

2. **Decision.** If the context was unchanged by the previous step, the remainder with the highest-priority literal among all the remainder literals is removed from the remainder queue, if any. Then, that literal is added to the context if it is neither p-subsumed by nor contradictory with the context. Otherwise, the remainder is discarded and this step is repeated.

3. **Unit Context Unifier Calculation.** If the context was extended in the previous steps, the newly added context literal $k$ is unified with the complement of each literal $l$ in each clause in the clause set. Each most general unifier computed this way, is stored as a *unit context unifier* for $l$.

4. **Remainder Generation.** The procedure identifies all sets $\{\sigma_1, \ldots, \sigma_n\}$ of substitutions where (1) for $i = 1, \ldots, n$, $\sigma_i$ is a unit context unifier for literal $l_i$ in some clause $l_1 \vee \cdots \vee l_n$, and (2) one (or more) of the $\sigma_i$'s was newly computed in the previous step. For each of these sets $\{\sigma_1, \ldots, \sigma_n\}$, the substitution $\sigma_1 \bowtie \cdots \bowtie \sigma_n$, when defined, is a context unifier of the corresponding clause $l_1 \vee \cdots \vee l_n$. The procedure computes all such context unifiers, makes them admissible, and adds their remainder to the remainder queue.[3] The process is interrupted, however, as soon as a context unifier with an empty remainder is computed. In that case, the procedure moves immediately to Step 6.

5. **Propagation Literal Generation.** A process similar to the one in the previous substep is used to generate propagation literals and add them to the propagation queue.[4]

6. **Backjumping.** After an analysis of the conflict represented by the empty remainder computed in the previous step, the procedures identifies a previous decision level $d$ to backtrack to. If $d$ is the top level, the procedure ends with an "unsatisfiable" result. Otherwise, it clears the propagation queue, undoes all additions to the context and to the remainder queue from that level on, adds to the context (at decision level $d-1$) the Skolemized complement $l^{\text{sko}}$ of the decision literal $l$ of $d$, and resumes the main loop from Step 1.

If neither of the first two steps is able to add literals to the context, the main loop aborts and the procedure terminates with success: the context denotes a model of the clause set.

**Selection Heuristics** In our current implementation, the priority function used in the propagation queue is determined by a ranking of literals where propositional literals[5] are preferred over (i.e., have a higher rank than) universal ones, which in turn are preferred over parametric literals. Among universal literals, those with more variables are preferred. Among parametric literals, those with less parameters are preferred. After the criteria above, literals introduced in the propagation queue in an earlier decision level are preferred. Any ties after that are broken in an arbitrary, but fixed, way. Something similar is done for the remainder queue. There, the same literal ranking as the one in the propagation queue is used first locally in each remainder, to select a literal, and then globally to chose among those selected literals.

For comparison, we also implemented two random selection heuristics: controlled random and totally random. The controlled random heuristics modifies the priority functions above

[3]We ignore here the enhancement that considers only *productive* context unifiers (see [3] for details).
[4]In reality, this step and Step 4 are interleaved. We present them here as sequential for simplicity.
[5]That is, literals with a 0-arity predicate symbol.

by breaking the final ties randomly (as opposed to a fixed way). With the totally random heuristics no priority function is actually used. The dequeue operation in both queues simply picks an element from the queue at random.

## 4 A Parallel Proof Procedure

The sequential proof procedure highlighted above presents several opportunities for parallelization. We have focused on parallelizing two main aspects: the computation of context unifiers and the exploration of the search space.

**Term/Clause-level Parallelization** At each iteration of the sequential proof procedure most of the computation is spent in the generation of context unifiers. Since the individual unit context unifiers computed in Step 3 are completely independent from each other, they can all be computed in parallel, MapReduce style. The computation of the context unifiers done in Step 4 (by joining unit context unifiers) can be parallelized in a similar way for each clause in the clause database.

**Search-level Parallelization** As in DPLL, in $\mathcal{ME}$ the exploration of the search space is driven by the selection of the next decision literal. In fact, since empty remainders trigger a backjump as soon as they are generated, the exploration is also driven by the order in which propagation literals are chosen. Our experiments on candidate selection confirm that, again as in DPLL, the selection heuristics can have a significant impact on performance for some problems. To account for that we also implemented a portfolio-based approach [11] where the input problem is given to several subsolvers running independently from each other. The subsolvers differ only for the candidate selection heuristics they use for the propagation and the remainder queues. They run completely independently except that they are all stopped once one of them proves or disproves the input problem.

### 4.1 General Architecture

Our parallel proof procedure follows the actor model of computation, and relies on a small number of actor classes. All actors communicate asynchronously via message queues and run in their own computation thread. The actor model considerably simplifies the implementation of parallel systems with respect to the shared-memory model. It also minimizes synchronization needs, leading to less overhead and greater scalability with the increase of computational resources.

**Main Actors** The main actors in our architecture, sketched in Figure 2, are one Main Solver, one Context Manager, one or more Clause Managers, one Unification Pool, and one or more Candidate Generators. Roughly, the **Main Solver** parses the input formulas, sets ups a number of data structures, creates the other actors, and then passes control to the Context Manager. The **Context Manager** manages the context data structure, and is the one effectively applying the rules of the calculus by selecting literals to add to the context, analyzing conflicts caused by empty remainders, deciding where to backjump, and shrinking the context accordingly. A **Clause Manager** is responsible for one or more input clauses and for the computation of unit context unifiers for them. In an ideal situation, with the system running on a unlimited number of parallel computational (e.g., cores), we would have one clause manager per clause. In reality, the clause set is partitioned so that each Clause Manager manages several clauses

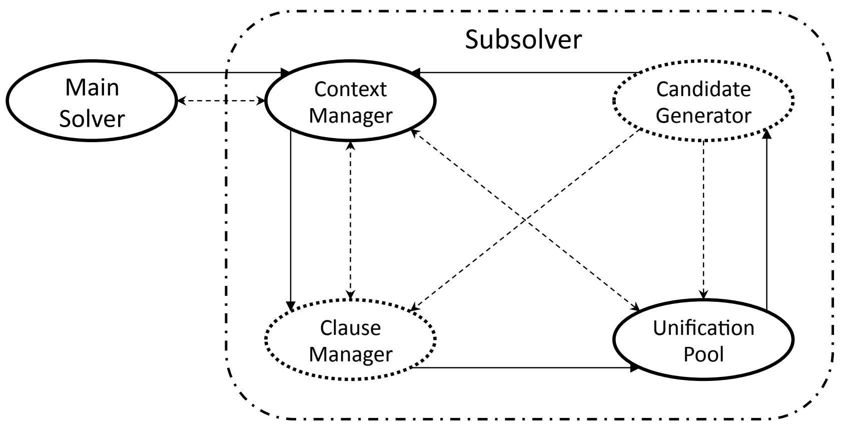


Figure 2: Architecture of the parallel EPR Solver. The solid lines represent data flow, the dash lines represent control flow. The dotted ovals stand for multiple actors.

sequentially, to reduce the number of threads running on the same core. The **Unification Pool** is in charge of computing context unifiers, by merging unit context unifiers received from the Clause Manager, as well as computing propagation literals or remainders from those context unifiers. It performs its functions by delegating them to one or more **Candidate Generators** it manages. It is essentially a scheduler, creating and assigning unification tasks to the Candidate Generators as they became available.

The portfolio extension adds several subsolver actors below the Main Solver, each relying on its own Context Manager, Clause Managers, Unification Pool, and Candidate Generators according to the architecture above.

## 4.2 Synchronization

The parallel proof procedure has been designed to minimize the need for the various actors to synchronize with one another while also minimizing runtime differences between identical runs when no randomized selections heuristics are used. The main synchronization points are discussed below.

**Synchronization between sub-solvers.** Once started, the subsolvers are completely independent from the main solver and each other. Synchronization with the main solver occurs only once a subsolver solves the input problem, i.e., determines if it is satisfiable or not, at which point the main solver terminates all subsolvers, outputs a response, and quits.

**Synchronization before candidate selection.** Before selecting a candidate as a new context literal, the Context Manager waits for all context unifier computations in each Candidate Generator to terminate, to make sure that all possible candidates are available for selection in the propagation and the remainder queue.

**Synchronization when backjumping.** When an empty remainder is generated, the Context Manager is immediately notified. In turn, it immediately instructs all Clause Managers and the Unification Pool to abort their computation, and waits for an acknowledgment from them before backjumping and sending a new context literal to the Clause Managers. Waiting for an acknowledgment is needed because the actor model does not guarantee

| | Seq | S1C1U1 | S1C3U20 | S4C1U1 | S4C3U20 | CR | TR |
|---|---|---|---|---|---|---|---|
| **EPT Solved** | 161 | 175 | 199 | 197 | 222 | 213 | 186 |
| **EPS Solved** | 125 | 125 | 128 | 129 | 134 | 135 | 128 |
| **Total Solved** | 286 | 300 | 327 | 326 | 356 | 348 | 314 |
| **Median runtime** (s) | 1.48 | 1.82 | 1.61 | 1.52 | 1.81 | 1.77 | 1.43 |
| **Average runtime** (s) | 11.83 | 19.93 | 27.29 | 27.46 | 27.23 | 32.62 | 18.74 |
| **Med speedup** | | 0.93 | 1.27 | 1.21 | 1.38 | 1.35 | 1.35 |
| **Avg speedup** | | 1.29 | 1.67 | 1.88 | 2.34 | 2.16 | 1.96 |
| **Max speedup** | | 11.99 | 14.27 | 45.50 | 43.62 | 42.36 | 39.26 |
| **Med speedup** ($\geq$ 5s) | | 1.75 | 1.93 | 2.16 | 2.93 | 2.61 | 2.48 |
| **Avg speedup** ($\geq$ 5s) | | 2.64 | 3.21 | 4.18 | 5.59 | 5.10 | 4.25 |

Figure 3: Results for several solver configurations over the EPS and EPT problems of TPTP. The columns represent the tested solver configurations: the sequential solver (Seq); the parallel solver with $i$ subsolvers, $j$ clause managers, and $k$ candidate generators in the unification pool (S$i$C$j$U$k$); the parallel solver with 4 subsolvers each using the controlled random heuristics (CR) or the totally random heuristics (TR).

that messages received by an actor in the same order they were sent. Note that an actor's computation cannot be interrupted by another actor. Since Candidate Generators take a while to complete their tasks, they are not notified about a backjump and just let run to completion. However, they are required to time-stamp, with the number of backjumps so far, the candidates they compute. This way, such candidates can be discarded if they arrive to the Context Manager when they are no longer current because of a later backjump.

In our current implementation of the parallel proof procedure several parallelization parameters such as the number of subsolvers, clause managers and candidate generators, are controlled by user-configurable options. Since we focus on parallel strategies, we did not implement at this time any preprocessing simplifications on the input clause set.

## 5 Experimental Evaluation

We evaluated the performance of our implementation of the sequential and the parallel proof procedures described above against the EPR benchmarks of the TPTP library [18] which are divided into EPS problems (satisfiable clause sets) and EPT problems (unsatisfiable clause sets).[6] Since our current solvers do not include inference rules for equality yet, we focused on the 483 clausal problems without equality. Of those, 318 are EPT problems and 165 are EPS problems.

All tests were run on a computer with two 12-core AMD Opteron 6172 processors and 32Gb of memory, and running under Ubuntu 11.10. The solvers were developed in Scala, a language based on the Java Virtual Machine. We used OpenJDK 64-Bit Version 1.6.0 as the JVM engine. All experiments were run using the JVM option "-XX:+UseCompressedOops". Since CPU time is not very meaningful when measuring the *runtime* performance of parallel programs, we used wall clock time[7], with a timeout limit of 300 seconds.

[6]Detailed results together with the benchmark problems and our implementation can be found at `http://www.cs.uiowa.edu/~tiliang/paar12/`.

[7]Measured with the same utilities used at SMT-COMP 2011 (`http://www.smtcomp.org/2011/`).

## 5.1 Results

Our experimental results are summarized in Figure 5 for the baseline sequential solver and for several configurations of the parallel solver. The results refer to a single run of each configuration. Configuration S1C1U1 of the parallel solver uses a single subsolver, one clause group (managing all clauses), and one candidate generator (again for all clauses) in the unification pool. It is parallel only in that its various actors run concurrently. In contrast, S1C3U20 is more properly MapReduce-style for having 3 clause managers and 20 candidate generators. Configuration S4C1U1 uses 4 subsolvers which differ from each other only in the fixed way they break the final ties in their selection heuristics. Each subsolver has just one clause manager and one candidate generator, making this configuration essentially a pure portfolio-style solver. Configuration S4C3U20 is a hybrid resulting from the combination of the previous two. Configuration CR (resp. TR) is like S4C3U20 except that the subsolvers use the controlled (resp., totally) random selection heuristic, each with a different seed.

For each configuration, averages and median runtime values are computed over the problems solved by that configuration. Speedup factors are with respect to the runtimes of the Seq configuration, and computed for each problem solved by Seq. The rows marked with ($\geq$ 5s) remove from consideration *easy problems*, defined as problems solved by Seq in less than 5s (216/483). Focusing on those rows for the parallel configurations is instructive because for easy problems the overhead caused by thread initialization and scheduling generally cancels out most of the performance improvement due to parallel execution—in fact, it actually increases overall runtimes for most problems solved by Seq within 1 second.

## 5.2 Analysis

As we conjectured, the sequential solver exhibits the lowest success rate, measured as the percentage of problems solved within the time limit, solving only 59% of the 483 problems. All the parallel configurations solve a superset of the problems solved by Seq, with an increased success rate that goes from 62% for the minimally parallel S1C1U1 to 75% for the hybrid S4C3U20. The improvement provided by S1C1U1 shows that just computing unit context unifiers in parallel with joining such unifiers to generate candidates is already advantageous.

We experimented with a number of additional configurations (not reported here) differing from S1C1U1 only in the number of clause managers and the size of the unification pool. Our main conjecture was that increasing those parameters would lead to a greater success rate because of the relative independence of the computations performed by each clause group and each candidate generator. Our general findings confirm that conjecture, with a sweet spot provided by S1C3U20. Adding more clause managers or more candidate generators usually leads to a degraded performance, possibly because then, as we have verified, the number of threads significantly exceeds the number of physical cores. These experiments, together with additional ones on machines with fewer cores, show that the success rate of the S1C*U* configurations increases lineraly with the number of cores. This strongly suggest that our solver will scale up well, within the limits of Amdahl's law [12], as processors with more and more cores become available.

The very simple portfolio approach implemented by configuration S4C1U1 impressively achieves almost the same success rate (67%) as that of the more sophisticated MapReduce configuration S1C3U20 (68%). Its superiority to the sequential solver is consistent with similar findings by others on parallelizing SAT and SMT solvers. What is interesting in our case is that the MapReduce and the portfolio strategies are complementary to a certain extent, as shown in the scatter plot of Figure 4. In particular, each solves about 20 problems that the

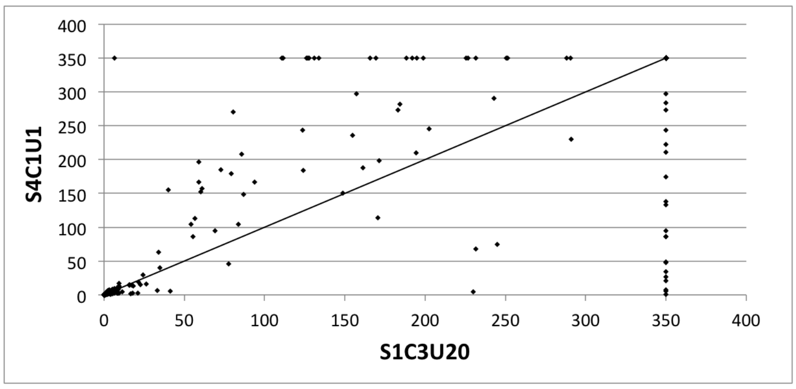


Figure 4: Comparative runtime performance of a MapReduce (S1C3U20) and a portfolio strategy (S4C1U1). Times are in seconds.

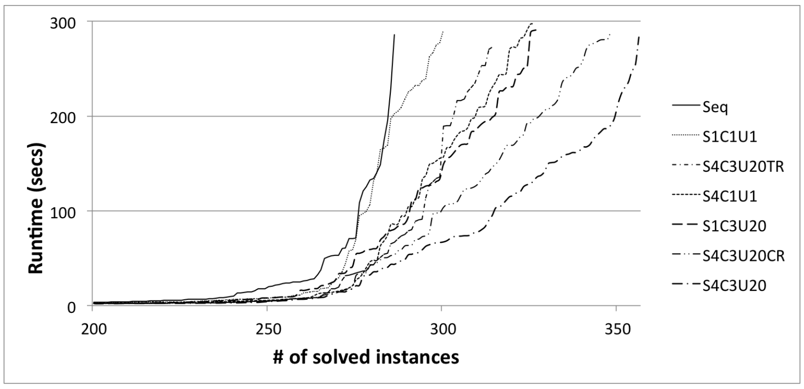


Figure 5: Runtime performance of all configurations.

other cannot solve. The same plot also shows that for problems solved by both, S1C3U20 is superior to S4C1U1 in terms of runtimes. The overall superiority of S1C3U20 is confirmed by a Wilcoxon rank-sum test on the whole set, which allows us to accept the alternative hypothesis that S1C3U20 is faster than S4C1U1 with a p-value smaller than 0.001.

We obtained similar results also for configuration S1C3U10 (not shown), which creates about as many threads as S4C1U1. It is possible that the subsolvers of S4C1U1 are not diverse enough to fully exploit the advantages of a portfolio approach. However, additional controlled experiments with one subsolver using random selection heuristics, for greater variability, did not improve the overall performance of S4C1U1.

The complementarity of S1C3U10 and S4C1U1 suggests that combining them could have a synergistic effect on performance. This is confirmed by the results obtained by S4C3U20 which can solve not only all the problems solved by S1C3U10 and by S4C1U1 individually, but also a few more. In fact, S4C3U20 is also faster than any other configuration on the problems solved by both. This is reflected by the average speed up factors in Figure 3. As with the pure portfolio strategies, adding randomization in the selection process, both in a controlled or uncontrolled fashion, did not improve the performance of S4C3U20 further, actually resulting in worse performance. The superiority S4C3U20 in general to all other configurations listed in Figure 3 is clearly shown in the chart of Figure 5, indicating again that clause-level and search-level parallelism can be combined to great effect in $\mathcal{ME}$.

Finally, we observe that the best average speedup factor we achieved for hard problems (5+) seems to be low with respect to the number of cores used. On the one hand, this contrasts with results achieved by the best portfolio SAT solvers [11, e.g.] whose average speedups versus a sequential version can be superlinear in the number of cores. On the other hand, our portfolio implementation is fairly unsophisticated yet and lacks crucial features such as lemma sharing. Also, EPR satisfiability is a much harder problem that propositional satisfiability (NEXPTIME vs. NP) and so it is possibly correspondingly harder to parallelize. So, while our results could be considered a good first step, more work and experimental evaluations are still needed.

## 6 Conclusion and Future Work

We have described concurrent proof procedures for the $\mathcal{ME}$ calculus that rely on term/clause-level as well as search-level parallelism. Our experiments provide initial evidence that the former is effective in reducing runtimes in instantiation-based theorem proving when using MapReduce-style approaches which minimize the interactions between concurrent threads. Our results show that, in addition to improving performance by themselves, such approaches also combine synergistically with the traditional portfolio approaches.

We are working on an enhancement of the proof procedure with lemma learning, and lemma sharing in the portfolio case. Lemma learning in $\mathcal{ME}$ is similar to lemma learning in SAT solvers but has its own distinct features for exploring at the first-order, as opposed to the propositional, level [2]. Further work will involve conducting further experimental evaluations on the effectiveness of lemma sharing between subsolvers in our parallel implementation.

## References

[1] P. Baumgartner, A. Fuchs, and C. Tinelli. Implementing the model evolution calculus. *International Journal of Artificial Intelligence Tools*, 15(1):21–52, Feb. 2006.

[2] P. Baumgartner, A. Fuchs, and C. Tinelli. Lemma learning in the model evolution calculus. In M. Hermann and A. Voronkov, editors, *Proceedings of the 13th International Conference on Logic for Programming, Artificial Intelligence and Reasoning (LPAR'06), Phnom Penh, Cambodia*, volume 4246 of *Lecture Notes in Computer Science*, pages 572–586. Springer, 2006.

[3] P. Baumgartner and C. Tinelli. The model evolution calculus as a first-order DPLL method. *Artificial Intelligence*, 172:591–632, 2008.

[4] A. Biere. Lingeling, Plingeling, PicoSAT and PrecoSAT at SAT Race 2010. FMV Reports Series Technical Report 10/1, Institute for Formal Models and Verification, Johannes Kepler University, 69, 4040 Linz, Austria, August 2010.

[5] M. P. Bonacina. A taxonomy of parallel strategies for deduction. *Annuals of Mathematics and Artificial Intelligence*, 29(1-4):223–257, 2000.

[6] M. Davis, G. Logemann, and D. Loveland. A machine program for theorem-proving. *Commun. ACM*, 5:394–397, July 1962.

[7] J. Dean and S. Ghemawat. Mapreduce: simplified data processing on large clusters. *Commun. ACM*, 51:107–113, Jan. 2008.

[8] Y. Feldman. Parallel multithreaded satisfiability solver: Design and implementation, 2005.

[9] M. Fränzle, C. Herde, T. Teige, S. Ratschan, and T. Schubert. Efficient solving of large non-linear arithmetic constraint systems with complex boolean structure. *Journal on Satisfiability, Boolean Modeling and Computation*, 1:209–236, 2007.

[10] J. Gaillourdet, T. Hillenbrand, B. Löchner, and H. Spies. The new Waldmeister loop at work. In F. Baader, editor, *Proceedings of the 19th International Conference on Automated Deduction, CADE-19*, volume 2741 of *Lecture Notes in Artificial Intelligence*, pages 317–321. Springer, 2003.

[11] Y. Hamadi and L. Sais. ManySAT: a parallel SAT solver. *Journal on Satisfiability, Boolean Modeling and Computation*, 6, 2009.

[12] M. Hill and M. Marty. Amdahl's law in the multicore era. *Computer*, 41(7):33–38, July 2008.

[13] S. Krstić and A. Goel. Architecting solvers for SAT modulo theories: Nelson-Oppen with DPLL. In B. Konev and F. Wolter, editors, *Proceeding of the Symposium on Frontiers of Combining Systems (Liverpool, England)*, volume 4720 of *Lecture Notes in Computer Science*, pages 1–27. Springer, 2007.

[14] M. Lewis, T. Schubert, and B. Becker. QMiraXT – A Multithreaded QBF Solver. In *GI/IT-G/GMM Workshop "Methoden und Beschreibungssprachen zur Modellierung und Verifikation von Schaltungen und Systemen"*, 2009.

[15] B. D. Mota, P. Nicolas, and I. Stéphan. A new parallel architecture for QBF tools. In *HPCS'10*, pages 324–330, 2010.

[16] J. A. Navarro Peréz. *Encoding and Solving Problems in Effectively Propositional Logic*. PhD thesis, The University of Manchester, 2007.

[17] R. Nieuwenhuis, A. Oliveras, and C. Tinelli. Abstract DPLL and abstract DPLL modulo theories. In F. Baader and A. Voronkov, editors, *Proceedings of the 11th International Conference on Logic for Programming, Artificial Intelligence and Reasoning (LPAR'04), Montevideo, Uruguay*, volume 3452 of *Lecture Notes in Computer Science*, pages 36–50. Springer, 2005.

[18] G. Sutcliffe. The TPTP Problem Library and Associated Infrastructure: The FOF and CNF Parts, v3.5.0. *Journal of Automated Reasoning*, 43(4):337–362, 2009.

[19] C. M. Wintersteiger, Y. Hamadi, and L. Moura. A concurrent portfolio approach to SMT solving. In *Proceedings of the 21st International Conference on Computer Aided Verification*, CAV '09, pages 715–720, Berlin, Heidelberg, 2009. Springer-Verlag.

[20] H. Zhang, M. P. Bonacina, M. Paola, Bonacina, and J. Hsiang. PSATO: a distributed propositional prover and its application to quasigroup problems. *Journal of Symbolic Computation*, 21:543–560, 1996.

# Synthesising and Implementing Tableau Calculi for Interrogative Epistemic Logics

Ştefan Minică
Amstelveen, The Netherlands
`stefan.minica@gmail.com`

Mohammad Khodadadi, Renate A. Schmidt, Dmitry Tishkovsky[*]
The University of Manchester, Manchester, UK
`khodadadi,dmitry,schmidt@cs.man.ac.uk`

**Abstract**

This paper presents a labelled tableau approach for deciding interrogative-epistemic logics (IEL). Tableau calculi for these logics have been derived using a recently introduced tableau synthesis method. We also consider an extension of the framework for a setting with questioning modalities over sequences of formulae called sequential questioning logic (SQL). We have implemented the calculi using two approaches. The first implementation has been obtained with the tableau prover generation software MetTeL2, while the other implementation is a prover implemented in `Haskell`.

## 1 Introduction

The paper focusses on developing and implementing automated reasoning tools for interrogative epistemic logics (IEL). Interrogative or erotetic logics have a long tradition alongside declarative and epistemic logics. Interrogative Epistemic Logic (henceforth, IEL) also referred to as DELQ (for Dynamic Epistemic Logic of Questions), enriches a standard multi-agent epistemic modal logic with interrogative components [15, 7]. Intuitively this is done by adding an "issue" relation over a set of possible worlds. This relation is meant to represent structural changes brought about by dynamic actions of raising and answering questions. Besides the standard epistemic modality the logic also introduces a static modality over the issue relation. This gives rise to interaction between the epistemic and interrogative components. Such aspects are captured by an intersection modality which is then used to describe how dynamic questioning effects depend on the structure of the issues raised and previous knowledge.

A second addition are the dynamic actions that express interrogative or epistemic events explicitly in the language. Their effect is to change the interrogative and epistemic states and to add more structure to the existing issue or epistemic relations.

While automated reasoning tools are widespread for declarative and epistemic modal logics, for interrogative epistemic logics there are currently no implemented automated reasoning systems. The usefulness of automating reasoning for other logics, such as epistemic modal logics, has been proven already by many applications. Very often in epistemic scenarios obtaining the relevant information is an essential part. Adding an interrogative component makes modelling and reasoning about obtaining relevant information possible.

For dynamic epistemic logics (DEL), automated reasoning tools focused so far on solving model-checking tasks [17]. Other tableau-based provers for modal logics, e.g., [3], incorporate dynamic modalities for informative actions, e.g., public announcements [4, 2]. However, none of the existing software tools contain questioning modalities and moreover, none of them offer a generic method to generate a prover for a logic starting from a semantic specifications.

[*]This research is supported by UK EPSRC research grant EP/H043748/1.

Planning in contexts involving interaction between questions and knowledge is reducible to testing validities of interrogative epistemic logic. However, the existing proof systems for dynamic epistemic logics [4, 14] are not fully automated. They are usually Hilbert-style calculi in which formulae with non factual content have special substitution rules.

A longstanding problem for dynamic logics is the fact that they are not closed under uniform substitution, and therefore, they are not suitable for an algebraic treatment and do not lend themselves well to automatic reasoning techniques. Previous research in this area focused on identifying substitution closed fragments of such logics, which can still preserve some of the features that have made dynamic logics so successful for modelling information exchange.

The approach of this paper gives an alternative solution based on first translating the semantics of the modal language into many-sorted first-order logic and then turning it into a tableau calculus for the corresponding first-order fragment. Based on this tableau calculus two tableau based reasoning tools have been developed for interrogative-epistemic logics.

The paper is structured as follows. We start in Section 2 by introducing the details of IEL. Then we apply the tableau synthesis framework to IEL in Section 3. In Section 4 we present an extended logic, called SQL, which uses sequences of formulae inside the dynamic modalities. We continue in Section 5 with introducing and discussing the MetTeL$^2$ implementation for IEL. In Section 6 we present and discuss the `Haskell` implementation `Qtab.lhs` for IEL which also illustrates the extension to questioning sequences. We draw some conclusions in the final section. Further implementation details and illustrative code output, which could not be included due to space limitations, can be found in the long version of the paper [8].

## 2 Interrogative Epistemic Logics

The approach of IEL [15, 7] starts by enriching a standard multi-agent epistemic modal logic with interrogative components. This is done in two stages. The first addition consists of a static modality over an "issue" relation. The intuitive meaning of this modality is close to the traditional epistemic notion, but instead of representing actual knowledge it stands for what the agents would like to find out. It represents future epistemic goals that are expressed by asking questions and will eventually be achieved by obtaining answers.

For technical reasons a third modality, expressing the interaction between the epistemic and the interrogative components is also introduced using the intersection of the standard epistemic relation and the newly introduce issue relation. This also has an intuitive meaning that goes beyond the traditional epistemic notion. It expresses how the future knowledge depends on both the current epistemic state of the agent and the epistemic goals guiding the ongoing questioning dynamics. This is what the agent will come to know if all the questions he raised so far would be answered one way or another.

In this paper nominals are added to the language alongside propositions. The second addition consists of two dynamic modalities, one for questioning actions or queries and one for answering actions or resolution actions. Intuitively, such modalities change the underlying structures by refining their component relations.

A formula $\varphi$ in the language of IEL is defined by the following BNF:

$$\varphi \quad ::= \quad n \mid p \mid \neg\varphi \mid \varphi \vee \varphi \mid \Box\varphi \mid [Q]\varphi$$

Here, $n, p, a$ denoting nominals, propositions, and agent labels, respectively. $\Box$ stands for modalities, that is $\Box \in \{Q_a, X_a, K_a\}$, respectively being static questioning, interaction, and epistemic modalities. Finally, $Q$ stands for actions that change the underlying models, that is

$Q \in \{\varphi?_a, !_a\}$, representing dynamic questioning actions and resolution actions, respectively. This language is meant to express minimal interrogative-epistemic facts. The @ operator, which is a standard addition for hybrid logics is introduced later in the specification language where it is useful for both internalizing the semantics and formulating labeled tableau rules.

The language can express the interaction between questions and information in two ways. First, by using a (static) intersection modality $X_a\varphi$. Second, through the dynamic modalities $[Q]$, encoding model-changing operations by means of questioning and resolution actions.

The logic has a standard modal semantics over issue-epistemic models, $M = \langle W, \overset{a}{\approx}, \overset{a}{\sim}, V\rangle$. When used inside a tuple representing a model $\overset{a}{\approx}$ and $\overset{a}{\sim}$ are meant as shorthand notations for $(\overset{a}{\approx})_{a\in A}$ respectively $(\overset{a}{\sim})_{a\in A}$ for $A$ the set of all agent labels. We use the expected Boolean clauses and the usual relational (modal) clauses involving $\overset{a}{\approx}$ for $Q_a$ and $\overset{a}{\sim}$ for $K_a$. We also use equivalence relations for $\approx$ and $\sim$ throughout the paper. However, if needed, the framework can be generalized to other structures by correspondingly changing the background theory.

The intersection modality $X_a$ is defined using $\overset{a}{\approx}\cap\overset{a}{\sim}$ as follows:

$$M \models_w X_a\varphi \quad \text{iff} \quad \forall v \in W : w\,(\overset{a}{\approx}\cap\overset{a}{\sim})\,v \Rightarrow M \models_v \varphi$$

Dynamic modalities express model changing operations of asking and resolution:

$[\varphi?]_a\psi$ "after $\varphi$ is asked, $\psi$ is the case" $M_? = \langle W, \overset{a}{\approx}_?, \overset{a}{\sim}, V\rangle$; $\overset{a}{\approx}_? = \overset{a}{\approx} \cap \overset{\varphi}{\equiv}_M$
$[\,!\,]_a\varphi$ "after having answered the questions raised, $\varphi$ is true"
$M_! = \langle W, \overset{a}{\approx}, \overset{a}{\sim}_!, V\rangle$; $\overset{a}{\sim}_! = \overset{a}{\sim} \cap \overset{a}{\approx}$

Here, $\overset{\varphi}{\equiv}_M = \{(w,v) \mid ||\varphi||^M_w = ||\varphi||^M_v\}$ is the set of $M$-world pairs in which $\varphi$ has the same truth value. The intuitive reading for a questioning action represented by the modality $[\varphi?]_a\psi$ is that of splitting the domain into $\varphi$ worlds and non-$\varphi$ worlds, by raising a question, or by making $\varphi$ an issue. The intuitive reading for the resolution action represented by the $[!]_a\varphi$ modality is to add new knowledge by refining the epistemic relation in such a way that afterwards all the issues raised so far are solved. Indexing the actions with agent labels can also model privacy in questioning or can be used to add agent-specific preconditions for question execution. However, we assume our actions to be 'public' and 'preconditionless', i.e., they affect the epistemic/questioning states for all agents, and they do not require any conditions for execution. For this reason, indexing the modalities with agent labels will only play a genuine role in this paper for the static part of the logic.

The language of IEL has two parts. The static part is a hybrid modal logic with nominals and intersection. The dynamic part adds the dynamic modalities capturing model changing operations. The *static part* of the logic is axiomatised by a customary hybrid logic system [12] with nominals, S5 axioms for $\sim$ and $\approx$, and an intersection axiom for static resolution expressed by the following pure formula:

$$\widehat{K}_a i \wedge \widehat{Q}_a i \leftrightarrow \widehat{X}_a i, \qquad \text{where } i \text{ is a nominal.}$$

Here, $\widehat{Q}_a, \widehat{K}_a$ and $\widehat{X}_a$ are the diamond modalities defined as the duals of the box modalities $Q_a, K_a, X_a$ introduced before.

The *dynamic part* of IEL introduces modalities which change the underlying static models. The logical behaviour of this new kind of connectives is captured by reduction axioms. These describe the relation between the underlying static structures before and after a questioning action or resolution action takes place. Formulas containing dynamic modalities can be reduced to equivalent static formulas using reduction axioms like the following ones (for $b \in \{n, p\}$, $\Box \in \{Q, X\}$ and $q \in \{\varphi?, !\}$):

$$[q]_a\, b \leftrightarrow b, \qquad [q]_a\neg\psi \leftrightarrow \neg[q]_a\psi, \qquad [q]_a(\psi \wedge \chi) \leftrightarrow [q]_a\psi \wedge [q]_a\chi \tag{1a}$$

$$[\,!\,]_a\Box_a\psi \leftrightarrow \Box_a[\,!\,]_a\psi, \quad [\,!\,]_aK_a\varphi \leftrightarrow X_a[\,!\,]_a\varphi, \quad [\varphi?]_aK_a\psi \leftrightarrow K_a[\varphi?]_a\psi \tag{1b}$$

$$[\varphi?]_aQ_a\psi \leftrightarrow (\varphi \wedge Q_a(\varphi \rightarrow [\varphi?]_a\psi)) \vee (\neg\varphi \wedge Q_a(\neg\varphi \rightarrow [\varphi?]_a\psi)) \tag{1c}$$

$$[\varphi?]_aX_a\psi \leftrightarrow (\varphi \wedge X_a(\varphi \rightarrow [\varphi?]_a\psi)) \vee (\neg\varphi \wedge X_a(\neg\varphi \rightarrow [\varphi?]_a\psi)) \tag{1d}$$

This treatment is in line with the generic DEL methodology introduced in [4, 14] extended to include an additional interrogative component. Further technical details, possible extensions and examples of applications of IEL can be found in [7, 15].

We start from IEL as minimal logic when synthesizing and implementing the tableau calculus. Several extensions of the framework that handle various levels of privacy for epistemic-questioning actions can be added in a modular way using the same general synthesis method.

- Multi-agent questioning preconditions
- Group-opaque dynamic questioning effects
- Epistemic indistinguishability in questioning
- Dynamic questioning sequences

Due to lack of space in this paper we will discuss in detail only the last extension.

The extension requires questioning actions of a more complex type capable to model modalities over sequences of questions not just one formula. We briefly motivate now why such an extension is desirable and useful. The concrete details of the extension method are introduced later in Section 4, after all the needed details of the specification language are introduced.

Note that the standard IEL reduction axioms do not have cases for iterated modalities. Indeed, such cases are not, at some level of abstraction, necessary since they can be dealt with logically “from inside out”. For any formulae $\varphi, \psi, \chi$ of IEL, $[\varphi?][\psi?]\chi$ can be dealt with in the following order, given the recursive structure of the reduction axioms:

$$[\varphi?][\psi?]\chi \leftrightarrow [\varphi?](\mathrm{trs}([\psi?]\chi)) \leftrightarrow \mathrm{trs}([\varphi?](\mathrm{trs}([\psi?]\chi)))$$

Here trs stands for the translation of the right side in the reduction axioms from Equation 1, as defined in Equation 4. While such a rule of thumb can be useful to some extent for human reasoning it is nevertheless not optimal for automatic reasoning. Even though we left out iterated modalities during the exposition of the logic we later deal with them as the implementation details require them. For this a direct recursion over the formulae would be optimal, and we approach this aspect in both of our implementations in Sections 5 and 6. Here we only briefly discuss some of the available modelling options.

One possible way to achieve this is by directly reducing iterated modalities to an equivalent non-iterated dynamic modality and then use the existing reduction axioms. For instance, for public announcement logic (PAL), which uses world-elimination instead of link-cutting, iteration of dynamic modalities boils down to the following equivalence for announcement composition:

$$[!\varphi][!\psi]\chi \leftrightarrow [!(\varphi \wedge [!\varphi]\psi)]\chi$$

However, there can be no such nor similar equivalent for IEL without sequences:

*No single question can induce a 4-equivalence-classes partition.*

All these complications are avoided by a language with questioning sequences:

$$[\varphi?][\psi?]\chi \leftrightarrow [\langle\varphi,\psi\rangle?]\chi$$

This can be achieved in a modular way by keeping the syntax and the semantics of the language unchanged for both the static part and the dynamic resolution part and replacing our initial questioning modality with a modality defined over sequences of questions. This is also a dynamic modality for the collective action of asking questions or raising issues but the syntax uses a list of formulae $\sigma = \langle\varphi_0, \dots, \varphi_n\rangle$:

$$[\sigma?]\,\varphi \quad \text{“after the questions in } \sigma \text{ are asked, } \varphi \text{ is the case”}.$$

The semantic definition of the dynamic modality is changed accordingly, the action's effect is to change an initial model $M$ into a new model $M_{\sigma?} = \langle W, \overset{x}{\approx}_?, \overset{a}{\sim}, V\rangle$ with:

$$\overset{x}{\approx}_? = \overset{x}{\approx}_{\langle\rangle?} \cap \bigcap_{n=0}^{|\sigma|-1} \overset{\varphi_n}{\equiv}_{M_n} \quad \text{for any agent } x.$$

For any model $M$ and questioning sequence $\sigma = \langle\varphi_0, \dots, \varphi_n\rangle$, the model $M_k$ denotes the model obtained after a questioning action using the sequence $\sigma_k = \langle\varphi_0, \dots, \varphi_k\rangle$ for $0 \leq k \leq n$. An empty questioning sequence does not change a model: $M_{\langle\rangle?} = M$ and, in particular, $\approx_{\langle\rangle?} = \approx$. This allows one to deal with longer sequences recursively using a head-tail pattern:

$$[\langle\rangle?]\varphi \leftrightarrow \varphi \text{ and } [\langle\varphi_0, \varphi_1, \dots, \varphi_n\rangle?]\varphi \leftrightarrow [\langle\varphi_0\rangle?][\langle\varphi_1, \dots, \varphi_n\rangle?]\varphi.$$

The case of iterating the resolution modality $[\,!\,]$ is much simpler because sequences of any length $[\langle !, !, \dots\rangle]$ can be collapsed to a resolution sequence of length one $[\langle !\rangle]$. This is so because the resolution modality is idempotent: $!;! = !$. Therefore, we only have to consider iteration between sequential asking modalities and single, i.e., depth one, resolution modalities.

This leads to the more general reduction axioms we introduce in Section 4. The reduction axioms in Equation 1 can be also seen as particular cases in which we take $n = 1$ inside the dynamic questioning modality $[\sigma(n)?]$. We lift the restriction to single questions from the vanilla version of the language and allow questioning sequences in two stages. First by introducing special reduction axioms for minimal sequences, i.e., sequences of length two, in Section 5. Second, we introduce questioning sequences of arbitrary length and give fully general reduction axioms for them in Sections 4 and 6. We continue our exposition using the simplest version of IEL and afterwards return in Section 4 to considering the SQL extension in more detail.

## 3 The Tableau Synthesis Framework Applied to IEL

In order to obtain a sound, complete and terminating tableau calculus for IEL we apply the tableau synthesis framework introduced in [11, 10]. In brief, the tableau synthesis method works as follows. The user defines the formal semantics of their logic in the first-order specification language of the framework. The semantic specification can then be automatically transformed into tableau rules that form a calculus which is sound and complete provided the semantic specification satisfies certain conditions. In a next step the possibility to refine the tableau calculus in two ways is explored. First, it may be possible to refine that tableau rules by reducing their branching factor and, second, it may be possible to internalise semantic constructs of the tableau language in the language of the logic. Finally, the unrestricted blocking mechanism can be added to the obtained calculus. The blocking mechanism ensures termination of the calculus if the logic has the finite model property. The final calculus is sound, complete and terminating, and, hence, provides the basis for a decision procedure for the logic.

The *object language* of specification of syntax of the logic IEL has several distinct sorts. The main sort of the language is the sort of formulae (sort $\mathsf{f}$) which are denoted as $\varphi, \psi, \ldots$. Other sorts are individuals (sort $\mathsf{i}$) denoted $i, j, \ldots$, propositional atoms (sort $\mathsf{p}$) denoted $p, q, \ldots$, and agent labels (sort $\mathsf{a}$) denoted as $a, a_0, a_1, \ldots$.

For reasons of economy and simplicity, we fix a (minimal) set of connectives for the syntax specification of IEL. The connectives and their types are listed in Figure 1.

Useful additions to the specification language include the *singleton set* operator $\{\cdot\}$ and the operator $\#\cdot$, which respectively link the individual and formula sorts, and the proposition and formula sorts. The operator $@.\cdot$ is the *at* (or *satisfaction*) operator, which is useful for internalising the semantic specification.

The *meta-language* of IEL for the specification of the semantics extends the object language of IEL with an additional domain sort $\mathsf{d}$ and the following symbols: binary predicate symbols (of type $(\mathsf{d}, \mathsf{d})$) $R_{\overset{a}{\approx}}$, $R_{\overset{a}{\approx} \cap \overset{a}{\sim}}$, and $R_{\overset{a}{\sim}}$; the equality symbol $\dot{\approx}$ (we use a dot to distinguish equality from the issue relation); domain variables $x, y, z, \ldots$; and the first-order quantifiers $\forall$ and $\exists$. Finally, the meta-language contains three interpretation symbols $\nu_{\mathsf{i}}$, $\nu_{\mathsf{f}}$, and $\nu_{\mathsf{p}}$. For every nominal $i$ of sort $\mathsf{i}$, $\nu_{\mathsf{i}}(i)$ is a term of sort $\mathsf{d}$. For every IEL-formula $\varphi$ of sort $\mathsf{f}$, proposition $p$ of sort $\mathsf{p}$, and term $t$ of sort $\mathsf{d}$, $\nu_{\mathsf{f}}(\varphi, t)$ and $\nu_{\mathsf{p}}(p, t)$ are atomic formulae in the semantic specification language for IEL.

Figure 2 shows the definition of the semantics of the IEL connectives in the meta-language. We give the standard Boolean and modal semantic definitions in the right column and the semantics of the sort-bridging connectives in the left column. Both are followed by definitions for the dynamic modalities. We denote the set of all these formulae by $S_0$.

Additionally there are conditions which specify properties of relations and equality. They are captured by the background theory axioms $S^b$ which are listed in Figures 3 and 4.

The described semantic specification captures in first-order sentences the semantic conditions for IEL from Section 2. In particular, the difference between the semantic definition of the static modalities and the dynamic modalities becomes more apparent. While the static modalities are dropped by the definitions, the dynamic ones are only dropped in the definitions for atomic components in their scope. However, for complex formulae the dynamic modality is applied in the right hand side of the definition to a formula with lower complexity. This is also reflected in the semantic specifications that the reduction axioms vary depending on whether they are for propositional atoms, for singletons, and for formulae.

The next step is to transform the semantic specification into a normalised implicational form [11]. This is done by decomposing each logical equivalence of the specification $S_0$ into the left-to-right implication and the contrapositive of the right-to-left implication. The resulting sets of formulae are denoted by $S^+$ and $S^-$.

| Connective | Type | Connective | Type | Connective | Type |
|---|---|---|---|---|---|
| $\{\cdot\}$ | $\mathsf{i} \mapsto \mathsf{f}$ | $\#\cdot$ | $\mathsf{p} \mapsto \mathsf{f}$ | | |
| $@.\cdot$ | $(\mathsf{i}, \mathsf{f}) \mapsto \mathsf{f}$ | $Q.\cdot$ | $(\mathsf{a}, \mathsf{f}) \mapsto \mathsf{f}$ | $[\cdot?].\cdot$ | $(\mathsf{f}, \mathsf{a}, \mathsf{f}) \mapsto \mathsf{f}$ |
| $\neg\cdot$ | $\mathsf{f} \mapsto \mathsf{f}$ | $K.\cdot$ | $(\mathsf{a}, \mathsf{f}) \mapsto \mathsf{f}$ | $[!].\cdot$ | $(\mathsf{a}, \mathsf{f}) \mapsto \mathsf{f}$ |
| $\cdot \wedge \cdot$ | $(\mathsf{f}, \mathsf{f}) \mapsto \mathsf{f}$ | $X.\cdot$ | $(\mathsf{a}, \mathsf{f}) \mapsto \mathsf{f}$ | | |

Figure 1: Connectives of the object language of IEL.

$$\begin{array}{cc}
\forall x(\nu_{\mathsf{f}}(\{i\},x) \leftrightarrow x \dot{\approx} \nu_{\mathsf{i}}(i)) & \forall x(\nu_{\mathsf{f}}(\neg\varphi,x) \leftrightarrow \neg\nu_{\mathsf{f}}(\varphi,x)) \\
\forall x(\nu_{\mathsf{f}}(\#p,x) \leftrightarrow \nu_{\mathsf{p}}(p,x) & \forall x(\nu_{\mathsf{f}}(\varphi\wedge\psi,x) \leftrightarrow \nu_{\mathsf{f}}(\varphi,x)\wedge\nu_{\mathsf{f}}(\psi,x))) \\
\forall x(\nu_{\mathsf{f}}(@_i\varphi,x) \leftrightarrow \nu_{\mathsf{f}}(\varphi,\nu_{\mathsf{i}}(i))) & \forall x(\nu_{\mathsf{f}}(Q_a\varphi,x) \leftrightarrow \forall y(R_{\overset{a}{\approx}}(x,y)\rightarrow\nu_{\mathsf{f}}(\varphi,y))) \\
\forall x(\nu_{\mathsf{f}}([\varphi?]_a\#p,x) \leftrightarrow \nu_{\mathsf{f}}(\#p,x))) & \forall x(\nu_{\mathsf{f}}(K_a\varphi,x) \leftrightarrow \forall y(R_{\overset{a}{\sim}}(x,y)\rightarrow\nu_{\mathsf{f}}(\varphi,y))) \\
\forall x(\nu_{\mathsf{f}}([q]_a\{i\},x) \leftrightarrow \nu_{\mathsf{f}}(\{i\},x))) & \forall x(\nu_{\mathsf{f}}(X_a\varphi,x) \leftrightarrow \forall y(R_{\overset{a}{\approx}\cap\overset{a}{\sim}}(x,y)\rightarrow\nu_{\mathsf{f}}(\varphi,y)))) \\
\forall x(\nu_{\mathsf{f}}([q]_a\neg\psi,x) \leftrightarrow \nu_{\mathsf{f}}(\neg[q]_a\psi,x))) & \forall x(\nu_{\mathsf{f}}([q]_a(\psi\wedge\chi),x) \leftrightarrow \nu_{\mathsf{f}}([q]_a\psi,x)\wedge\nu_{\mathsf{f}}([q]_a\chi,x)) \\
\forall x(\nu_{\mathsf{f}}([\varphi?]_aK_a\psi,x) \leftrightarrow \nu_{\mathsf{f}}(K_a[\varphi?]\psi,x)) & \forall x(\nu_{\mathsf{f}}([\,!\,]_aQ_a\psi,x) \leftrightarrow \nu_{\mathsf{f}}(Q_a[\,!\,]_a\psi,x)) \\
\forall x(\nu_{\mathsf{f}}([\,!\,]_aK_a\psi,x) \leftrightarrow \nu_{\mathsf{f}}(X_a[\,!\,]_a\psi,x)) & \forall x(\nu_{\mathsf{f}}([\,!\,]_aX_a\psi,x) \leftrightarrow \nu_{\mathsf{f}}(X_a[\,!\,]_a\psi,x)) \\
\multicolumn{2}{c}{\forall x(\nu_{\mathsf{f}}([\varphi?]_aQ_a\psi,x) \leftrightarrow (\nu_{\mathsf{f}}(\varphi\wedge Q_a(\neg\varphi\vee[\varphi?]_a\psi),x)\vee\nu_{\mathsf{f}}(\neg\varphi\wedge Q_a(\varphi\vee[\varphi?]_a\psi),x))} \\
\multicolumn{2}{c}{\forall x(\nu_{\mathsf{f}}([\varphi?]_aX_a\psi,x) \leftrightarrow (\nu_{\mathsf{f}}(\varphi\wedge X_a(\neg\varphi\vee[\varphi?]_a\psi),x)\vee\nu_{\mathsf{f}}(\neg\varphi\wedge X_a(\varphi\vee[\varphi?]_a\psi),x))}
\end{array}$$

Figure 2: Semantic specification $S_0$ of connectives for IEL ($q \in [\varphi?], [\,!\,]$)

$$\begin{array}{l}
\forall x\forall y(R_{\overset{a}{\approx}\cap\overset{a}{\sim}}(x,y) \leftrightarrow R_{\overset{a}{\approx}}(x,y)\wedge R_{\overset{a}{\sim}}(x,y)), \\
\forall x\forall y\forall z((R_{\overset{a}{\approx}}(x,y)\wedge R_{\overset{a}{\approx}}(y,z))\rightarrow R_{\overset{a}{\approx}}(x,z)), \quad \forall x\forall y\forall z((R_{\overset{a}{\sim}}(x,y)\wedge R_{\overset{a}{\sim}}(y,z))\rightarrow R_{\overset{a}{\sim}}(x,z)), \\
\forall x\forall y\forall z((R_{\overset{a}{\approx}\cap\overset{a}{\sim}}(x,y)\wedge R_{\overset{a}{\approx}\cap\overset{a}{\sim}}(y,z))\rightarrow R_{\overset{a}{\approx}\cap\overset{a}{\sim}}(x,z)), \\
\forall x\forall y(R_{\overset{a}{\approx}}(x,y)\rightarrow R_{\overset{a}{\approx}}(y,x)), \quad \forall x\forall y(R_{\overset{a}{\approx}\cap\overset{a}{\sim}}(x,y)\rightarrow R_{\overset{a}{\approx}\cap\overset{a}{\sim}}(y,x)), \\
\forall x\forall y(R_{\overset{a}{\sim}}(x,y)\rightarrow R_{\overset{a}{\sim}}(y,x)), \quad \forall xR_{\overset{a}{\approx}}(x,x), \quad \forall xR_{\overset{a}{\approx}\cap\overset{a}{\sim}}(x,x), \forall xR_{\overset{a}{\sim}}(x,x)
\end{array}$$

Figure 3: Semantic specification of background theory axioms for the relations

It is not difficult to check that the semantic specification is well-defined in the sense of [11]. A semantic specification $S$ is well defined iff $S$ is normalized and the following conditions hold:

(wd1) $\forall S^0, \forall S^b \models \forall S$,

(wd2) the relation $\prec$ induced by $S$ is a well-founded ordering on formulae, and

(wd3) for every formula $\varphi = \sigma(\varphi_1,\ldots,\varphi_m)$, defining a connective $\sigma$:

$$\forall S^0, \forall S^b \restriction \mathsf{sub}_{\prec}(\varphi) \models_c \forall\overline{x}\Big(\big(\bigwedge\Phi_+^{\varphi} \rightarrow \phi^{\sigma}(\varphi_1,\ldots,\varphi_m,\overline{x})\big)\wedge\big(\phi^{\sigma}(\varphi_1,\ldots,\varphi_m,\overline{x})\rightarrow\bigvee\Phi_-^{\varphi}\big)\Big)$$

Here $\Phi_+^{\varphi}$ is the set obtained by collecting all instantiations of consequents from $S^+$ (the positive specifications in $S$) matching the formula $\varphi$. $\Phi_-^{\varphi}$ is the set obtained by collecting all instantiations of antecedents from $S^-$ (the negative specifications in $S$) matching formula $\varphi$.

Condition (wd1) expresses the decomposition of the specification $S$ to $S^0$ and $S^b$ after normalization. The set $S^0$ contains the connective definitions, and the set $S^b$ contains the background theory conditions. The set $S^0$ is further decomposed in two disjoint sets $S^+$ and $S^-$. Since the semantic specification is the union of the connective definitions and the background

$$\begin{array}{l}
\forall x(x\dot{\approx}x), \quad \forall x\forall y(x\dot{\approx}y\rightarrow y\dot{\approx}x), \quad \forall x\forall y\forall z(x\dot{\approx}y\wedge y\dot{\approx}z\rightarrow x\dot{\approx}z), \\
\forall\bar{p}\forall\bar{x}\forall y_i(x_i\dot{\approx}y_i\rightarrow f(\bar{p},\bar{x})\dot{\approx}f(\bar{p},x_1,\ldots,x_{i-1},y_i,x_{i+1},\ldots,x_n)), \\
\forall p\forall x\forall y(\nu_{\mathsf{f}}(\#p,x)\wedge x\dot{\approx}y\rightarrow\nu_{\mathsf{f}}(\#p,y)), \quad \forall p\forall x\forall y(\nu_{\mathsf{p}}(p,x)\wedge x\dot{\approx}y\rightarrow\nu_{\mathsf{p}}(p,y)).
\end{array}$$

Figure 4: Semantic specification of equality axioms

Tableau Expansion Rules (generated from $S_0 = S^+ \cup S^-$):

$$\frac{@_l[\varphi?]\Box_a\psi}{@_l\varphi, @_l\Box_a(\neg\varphi\vee[\varphi?]\psi) \mid @_l\neg\varphi, @_l\Box_a(\varphi\vee[\varphi?]\psi)}\ (\Box\in Q,X),\qquad \frac{@_l[\,!\,]K_a\psi}{@_lX_a[\,!\,]\psi}, \tag{3a}$$

$$\frac{@_l\neg[\varphi?]\Box_a\psi}{@_l(\neg\varphi\vee\neg\Box_a(\neg\varphi\vee[\varphi?]\psi)),\ @_l(\varphi\vee\neg\Box_a(\varphi\vee[\varphi?]\psi))}\ (\Box\in Q,X),\qquad \frac{@_l\neg[\,!\,]K_a\psi}{@_l\neg X_a[\,!\,]\psi}. \tag{3b}$$

$$\frac{@_l\blacksquare b}{@_lb},\quad \frac{@_l\blacksquare\neg\varphi}{@_l\neg\blacksquare\varphi},\quad \frac{@_l\blacksquare(\varphi\wedge\psi)}{@_l\blacksquare\varphi,\ @_l\blacksquare\psi},\quad \frac{@_l[\varphi?]K_a\psi}{@_lK_a[\varphi?]\psi},\quad \frac{@_l[\,!\,]\Box_a\varphi}{@_l\Box_a[\,!\,]\varphi}\ (\Box\in Q,X), \tag{3c}$$

$$\frac{@_l\neg\blacksquare b}{@_l\neg b},\quad \frac{@_l\neg\blacksquare\neg\varphi}{@_l\blacksquare\varphi},\quad \frac{@_l\neg\blacksquare(\varphi\wedge\psi)}{@_l\neg\blacksquare\varphi \mid @_l\neg\blacksquare\psi},\quad \frac{@_l\neg[\varphi?]K_a\psi}{@_l\neg K_a[\varphi?]\psi},\quad \frac{@_l\neg[\,!\,]\Box_a\varphi}{@_l\neg\Box_a[\,!\,]\varphi}\ (\Box\in Q,X), \tag{3d}$$

$$\frac{@_l\neg\Box_a\varphi}{@_l\Diamond_a\{f_{\neg\Box}(l,a,\varphi)\}, @_{f_{\neg\Box}(l,a,\varphi)}\neg\varphi}\ (\Box\in Q,K,X),\qquad \frac{@_l\Box_a\varphi, @_l\Diamond_a\{l_2\}}{@_{l_2}\varphi}\ (\Box\in Q,K,X), \tag{3e}$$

$$\frac{@_l\neg\neg\varphi}{@_l\varphi},\quad \frac{@_l\varphi\wedge\psi}{@_l\varphi,\ @_l\psi},\quad \frac{@_l\neg(\varphi\wedge\psi)}{@_l\neg\varphi \mid @_l\neg\psi}, \tag{3f}$$

Background Theory Rules (generated from $S_b$):

$$\frac{@_l\widehat{X}_a\{l_2\}}{@_l\widehat{Q}_a\{l_2\},\ @_l\widehat{K}_a\{l_2\}},\quad \frac{@_l\widehat{Q}_a\{l_2\},\ @_l\widehat{K}_a\{l_2\}}{@_l\widehat{X}_a\{l_2\}}, \tag{3g}$$

$$\frac{@_l\Diamond_a\{l_2\}, @_{l_2}\{l_3\}}{@_l\Diamond_a\{l_3\}},\quad \frac{@_l\{l\}}{@_l\Diamond_a\{l\}},\quad \frac{@_l\Diamond_a\{l_2\}}{@_{l_2}\Diamond_a\{l\}},\quad \frac{@_l\Diamond_a\{l_2\},\ @_{l_2}\Diamond_a\{l_3\}}{@_l\Diamond_a\{l_3\}}, \tag{3h}$$

$$\frac{@_l\Diamond_a\{l_2\}}{@_{l_2}\{l_2\}},\ \frac{@_l\{l_2\}}{@_{l_2}\{l\}},\ \frac{@_l\neg\{l_2\}}{@_{l_2}\neg\{l\}},\quad \frac{@_l\varphi}{@_l\{l\}},\ \frac{@_l\varphi, @_l\{l_2\}}{@_{l_2}\varphi},\quad \text{(Clash): } \frac{@_l\varphi,\ @_l\neg\varphi}{\bot}. \tag{3i}$$

Figure 5: Refined calculus for IEL, where $\blacksquare \in \{[\varphi?]_a, [\,!\,]_a\}$, $b \in \{\#p, \{n\}\}$, $\Diamond \in \{\widehat{Q}, \widehat{K}, \widehat{X}\}$

theory, conditions (wd1) and (wd3) are trivially satisfied. Showing condition (wd2), i.e., well-foundedness of the ordering $\prec$ induced by the normalised specification, is more involved than usual because of the statements capturing the reduction axioms. Well-foundedness of the order can be established by assigning IEL formulae the following complexity measure:

$$c(p) = 1,\quad c(!) = 1,\quad c(\neg\varphi) = 1 + c(\varphi),\quad c(\varphi\wedge\psi) = 1 + \max(c(\varphi), c(\psi)), \tag{2a}$$

$$c(\Box_a\varphi) = 1 + c(\varphi)\quad \text{for } \Box_a \in \{K_a, Q_a, X_a\}, \text{ and} \tag{2b}$$

$$c([q]\psi) = (c(q) + 5)\cdot c(\psi)\quad \text{for } q \in \{\varphi?, !\}. \tag{2c}$$

Turning the normalised semantic specification into tableau rules in accordance with [11] then produces a sound and complete tableau calculus for checking satisfiability for IEL. We do not present this calculus here, but present (in Figure 5) immediately the calculus obtained after refinement.

Two refinements described in [11] have been applied. The first refinement is the internalisation of the domain symbols including interpretation symbols $\nu_{\mathsf{i}}$, $\nu_{\mathsf{f}}$, and $\nu_{\mathsf{p}}$ in the language of the logic. For example, the rules generated for the $\Box$ operators are ($\Box \in \{Q, K, X\}$):

$$\frac{\nu_{\mathsf{f}}(\neg\Box_a\varphi, l)}{R(l, f_{\neg\Box}(l,a,\varphi)),\ \nu_{\mathsf{f}}(\neg\varphi, f_{\neg\Box}(l,a,\varphi))}\quad \text{and}\quad \frac{\nu_{\mathsf{f}}(\Box_a\varphi, l),\ l_2 \mathrel{\dot\approx} l_2}{\neg R(l, l_2) \mid \nu_{\mathsf{f}}(\varphi, l_2)}$$

$R$ denotes the appropriate accessibility relation associated with $\Box_a$. $f_{\neg\Box}$ represents one of three fixed Skolem functions used as a convenient way to create witnesses for formulae of existential extent. Because IEL is a hybrid logic it fully supports individuals and the rules can be rewritten as

$$\frac{@_l\neg\Box_a\varphi}{@_l\Diamond_a\{f_{\neg\Box}(l,a,\varphi)\},\ @_{f_{\neg\Box}(l,a,\varphi)}\neg\varphi} \quad\text{and}\quad \frac{@_l\Box_a\varphi,\ @_{l_a}\Diamond_a\{l_2\}}{@_l\neg\Diamond_a\{l_2\}\mid @_{l_2}\varphi}.$$

Similarly for the other rules.

The second refinement attempts to replace branching rules by rules with fewer or no branches. For example, the rule for positive occurrences of $\Box$,

$$\frac{@_l\Box_a\varphi,\ @_{l_a}\Diamond_a\{l_2\}}{@_l\neg\Diamond_a\{l_2\}\mid @_{l_2}\varphi}, \quad\text{is refined to}\quad \frac{@_l\Box_a\varphi,\ @_l\Diamond_a\{l_2\}}{@_{l_2}\varphi}.$$

Other refined rules in the presented calculus are the rules expressing triangular properties in (3g) and (3h). These rule refinements are justified because the (†) condition from [11] can be shown to hold in each case. The presented calculus is therefore sound and complete for IEL.

Finally, if the logic has the finite model property then the generated tableau calculus can be turned into a decision procedure by adding the blocking mechanism introduced in [9] which is based on the following unrestricted blocking rule:

$$(\mathrm{UB}){:}\frac{@_l\{l\},\ @_{l_0}\{l_0\}}{@_l\{l_0\}\mid @_l\neg\{l_0\}}$$

For the static part of IEL the finite model property is obtained by a standard filtration argument. The reduction axioms introduced before provide a way to translate formulae from the dynamic part to equivalent formulae in the static language. The translation is:

$$t(p)=p,\quad t(\neg\varphi)=\neg t(\varphi),\quad t(\varphi\wedge\psi)=t(\varphi)\wedge t(\psi), \tag{4a}$$
$$t(\Box_a\varphi)=\Box_a t(\varphi)\quad\text{for } \Box_a\in\{K_a,Q_a,X_a\}, \tag{4b}$$
$$t(lhs)=t(rhs)\quad\text{for the reduction axioms in (1a)–(1d).} \tag{4c}$$

This implies that IEL has the finite model property and we can obtain the following result:

**Theorem 1.** *The calculus listed on Figure 5 is sound and complete for* IEL *satisfiability and it is also terminating if equipped with the unrestricted blocking mechanism.*

## 4 Extension to Sequential Questioning Logic

In this section we generalize the IEL/DELQ framework to a setting using questioning sequences and we call the emerging theory *Sequential Questioning Logic* (henceforth, *SQL*).

The language of SQL is recursively defined by the following BNF:

$$\varphi \quad ::= \quad n \mid p \mid \neg\varphi \mid \varphi\vee\varphi \mid \Box\varphi \mid [\sigma(k)?]\varphi \mid [\,!\,]\varphi$$

with $n, p, a, \neg, \vee, \Box, [\,!\,]$ as before and $\sigma(n)?$ representing dynamic questioning actions where $\sigma(n)=\langle\varphi_0,\ldots,\varphi_{n-1}\rangle$ is a sequence of SQL formulae. The semantics of the operators is also as before with the generalized intersection already introduced in Section 2 used for sequences. The fact that the questioning modalities are the only ones different between IEL and SQL dialects allows us to add questioning sequences while preserving all the other components.

Figure 6: SQL specific dynamic connectives, semantics and tableau rules

| conn. | type | semantics | rules |
|---|---|---|---|
| | | $\forall x(\nu_{\mathsf{f}}([\sigma?]\#p, x) \leftrightarrow \nu_{\mathsf{f}}(\#p, x)))$ | |
| | | $\forall x(\nu_{\mathsf{f}}([\sigma?]\{n\}, x) \leftrightarrow \nu_{\mathsf{f}}(\{n\}, x)))$ | |
| | | $\forall x(\nu_{\mathsf{f}}([\sigma?]\neg\varphi, x) \leftrightarrow \nu_{\mathsf{f}}(\neg[\sigma?]\varphi, x)))$ | As in Figure 5 with $[\sigma?]$ instead of $[\varphi?]$ |
| $[\cdot?]$ | $[\mathsf{f}] \mapsto \mathsf{f}$ | $\forall x(\nu_{\mathsf{f}}([\sigma?](\varphi \wedge \psi), x) \leftrightarrow \nu_{\mathsf{f}}([\sigma?]\varphi, x) \wedge \nu_{\mathsf{f}}([\sigma?]\psi, x))$ | |
| | | $\forall x(\nu_{\mathsf{f}}([\sigma?]K_a\varphi, x) \leftrightarrow \nu_{\mathsf{f}}(K_a[\sigma?]\varphi, x))$ | |
| | | $\forall x(\nu_{\mathsf{f}}([\sigma(n)?]Q_a\varphi, x) \leftrightarrow$ see below | See the generalized rules below |
| | | $\forall x(\nu_{\mathsf{f}}([\sigma(n)?]X_a\varphi, x) \leftrightarrow$ see below | |

The static part of SQL brings nothing new, it is axiomatized, as before in IEL, by standard hybrid logic axioms for nominals and intersection. The dynamic part of SQL brings some new features that generalize the initial setting from IEL, and we do not require formulae in $\sigma(n)$ to induce a partition of the domain.

The significant new feature in SQL relative to IEL is the presence of dynamic questioning modalities over sequences of questions. This has to bring about new reduction axioms. For formulae $\varphi$ with factual content the jump to questioning sequences is an obvious generalization of the pattern in previous reduction axioms. For such $\varphi_0, \varphi_1$ in a minimal sequence we have:

$$\begin{aligned}
[\varphi_0, \varphi_1]X\varphi \leftrightarrow\ & (\varphi_0 \wedge \varphi_1 \wedge X((\varphi_0 \wedge \varphi_1) \to [\varphi_0, \varphi_1]\varphi)) \\
\vee\ & (\varphi_0 \wedge \neg\varphi_1 \wedge X((\varphi_0 \wedge \neg\varphi_1) \to [\varphi_0, \varphi_1]\varphi)) \\
\vee\ & (\neg\varphi_0 \wedge \varphi_1 \wedge X((\neg\varphi_0 \wedge \varphi_1) \to [\varphi_0, \varphi_1]\varphi)) \\
\vee\ & (\neg\varphi_0 \wedge \neg\varphi_1 \wedge X((\neg\varphi_0 \wedge \neg\varphi_1) \to [\varphi_0, \varphi_1]\varphi)).
\end{aligned}$$

This generalizes in the expected way to longer factual sequences. However, this cannot be extended beyond factual formulae, not even for minimal questioning sequences of length two. This approach fails for complex formulae that have questioning content or, in general, extra-factual or higher-order content. Consider as an illustration the following complex questioning formula: $\xi := (\widehat{Q}i \to (j \vee k)) \wedge ((\widehat{Q}j \wedge p) \to \widehat{Q}i)$. A model with a domain of three worlds $i, j, k$, universal issue and epistemic relations, and a valuation that makes $p$ true at $k$ provides a counterexample when we make the following substitutions: $\varphi_0 \mapsto \xi$, $\varphi_1 \mapsto \xi$, $\varphi \mapsto \neg p$.

For questioning sequences the disjunctive structure of the reduction axiom has to induce a partition of the domain. However, the questioning sequence does not have to give rise to a partition. This difference is not always fully understood and appreciated. If the questioning sequence has a more complex structure, for instance, if it induces a cover of the domain, more complex patterns are needed in the reduction axiom, that can ensure that the right hand side remains an exhaustive exclusive disjunction. In this way, fully general reduction axioms for SQL can be obtained and they will have the pattern given below.

We present the new additions in a synthetic way in Table 6. The questioning modalities have a different type than before as they are now defined over lists of formulae. Except for this type difference most of the tableau rules will be as before. The new connectives using sequences will have new reduction axioms and new rules as specified in the table.

The reduction axioms will have the following pattern, for $\blacksquare \in \{Q, X\}$:

$$[\sigma(n)]\blacksquare_a\varphi \leftrightarrow \bigvee_{i=0}^{2^{|\sigma(n)|}} \big( \bigwedge_{k=0}^{|\sigma(n)|} ([\sigma(k-1)]\varphi_k)^{\beta(i)(k)} \wedge \blacksquare_a( \bigwedge_{k=0}^{|\sigma(n)|} ([\sigma(k-1)]\varphi_k)^{\beta(i)(k)} \rightarrow [\sigma(n)]\varphi)\big),$$

where $|\sigma(n)|$ is the length of the questioning sequence $\sigma(n) = \langle\varphi_0, .., \varphi_{n-1}\rangle$,

and the value of $\varphi^{\beta(k)(i)}$ is determined by: $\varphi^{\beta(k)(i)} = \begin{cases} \varphi & \text{if } \beta(k)(i) = 1, \text{ and} \\ \neg\varphi & \text{if } \beta(k)(i) = 0. \end{cases}$

$\beta(k)(i)$ represents the $i$-th position in the binary encoding $\beta(k)$ of the decimal number $k$.

The corresponding tableau rules are obtained as follows, for $\chi_i^k = \bigwedge_{k=0}^{|\sigma(n)|}([\sigma(k-1)]\varphi_k)^{\beta(i)(k)}$:

$$\frac{@_l[\sigma?]\blacksquare_a\varphi}{@_l \bigwedge_{k=0}^{|\sigma(n)|}\chi_1^k \wedge \blacksquare_a(\bigwedge_{k=0}^{|\sigma(n)|}\chi_1^k \rightarrow [\sigma(n)]\varphi) \mid \cdots \mid @_l \bigwedge_{k=0}^{|\sigma(n)|}\chi_n^k \wedge \blacksquare_a(\bigwedge_{k=0}^{|\sigma(n)|}\chi_n^k \rightarrow [\sigma(n)]\varphi)}$$

In addition the complexity function for formulae has to add to Equation 2 values that take into account the length of the questioning sequences.

## 5 Implementing an IEL Prover with MetTeL$^2$

In this section, we describe our experience in using MetTeL$^2$ [13] to generate a tableau prover for the tableau calculus derived in the previous section. MetTeL$^2$ is a prototypical tableau prover generator developed with the tableau synthesis framework as its theoretical foundation. Given the specification of a logic and the specification of a tableau calculus for this logic MetTeL$^2$ generates Java code for a tableau prover implementing the tableau calculus. MetTeL$^2$ has been successfully applied to several of logics, including Boolean logic, modal logics K, KT, S4, description logics $\mathcal{ALCO}$ and $\mathcal{ALBO}$id, and a hybrid logic with global counting operators [6]. The list is constantly growing. These test cases and downloadable copies of the generated provers are publicly available from the MetTeL website.

The underlying language for syntax specification of IEL in MetTeL$^2$ is in line with the tableau synthesis framework object specification language. As the object language is settled in Section 3, preparing the syntax specification for MetTeL$^2$ is straightforward. For example, the syntax specification contains declaration of four sorts.

```
sort formula, agent, prop, individual;
```

Further, declarations of each connectives follow their representation in Figure 1. For instance, the connective # is specified by the following declaration.

```
formula proposition = '#' prop;
```

The declaration of the dynamic modality for questioning is given by:

```
formula query = '[?' formula ']' agent formula;
```

The syntax for Skolem terms which are fresh labels introduced during the application of diamond rules is given as follows.

```
individual fq = 'fq' '(' individual ',' agent ',' formula ')';
individual fk = 'fk' '(' individual ',' agent ',' formula ')';
individual fx = 'fx' '(' individual ',' agent ',' formula ')';
```

The specification of the tableau calculus in MetTeL$^2$ reflects all the rules of Figure 5. There are decomposition rules for positive as well as negative occurrences of all connectives. The exception is negation, which only has a rule for negative occurrence, namely elimination of double negation. For example, the rules for the three static modalities are specified as follows.

```
@l<q> A P / @l <q> A {fq(l,A,P)} @fq(l,A,P)P priority 7$;
@l<k> A P / @l <k> A {fk(l,A,P)} @fk(l,A,P)P priority 7$;
@l<x> A P / @l <x> A {fx(l,A,P)} @fx(l,A,P)P priority 7$;
@l ~(<q> A P) @l <q> A {l2} / @l2~P priority 2$;
@l ~(<k> A P) @l <k> A {l2} / @l2~P priority 2$;
@l ~(<x> A P) @l <x> A {l2} / @l2~P priority 2$;
```

The rules for dynamic modalities have specific cases for atomic formulae, i.e., propositional atoms or nominals, and cases for complex formulae with non-factual content: questioning, epistemic or both. The cases for atomic formulae are specified as follows.

```
@l ([?P] A #B) /  @l #B priority 2$;             @l ~([?P] A #B) /  @l ~(#B) priority 2$;
@l ([?P] A {l2})  /  @l ({l2})  priority  2$;    @l ~([?P] A {l2})  /  @l ~{l2}  priority  2$;
@l ([!]   A {l2})  /  @l {l2}  priority  2$;     @l ~([!]   A {l2})  /  @l ~{l2}  priority  2$;
@l ([!]   A #B) /  @l #B priority 2$;            @l ~([!]   A #B) /  @l ~(#B) priority 2$;
```

Having the rules for intersection of accessibility relations in the background theory is important because it plays a crucial role in the reduction rule for the dynamic modalities:

```
@l<q> A {l2} @l<k> A {l2} / @l<x> A {l2} priority 2$;
@l<x> A {l2} / @l<q> A {l2} @l<k> A {l2} priority 2$;
```

In MetTeL$^2$, appearance of an equality formula in a branch immediately triggers ordered rewriting within the branch. The unrestricted blocking rule is implemented with use of this feature and the equality formula on its left branch forces the branch to be rewritten with respect to the additional equality.

An important feature provided by the MetTeL$^2$ implementation is the possibility to assign priorities to the tableau rules. This can be used to control the way the rules are applied in the generated prover. Each rule is followed by a number, which defines the rule application priority. Rules with smaller priority values have higher priority and applied more eagerly. Use of this feature is essential for the efficiency of the generated provers and especially in the case of IEL because the rules corresponding to reduction axioms with disjunctive patterns can be assigned lower priorities (higher priority values), thus reducing the branching factor and improving efficiency.

From the syntax specification and the tableau calculus, a tableau prover for IEL is automatically generated by MetTeL$^2$ according to the process described in detail in [13]. The generated prover can be used like most other tableau provers. Given a set of formulae as an input, the prover returns an answer **`Satisfiable`** or **`Unsatisfiable`** together with a model in the first case or with a set of contradictory formulae in the latter case.

We have tested the generated prover on a small set of sample formulae, where all the answers were correct.

## 6 Implementing IEL and SQL in Haskell

The second implementation uses the literate Haskell script Qtab.lhs. In this section, we provide a brief description of this implementation and include the implementation itself in the long version of the paper [8]. The implementation started from a pre-existing tableau prover for hybrid logic [16], to which we have added the details needed to model dynamic questioning actions. The tableau construction functionality was also completely changed. The current setting is more congenial with the framework from [11], which includes having a background theory for intersection and using unrestricted blocking.

We give next a broad view of the modules contained in the Qtab.lhs architecture and their functionality. Syntax.hs: Preliminary module containing the data structures for modal and first-order logic formulae as well as the tableaux. Qtab.lhs: The module containing the main functionality for the tableau prover such as the decide function that takes a formula in the IEL language and decides if it is or is not a tautology. Also functions controlling the order in which the formulae are analysed. Decomp.hs: The module containing the functionality associated with tableau expansion rules by logical decomposition. This proceeds either by standard logical analysis or by rules synthesized from reduction axioms. Backgrd.hs: The module containing the main components of the IEL background theory. In particular, the rules governing the behaviour of nominals and the rules for intersection are defined here. Also the unrestricted blocking mechanism is handled by functions in this module. Divide.hs: The order in which branches in a tableau are expanded is determined by their syntactic structure. The module contains functions used to recognize structural properties of formulae and to divide tableau nodes into component lists of prioritised formulae. Auxilar.hs: The module containing auxiliary functionality (such as displaying tableaux and translating formulae).

One particular aspect in which the Haskell implementation proved to be useful was in dealing with questioning sequences. Questioning sequences can also be added in MetTeL2, see the rules for sequences of length two in the appendix of the long version. The features of functional programming and the way in which recursion is implicitly built in Haskell definitions makes working with arbitrary sequences of questions easier. It also made it obvious that modalities capturing questioning sequences can be modelled as fully functional algebraic data structures suitable for recursive manipulation. The decomposition rules for the static connectives and resolution are as in Figure 5. We include several illustrations of Qtab.lhs output for questioning sequences of length two in the long version. The decomposition rules for the general case of arbitrarily long sequences follow the pattern of the rules from Table 6. We include below some illustrative examples of prover output for some paradigmatic examples of SQL formulae:

```
*Sql> deciden (Quest [Prop (P 0), Prop (Q 0)] (Box 2 (Disj [Prop (P 0), Prop (Q 0)])))
(False,5)
*Sql> deciden (Quest [Prop (P 0), Prop (Q 0)] (Box 1 (Disj [Prop (P 0), Prop (Q 0)])))
(False,14)
*Sql> deciden (Quest [Prop (P 0)] (Reso  (Box 2 (Prop (P 0)))))
(False,12)
```

The first example illustrates a questioning sequence of length two combined with the static knowledge modality, which has a commutating behaviour. The second example illustrates a questioning sequence of length two in combination with the static issue modality, which uses reduction axioms based on the disjunction pattern: The third example illustrates a questioning sequence combining both asking actions and resolution actions. Because the resolution modality is idempotent, all resolution sequences are equivalent to a sequence of length one. Therefore the following reduction axioms are used for dealing with the aspect of the interaction between a questioning-sequence followed by a resolution-sequence:

$$[\sigma?][\,!\,]Q_a\psi \leftrightarrow [\sigma?]Q_a[\,!\,]\psi, \quad [\sigma?][\,!\,]K_a\psi \leftrightarrow [\sigma?]X_a[\,!\,]\psi, \quad [\sigma?][\,!\,]X_a\psi \leftrightarrow [\sigma?]X_a[\,!\,]\psi \tag{5a}$$

The final illustration shows the difference between knowing that and knowing whether. After a yes/no question about $p$ the issue relation decides whether $p$, this turns out to be a SQL validity, however, it does not settle that $p$ holds.

```
*Sql> (deciden (Neg (Quest [Prop (P 0), Neg (Prop (P 0))] (Box 1 (Prop (P 0))))))
(False,40)

*Sql> (deciden (Neg (Quest [Prop (P 0), Neg (Prop (P 0))]
      (Disj [Box 1 (Prop (P 0)), Box 1 (Neg (Prop (P 0)))]))))
(True,84)
```

More detailed code output, traces of step-by step inference and tableau generation, and further explanation of the code functionality is included in the long version of the paper.

## 7 Concluding Remarks

In this paper we have shown what can be achieved when applying tableau synthesis and implementation for dynamic modalities of the simplest kind. This is only an initial illustration that serves as a case study for further extensions. We have considered one such extension to questioning sequences. Further extensions that we want to consider in the future include dynamic questioning actions that can model privacy and insecure communication and employ product update [2, 1] for computing issue relations [7] and a richer repertoire of questioning actions that go beyond the propositional case and include wh-questions [5, 18].

MetTeL$^2$ provides a robust and efficient platform for automatically generating a tableau prover for IEL. On the small set of formulae we used for testing, MetTeL$^2$ was faster than the `Haskell` prover, because it implements clever backtracking techniques and other optimisations, currently not supported in the `Haskell` implementation.

On the other hand, the `Haskell` implementation provides a framework in which more experimental features of further extensions can be easily programmed and tested before they are ready to become mainstream conditions. We used the case of SQL to illustrate such an extension. We conclude with two main points about the overall significance of our approach.

We have shown how a dynamic component, in particular, dynamic questioning actions, can be integrated in the tableau synthesis framework. Based on the synthesised tableau calculus, two implementations have been developed: MetTeL$^2$ and `Qtab.lhs`. This dynamic extension relies on rules in which the complexity of the formulae inside the scope of the dynamic modalities is reduced, even if the complexity of the conclusion formula in the rule can increase.

The second contribution facilitated by the implementations is an extension of the underlying dynamic logic itself. Implementing the reduction details made it obvious that a logical language containing sequences of questions, not just modalities for questioning actions, can be modelled in the framework, and extends the dynamic logic in a useful direction.

## References

[1] A. Baltag. Logics for insecure communication. In *Proceedings of the 8th Conference on Theoretical Aspects of Rationality and Knowledge*, pages 111–121. Morgan Kaufmann, 2001.

[2] A. Baltag, L. S. Moss, and S. Solecki. The logic of public announcements, common knowledge, and private suspicions. In *Proceedings of the 7th Conference on Theoretical Aspects of Rationality and Knowledge*, TARK '98, pages 43–56. Morgan Kaufmann, 1998.

[3] L. del Cerro, D. Fauthoux, O. Gasquet, A. Herzig, D. Longin, and F. Massacci. Lotrec: the generic tableau prover for modal and description logics. In *Proceedings of the First International Joint Conference on Automated Reasoning*, pages 453–458. Springer, 2001.

[4] H. Ditmarsch, W. Hoek, and B. Kooi. *Dynamic epistemic logic.* Springer, 2007.

[5] J. Hintikka, I. Halonen, and A. Mutanen. Interrogative logic as a general theory of reasoning. In D. M. Gabbay, R. H. Johnson, H. J. Ohlbach, and J. Woods, editors, *Handbook of logic of argument and inference: The turn towards the practical*, pages 295–337. Elsevier, 2002.

[6] M. Khodadadi, R. A. Schmidt, D. Tishkovsky, and M. Zawidzki. Terminating tableau calculi for modal logic K with global counting operators. Manuscript, `http://www.mettel-prover.org/papers/KEn12.pdf`, 2012.

[7] Ş. Minică. *Dynamic Logic of Questions.* PhD thesis, ILLC, University of Amsterdam, 2011.

[8] Ş. Minică, M. Khodadadi, D. Tishkovsky, and R. A. Schmidt. Synthesising and implementing tableau calculi for interrogative epistemic logics, 2012. Long version of the present paper, `http://www.mettel-prover.org/papers/IEL-long.pdf`.

[9] R. A. Schmidt and D. Tishkovsky. Using tableau to decide expressive description logics with role negation. In *Proc. ISWC 2007 + ASWC 2007*, volume 4825 of *LNCS*, pages 438–451. Springer, 2007.

[10] R. A. Schmidt and D. Tishkovsky. A general tableau method for deciding description logics, modal logics and related first-order fragments. In *Proc. IJCAR 2008*, volume 5195 of *LNCS*, pages 194–209. Springer, 2008.

[11] R. A. Schmidt and D. Tishkovsky. Automated synthesis of tableau calculi. *Logical Methods in Computer Science*, 7(2):1–32, 2011.

[12] B. D. ten Cate. *Model Theory for Extended Modal Languages.* PhD thesis, ILLC, University of Amsterdam, 2005.

[13] D. Tishkovsky, R. A. Schmidt, and M. Khodadadi. MetTeL2: Towards a tableau prover generation platform. In these proceedings, 2012.

[14] J. van Benthem. *Logical dynamics of information and interaction.* Cambridge Univ. Press, 2011.

[15] J. van Benthem and Ş. Minică. Toward a dynamic logic of questions. In Xiangdong He, John F. Horty, and Eric Pacuit, editors, *Logic, Rationality, and Interaction*, volume 5834 of *Lecture Notes in Computer Science*, pages 27–41. Springer, 2009.

[16] J. van Eijck. Hylotab: Tableau-based theorem proving for hybrid logics, 2002. Manuscript, CWI, Amsterdam.

[17] J. van Eijck. DEMO: A demo of epistemic modelling. In *Interactive Logic. Selected Papers from the 7th Augustus de Morgan Workshop, London*, volume 1, pages 303–362, 2007.

[18] A. Wiśniewski. Erotetic search scenarios, problem solving, and deduction. *Logique & Analyse*, 185-188:139–166, 2004.

# CDCL with Less Destructive Backtracking through Partial Ordering

Anthony Monnet, Roger Villemaire
Université du Québec à Montréal
Montreal, Quebec, Canada
anthonymonnet@aol.fr, villemaire.roger@uqam.ca

**Abstract**

Conflict-driven clause learning is currently the most efficient complete algorithm for satisfiability solving. However, a conflict-directed backtrack deletes potentially large portions of the current assignment that have no direct relation with the conflict. In this paper, we show that the CDCL algorithm can be generalized with a partial ordering on decision levels. This allows keeping levels that would otherwise be undone during backtracking under the usual total ordering. We implement partial ordering CDCL in a state-of-the-art CDCL solver and show that it significantly ameliorates satisfiability solving on some series of benchmarks.

## 1 Introduction

Conflict-driven clause learning (CDCL) [14] is a very efficient algorithm for solving the propositional satisfiability problem, currently used in virtually all complete state-of-the-art SAT solvers. For each conflict, it deduces a new clause that will allow an early detection of future similar conflicts, thus helping to prune the search space. It also performs a conflict-directed backtracking, which may undo several decision levels at once in order to return faster to the cause of the conflict and propagate this new learnt clause as early as possible in the search tree.

Despite this approach was proved very effective, each conflict-directed backtrack deletes a possibly large amount of instantiations that have no direct connection with the detected conflict. Indeed, by definition, none of the deleted levels contains any variable from the conflict, except for the conflict level itself. In the worst case, these levels could even belong to a distinct connected component of the problem, meaning that they can't be affected by the conflict and the resulting assertion, even indirectly. This results in a partial loss of previous search work, which may delay the discovery of a model or of another conflict. CDCL may have to rebuild this part of the search and reprocess all propagations. Given that propagations are the most time-consuming task of SAT solving, it is natural to try avoiding the destruction of instantiations that are still consistent with the current state. Several methods have been conceived to tackle this issue and minimize the amount of unrelated instantiations that are deleted, for instance tree decompositions [10, 3, 12, 5, 16] and phase saving [19].

In this paper, we propose a novel variation of the CDCL algorithm that detects instantiations that would be undone by the regular algorithm but can be safely retained. This is achieved by relaxing the ordering between decision levels. Indeed, with the usual total order, conflict-directed backtracking must delete all levels above the assertion level in order to return to that level and propagate the conflict clause. We show that this total ordering is not required to maintain essential properties of the algorithm, and that a partial ordering reflecting dependencies between decision levels can be used instead. As a consequence, instantiations are only deleted by the conflict-directed backtracking if they actually interfere with the conflict resolution. Partial order backtracking [7, 15, 4] has previously been described for the Constraint Satisfaction Problem (CSP), but to the best of our knowledge, it has never been used in the context of SAT solving, moreover within the CDCL algorithm. This is the main contribution of our paper.

We also provide experimental results obtained by implementing partial order CDCL (PO-CDCL) in a state-of-the-art CDCL solver. We show that although PO-CDCL is not efficient on all SAT benchmarks, it seems to significantly reduce the solving trace on instances with a low partial order density, and that some benchmark series have a consistantly low density. Thus PO-CDCL manages to solve these series faster than the original CDCL solver.

The rest of this paper is organized as follows: section 2 summarizes the CDCL algorithm. Section 3 quickly introduces previous related works, namely tree decompositions, phase saving and partial order CSP. Section 4 presents the algorithm of partial order CDCL and gives the proof of some of its essential properties. Finally, section 5 shows and analyzes experimental results obtained by our implementation of PO-CDCL.

## 2 Conflict-Driven Clause Learning

Let $\mathcal{V}$ be a set of variables and $\mathcal{L} = \{v, \neg v \mid v \in \mathcal{V}\}$ the set of literals on $\mathcal{V}$. A propositional formula in conjunctive normal form $\mathcal{F}(\mathcal{V}, \mathcal{C})$ is defined by a set $\mathcal{V}$ of variables and a set $\mathcal{C}$ of clauses on $\mathcal{V}$, each clause $c \in \mathcal{C}$ being a set of literals. An assignment $\sigma \subset \mathcal{L}$ is a set of non-conflicting literals considered true. $\sigma$ can be extended and interpreted as a partial function associating boolean values to variables, literals, clauses and formulas. If $\sigma$ is defined on $v \in \mathcal{V}$, we will say that $v$ is instantiated by $\sigma$; if it isn't, we will note $\sigma(v) = \text{undef}$. A total assignment $\sigma$ on $\mathcal{V}$ is a model of the formula $\mathcal{F}(\mathcal{V}, \mathcal{C})$ iff $\sigma(\mathcal{F}(\mathcal{V}, \mathcal{C})) = \text{true}$. Given a formula, the SAT problem consists in determining whether it is satisfiable, i.e. whether it has at least one model.

The CDCL algorithm [14] determinates the satisfiability of a formula through a combination of depth-first search and inference. Algorithm 1 presents a pseudocode of CDCL. The search

**Algorithm 1** CDCL

```
1: $\sigma \leftarrow \emptyset$  /* begin with the empty assignment */
2: loop
3:     $c \leftarrow$ PROPAGATE /* propagate new instantiations */
4:     if $c \neq$ NIL then /* a conflict was found during propagations */
5:         if $\lambda = 0$ then /* conflict at decision level 0 */
6:             return false /* $\mathcal{F}$ is unsatisfiable */
7:         else
8:             $\gamma \leftarrow$ ANALYZE($c$) /* infer the conflict clause $\gamma$ */
9:             $a \leftarrow$ ASSERTIONLEVEL($\gamma, \lambda$)
10:            BACKTRACK($a$) /* backtrack to assertion level */
11:            $\lambda \leftarrow a$ /* $a$ becomes the current level */
12:            $\mathcal{C} \leftarrow \mathcal{C} \cup \{\gamma\}$ /* $\gamma$ is learnt */
13:            PROPAGATEASSERTION($\gamma$)
14:    else /* no conflict during propagations */
15:        if all variables are instantiated then
16:            return $\sigma$ /* $\sigma$ is a model of $\mathcal{F}$ */
17:        else
18:            $\lambda \leftarrow$ NEWLEVEL
19:            DECIDE($\lambda$)
```

**Algorithm 2** AssertionLevel($\gamma, \lambda$) *[CDCL]*

```
$a \leftarrow \max(\{\lambda(l) \,|\, l \in \gamma\} \setminus \{\lambda\})$
return $a$
```

**Algorithm 3** Backtrack($a$) *[CDCL]*

```
for $v \in \mathcal{V} \,|\, \lambda(v) > a)$ do
    $\sigma(v) \leftarrow$ undef
```

part of the algorithm is conducted by repeatedly choosing instantiations to add to the current assignment $\sigma$ (through the procedure Decide) until either all variables are instantiated or a conflict is reached. Conflicts are solved by undoing some of the last search choices.

The inference engine used within CDCL is the unit propagation rule: for any clause $c = \{l_1, \ldots, l_i\}$ such that $\sigma(l_1) = \sigma(l_2) = \ldots = \sigma(l_{i-1}) =$ false and $\sigma(l_i) =$ undef, $c$ entails $l_i$ under $\sigma$ so $l_i$ is added to $\sigma$. $c$ is called the antecedent of $l_i$, noted $\alpha(l_i) = c$. Unit propagation is exhaustively applied to all unit clauses by procedure Propagate before making any new decision. The $n^{\text{th}}$ decision and all unit propagations it entails form the $n^{\text{th}}$ decision level of the search; all propagations which were deduced without any decision belong to decision level 0. We will note $\lambda(v)$ the decision level of a variable $v$ and $\lambda$ the current level of the search.

Propagate encounters a conflicts if it finds a clause $c$ for which all literals are false under the current assignment $\sigma$. CDCL Analyzes this conflict and its reasons to produce a conflict clause $\gamma$ which is also falsified by $\sigma$ but only has one literal of current decision level $\lambda$. If $\lambda = 0$, then the conflict can't be avoided and $\mathcal{F}$ is unsatisfiable. Else $\gamma$ defines an AssertionLevel $a$, which is the second largest decision level in this clause (Alg. 2). CDCL performs a Backtrack to the assertion level by entirely deleting all decision levels above $a$ (Alg. 3). $\gamma$ becomes unit, is propagated by PropagateAssertion, and Propagate is called again to deduce all possible inferences from this new instantiation. When a call to Propagate exhausts all unit propagations without encountering any conflict, a new decision is taken. If all variables have already been instantiated then $\sigma$ is a model of $\mathcal{F}$.

All modern CDCL solvers implement unit propagation using watched literals [17], a method allowing a very efficient detection of unit propagations. Its pseudocode is shown by Alg. 4. As long as a clause has at least two literals that aren't false under $\sigma$, it can't be propagated. Therefore, for each clause $c$, CDCL keeps track of two of its literals $\omega(c) = \{w_1, w_2\} \subseteq c$. For each new propagation $l$, CDCL checks all clauses $c$ where $\neg l$ is watched. If the second watched literal $w$ is true under $\sigma$, then $c$ is true and obviously can't be propagated; CDCL doesn't need to replace $\neg l$. Else, CDCL looks for another non-false literal $w'$ to watch instead of $\neg l$. If it can't find one, either $w$ is false (and $c$ is a conflict), or $w$ is undefined. In the latter case, $c$ is unit and $w$ is added to $\sigma$.

Checking clauses for propagations (lines 4 to 17 of Alg. 4) is the innermost loop of the Propagate procedure, which is generally by far the procedure in which the most time is spent during solving. Because of this, we will use in the rest of the paper the number of clause checks as a secondary indicator of solving efficiency, less implementation-dependent than solving time.

Note that the Backtrack procedure is described here as implemented in zChaff [17] and Glucose [1] for instance. One of the original CDCL solvers GRASP [14] uses a less destructive backtracking: it generally only deletes the last decision level and instantiates the assertion as a new "pseudo-decision". It only performs an actual conflict-directed backtracking when the decision at the conflict level is already a pseudo-decision itself. Although pseudo-decisions allow the use of less destructive backtracks, they are propagations stored in a pseudo-

**Algorithm 4** PROPAGATE *[CDCL]*

```
1: $\Pi \leftarrow$ {instantiations not yet propagated}
2: while $\{\Pi \neq \emptyset\}$ do
3:     choose $l \in \Pi$
4:     for $c \in \mathcal{C} \mid \neg l$ is watched in $c$ do
5:         $w \leftarrow$ the second watched literal in $c$
6:         if $\sigma(w) \neq$ true then
7:             $\Omega \leftarrow \{l' \in c \mid \sigma(l') \neq \text{false}\} \setminus \{w\}$
8:             /* $\Omega$ is the set of literals that could replace $\neg l$ */
9:             if $\Omega = \emptyset$ then /* no other literal in $c$ can be watched */
10:                if $\sigma(w) =$ undef then /* $c$ is unit */
11:                    $\sigma(w) \leftarrow$ true /* $w$ is propagated by $c$*/
12:                    $\Pi \leftarrow \Pi \cup \{w\}$
13:                else
14:                    return $c$ /* $c$ is a conflict */
15:            else
16:                choose $w' \in \Omega$
17:                $\omega(c) \leftarrow \{w, w'\}$ /* $w'$ is watched instead of $\neg l$ */
18:    $\Pi \leftarrow \Pi \setminus \{l\}$
19: return $NIL$ /* no conflict occured */
```

level without any other literal of their antecedent. Therefore they can be deleted without these causes, leaving an undetected unit clause. GRASP-type backtracks thus do not ensure that all possible unit propagations have been performed before taking a new decision, unlike backtracks used in zChaff and Glucose. As a result, conflicts discovered by GRASP can involve clauses that were already unit several decision levels earlier, which means these conflicts could have been avoided much earlier in the search by an exhaustive unit propagation. As unit propagations are crucial for efficiently pruning the search space, we suspect that the incompleteness of unit propagations in GRASP is partly responsible for its lower performance wrt. zChaff ([17], Section 4.4.4. of [13]). Enforcing complete unit propagations within a GRASP backtrack type would require to exhaustively check all clauses after each conflict, which may be time expensive, and would still cause pseudo-decisions. In contrast, PO-CDCL aims to reduce the amount of instantiations undone during conflict-directed backtracking while keeping the exhaustive unit propagation property.

## 3 Related Works

Several methods have been proposed to directly or indirectly minimize the quantity of search progress lost during conflict-directed backtracking while solving SAT or CSP problems.

Some of them rely on tree decompositions [20] of the connectivity graph between variables. They constrain the order of decision variables so that the instance first breaks into several connected components, and then that the solving of one component can't undo instantiations in another component [10, 3, 12, 5, 11]. The main practical drawback is that challenging SAT problems are typically so large that computing a good decomposition becomes untractable [16].

Phase saving [19] is a more heuristic and very lightweight approach. It simply memorizes the last polarity assigned to a variable and reuses it if the variable is picked for a decision.

Phase saving actually doesn't prevent instantiations from being undone, but makes it possible to rediscover the deleted instantiations later and recover the search progress. This recovery however doesn't save the cost of repeating the time-consuming propagation phase.

A set of CSP solving algorithms was designed with the goal of undoing less search progress than conflict-directed backjumping (CBJ) by relaxing the order between variables. While CBJ resolves a conflict by deleting all instantiations and restoring all eliminated values from the culprit variable on (the most recent variable in the nogood), dynamic backtracking (DB) [7] only undoes the culprit variable and restores only eliminated values where the culprit variable was part of the nogood. Partial order backtracking (POB) [15] uses the same backtrack as DB and additionally allows to pick any variable in the nogood as the culprit variable. To ensure termination, it however progressively sets permanent order constraints between variables. Both algorithms were hybridated [8] and generalized [4].

Similarly to DB and POB, PO-CDCL undoes less search progress than regular conflict-directed algorithms, and like POB it allows some freedom in the choice of the assertion level. However, instead of setting definitive constraints on the order of variable instantiations, it sets local constraints on the order in which decision levels will be undone. Moreover, PO-CDCL is specifically adapted to various aspects of CDCL, such as the integration of unit propagations and the watched literal mechanism, which correctness implicitely relies on the total order between decision levels.

Finally, some techniques aim to enhance performances of SAT solvers by increasing the quantity of instantiations undone by backtracks [18, 2], which is a totally opposite strategy wrt. PO-CDCL.

## 4 Partial Order CDCL

This section introduces PO-CDCL, a generalization of the usual CDCL that relies on a partial order on decision levels during the search. In the first subsection, we will present the algorithm of PO-CDCL, and in the second we will show amongst others that it is correct and complete and that it terminates.

### 4.1 Algorithm

Algorithm 1, that we used to describe CDCL, remains the backbone of PO-CDCL, but some of its elements are modified.

In the original CDCL, decision levels are assumed to be totally ordered such that $i < j$ iff the decision of level $i$ was set before the decision of level $j$. In PO-CDCL, we only set a strict partial ordering $\Delta$ between decision levels. We will say that $i$ is a dependency for $j$, or equivalently that $j$ depends on $i$, and note $i <_\Delta j$ if $(i, j) \in \Delta$. $i \leq_\Delta j$ is the reflexive extension of $<_\Delta$. $i <_\Delta j$ means that decision level $i$ had an influence on propagations at level $j$. Consequently, level $j$ should be deleted when level $i$ is deleted or modified. Two cases of the Propagate procedure add dependencies between levels (see Alg. 5):

1. At lines 14 and 15, when a unit clause $c = \{l_1, \dots, l_i\}$ propagates the literal $l_i$, then this propagation at the current level obviously depends on all other levels occurring in $c$: $\lambda(l_1), \dots, \lambda(l_{i-1}) <_\Delta \lambda$ (except when $\lambda(l_j) = \lambda$).
2. At line 7, when $\sigma(w) =$ true, we add the dependency $\lambda(w) <_\Delta \lambda$ if $\lambda(w) \neq \lambda$. Indeed, in this case a clause $c$ is checked because one of its watched literals $\neg l$ is false, but $\neg l$ doesn't need to be replaced because the second watched literal $w$ is true. $w$ is the reason why we

**Algorithm 5** PROPAGATE *[PO-CDCL]*

```
 1: $\Pi \leftarrow$ {instantiations not yet propagated}
 2: while $\{\Pi \neq \emptyset\}$ do
 3:     choose $l \in \Pi$
 4:     for $c \in \mathcal{C} \,|\, \neg l$ is watched in $c$ do
 5:         $w \leftarrow$ the second watched literal in $c$
 6:         if $\sigma(w) =$ true then
 7:             $\Delta \leftarrow \Delta \cup \{(\lambda(w), \lambda)\}$ /* $\lambda$ depends of $\lambda(w)$ */
 8:         else
 9:             $\Omega \leftarrow \{l' \in c \,|\, \sigma(l') \neq \text{false}\} \setminus \{w\}$
10:             /* $\Omega$ is the set of literals that could replace $\neg l$ */
11:             if $\Omega = \emptyset$ then /* no other literal in $c$ can be watched */
12:                 if $\sigma(w) =$ undef then /* $c$ is unit */
13:                     $\sigma(w) \leftarrow$ true /* $w$ is propagated by $c$*/
14:                     for $l' \in c \setminus \{w\} \,|\, \lambda(l') \neq \lambda$ do
15:                         $\Delta \leftarrow \Delta \cup \{(\lambda(l'), \lambda)\}$ /* $\lambda$ depends of $\lambda(l')$ */
16:                     $\Pi \leftarrow \Pi \cup \{w\}$
17:                 else
18:                     return $\{c\}$ /* $c$ is a conflict */
19:             else
20:                 choose $w' \in \Omega$
21:                 $\omega(c) \leftarrow \{w, w'\}$ /* $w'$ is watched instead of $\neg l$ */
22:     $\Pi \leftarrow \Pi \setminus \{l\}$
23: return $\emptyset$ /* no conflict occured */
```

**Algorithm 6** ASSERTIONLEVEL($\gamma, \lambda$) *[PO-CDCL]*

```
$\Theta \leftarrow \{\lambda(l) \,|\, l \in \gamma\} \setminus \{\lambda\}$ /* $\Theta$ is the set of levels involved in the conflict, except $\lambda$*/
$\Gamma \leftarrow \{i \in \Theta, \nexists j \in \Theta \,|\, i <_\Delta j\}$ /* $\Gamma$ is the set of maximal elements in $\Theta$ */
choose $a \in \Gamma$
return $a$
```

can stop watching $c$ for unit propagations, but we have to make sure that $w$ will not be uninstantiated before $\neg l$, else $c$ could become unit without being properly watched. This is impossible with a total order on decision levels but could happen with a partial order.

ASSERTIONLEVEL also has to be modified, as indicated in Alg. 6. Partial order will allow some freedom in the choice of the assertion level. In CDCL, it is uniquely defined as the largest level in the set $\Theta = \{\lambda(l) \,|\, l \in \gamma\} \setminus \{\lambda\}$ of decision levels involved in the conflict clause, minus the current decision level. In PO-CDCL, due to the partial order, $\Theta$ may have several largest elements. Each of these largest elements is eligible as a valid assertion level, so that the assertion level can be arbitrarily picked amongst them.

Finally, we also modify BACKTRACK (see Alg. 7) since the goal of our method is to undo less instantiations during this phase. CDCL resolves a conflict by undoing all instantiations which decision level is larger than the assertion level $a$. PO-CDCL performs a similar deletion, except that it only deletes decision levels $i$ such that $a <_\Delta i$ ($\lambda$ may not depend on $a$ but must obviously be deleted in any case). This deletion ensures the antisymmetry of $\Delta$: if a level $i$ such that $a <_\Delta i$ wasn't deleted, the search returning to level $a$ may produce a propagation of level

**Algorithm 7** BACKTRACK($a$) *[PO-CDCL]*

```
$\Lambda \leftarrow$ the set of all decision levels
for $i \in \Lambda \mid (a <_\Delta i)$ or $(i = \lambda)$ do
    for $v \in \mathcal{V} \mid \lambda(v) = i$ do
        $\sigma(v) \leftarrow$ undef
```

$a$ depending on level $i$, so we would have simultaneously $a <_\Delta i$ and $i <_\Delta a$. The antisymmetry of $\Delta$ is crucial to ensure that the assertion level of a conflict is well-defined. Indeed, without this property, the set of decision levels involved in a conflict clause may not have any maximal element.

Note that we should also always enforce $\forall i \neq 0$, $0 <_\Delta i$; else, when backtracking to level 0, it would be possible to make a propagation at top-level which depends on a decision.

## 4.2 Properties

In this subsection, we will prove some properties of PO-CDCL, including that it is correct, complete and that it terminates. Most other properties we will prove are implicit or obvious properties within the original CDCL, but are less straightforward in the case of a partial order.

**Proposition 1.** *$\Delta$ is antisymmetric.*

*Proof.* Algorithm 5 only adds dependencies to the current decision level $\lambda$. To show the antisymmetry of $\Delta$, it is thus sufficient to prove that no other level depends on $\lambda$ at the moment it becomes the current level. $\lambda$ can be a newly created decision level, in which case it has initially no dependency. Else, the search returned to $\lambda$ because it has been chosen as the assertion level for some conflict. Then BACKTRACK deleted all decision levels which depended on $\lambda$. In both cases, no non-empty level depends on $\lambda$. □

**Corollary 1.** *$\Delta$ is a strict partial order.*

**Definition 1.** *A propagation $l$ is valid iff $\forall a \in \alpha(l)$, $\upsilon(a) = false$.*

**Proposition 2.** *During a PO-CDCL solving, all propagations remain valid.*

*Proof.* The only way to make a propagation invalid would be to delete a level to which a literal from its antecedent belongs, without deleting the level of the propagation itself. Dependencies added in Alg. 5 when a propagation occurs ensure that such a case can't happen. □

**Definition 2.** *A SAT solver is propagation-complete iff when its* PROPAGATE *function stops without having detected a conflict, no more clause is unit.*

**Lemma 1.** *Whenever* PROPAGATE *terminates without encountering any conflict, the following propreties hold. All clauses not yet satisfied watch two undefined literals. Satisfied clauses may watch true, false, or undefined literals, but each clause watches at most one false literal. If a satisfied clause watches a false literal $w_1$, the second watched literal $w_2$ is true, and $\lambda(w_2) \leq_\Delta \lambda(w_1)$.*

*Proof.* We will prove the lemma by recurrence on conflictless calls to PROPAGATE.

**Initialization:** Before the initial propagation round of the search, all variables are uninstantiated, so all clauses are unsatisfied and watch two undefined variables.

Assume a clause $c$ whose two watched literals become false. PROPAGATE will eventually

check one on them and try to replace it by a true or undefined literal. If it fails, it means that the clause is already unsatisfiable before any decision was made, so the entire formula is unsatisfiable. If it succeeds, the clause now belongs to the next case.
Now assume a clause $c$ with only one false watched literal $w_1$. If the second watched literal $w_2$ is true, then $c$ is true and $\lambda(w_1) = \lambda(w_2) = \lambda_0$ so the property is true. If $w_2$ is undefined, PROPAGATE will look for a second non-false literal $w_3$. If there is one, $c$ will watch $w_3$ instead of $w_1$. Else, it means that the clause is unit, so $w_2$ is added to the current assignment and $c$ is then a true clause watched by one true and one false literal of the same level.

**Recurrence:** Let's assume the property holds after the $n^{\text{th}}$ conflictless call to PROPAGATE.
If the property holds before the $(n+1)^{\text{th}}$ conflictless call, then we can prove it still holds after this call using the same reasoning as for the initialization phase. However, there may be one or more conflictual calls between the $n^{\text{th}}$ and $(n+1)^{\text{th}}$ conflictual call. We will now show by another recurrence that after the backtrack following any of these conflictual calls (but before the learnt clause is added to the formula) the recurrence is verified.
Let's assume the property holded after the previous backtrack (or after the last conflictless call in the case of the initialization). When a conflict occurs, then several decision levels, including the current level, are undone. After a backtrack, all clauses are then either in a state verifying the recurrence property, or in a state reached by deinstantiating some literals from such a recurrence state.
Let $c$ be a clause, $w_1$, $w_2$ its watched literals and $\sigma$, $\sigma'$ the partial assignments resp. before the conflictual call and after the following backtrack (so $\sigma' \subseteq \sigma$).

- If $\sigma(c) = \text{undef}$, then by recurrence $\sigma(w_1) = \sigma(w_2) = \text{undef}$. Since $\sigma' \subseteq \sigma$, $\sigma'(w_1) = \sigma'(w_2) = \text{undef}$ so the property still holds.
- If $\sigma(c) = \text{true}$ and $\sigma(w_1) = \text{false}$, then by recurrence $\sigma(w_2) = \text{true}$ and $\lambda(w_2) \leq_\Delta \lambda(w_1)$.
  - If $\sigma'(w_2) = \text{true}$, $\sigma'(c) = \text{true}$ and the property still holds regardless of $\sigma'(w_1)$.
  - Else $\sigma'(w_2) = \text{undef}$. Since $\lambda(w_2) \leq_\Delta \lambda(w_1)$, $\sigma'(w_1) = \text{undef}$, so the property holds regardless of $\sigma'(c)$.
- If $\sigma(c) = \sigma'(c) = \text{true}$ and $\sigma(w_1)$, $\sigma(w_2) \neq \text{false}$, then the property holds regardless of $\sigma'(w_1)$ and $\sigma'(w_2)$.
- If $\sigma(c) = \text{true}$, $\sigma(w_1)$, $\sigma(w_2) \neq \text{false}$ and $\sigma'(c) = \text{undef}$, then $\sigma'(w_1) = \sigma'(w_2) = \text{undef}$ so the property holds.

□

**Corollary 2.** *After a conflictless run of* PROPAGATE, *no clause is false under the current assignment.*

**Proposition 3.** *PO-CDCL is propagation-complete.*

*Proof.* According to Lem. 1, after a conflictless run of PROPAGATE, all unsatisfied clauses watch two distinct undefined literals. Hence, none of these clauses is unit (which proves Prop. 3) or false (which proves 2). □

**Theorem 1.** *PO-CDCL is correct.*

*Proof.* A SAT solver is correct iff any total assignment it returns is indeed a model of the input formula, i.e. if it satisfies all clauses. A total assignment can only be returned by PO-CDCL after a conflictless run of Propagate. According to Cor. 2, no clause is false under this assignment. As the assignment is total, no clause can be undefined either. So all clauses are satisfied, and the total assignment is a model. □

**Theorem 2.** *PO-CDCL is complete.*

*Proof.* A SAT solver is complete iff it never erroneously reports a satisfiable formula as being unsatisfiable. Lemma 3 of [22] proves the completeness of CDCL by showing that the empty clause can be derived by recursively resolving the final conflict clause against the antecedents of its variables. This proof is also valid within CDCL because according to Prop. 2 all propagations are valid, hence all literals of its antecendent are still false, except for the propagation itself. The proof also shows that the resolution is finite, since the process doesn't resolve against the same variable twice. This is also still true in PO-CDCL, because $\Delta^+$ is a partial order (Cor. 1). □

**Theorem 3.** *PO-CDCL always terminates.*

*Proof.* $\forall i \in \mathbb{N}$, let $\Lambda_i$ and $\Delta_i$ be the set of decision levels and the associated partial order after the first $i$ instantiations in the PO-CDCL search ("at time $i$"). If the search terminates after $n$ instantiations, we will assume that $\forall i > n$, $\Lambda_i$ and $\Delta_i$ represent the state at the end of the search. PO-CDCL as we described it never actually deletes any decision level or dependency, so we can write $\forall i < j \in \mathbb{N}$, $\Lambda_i \subseteq \Lambda_j$ and $\Delta_i \subseteq \Delta_j$. Let us define the (possibly infinite) sets of all decision levels and dependencies during the search: $\Lambda_\infty = \bigcup_{i\in\mathbb{N}} \Lambda_i$, $\Delta_\infty = \bigcup_{i\in\mathbb{N}} \Delta_i^+$. Thanks to the infinite chain of inclusions on $(\Lambda_i)_{i\in\mathbb{N}}$ and $(\Delta_i)_{i\in\mathbb{N}}$, $\Delta_\infty$ is a partial order on $\Lambda_\infty$, and $\forall i \in \mathbb{N}$, $\Delta_\infty \cap (\Lambda_i \times \Lambda_i)$ is a partial order on $\Lambda_i$. Let $\Psi$ be any total order extending $\Delta_i$. Similarly, its restriction to $\Lambda_i \times \Lambda_i$ is a total order on $\Lambda_i$. We now have a total order on all decision levels which is compatible with the local partial order at any point of the search.

$\forall i \in \mathbb{N}$, $\forall j \in \Lambda_\infty$, let us note $k_i(j)$ the number of variables instantiated at level $j$ at time $i$ (or at the end of the search if it terminated after less than $i$ instantiations).

$$\rho_i(j) = \begin{cases} 0 & \text{if}\, j = 0 \,\text{or}\, k_i(j) = 0 \\ |\{k \in \Lambda_\infty \setminus \{0\} \mid k <_\Psi j \,\text{and}\, k_i(j) \neq 0\}| + 1 & \text{else} \end{cases}$$

is a function that orders all non-empty decision levels at time $i$ according to $\Psi$. Finally, let us define

$$f(i) = \sum_{j \in \Lambda_\infty} \frac{k_i(j)}{|\mathcal{V}|^{\rho_i(j)+1}} \quad .$$

$f(i)$ is defined, as in Lem. 1 from [22], such that one variable at a decision level $j$ has more weight that the sum of the weight all variables at higher decision levels. As in this lemma, it proves that $f(i)$ is a strictly growing function until the search finishes. Indeed, when some decision levels are uninstantiated, their weight is compensated by the assertion added at assertion level, which is strictly lower than all undone levels.[1] Similarly, the weight of a decision level can decrease when a decision is taken in a formerly empty level with a lower $\rho$ order, but again their weight loss is compensated by the higher weight of this new decision. As $f(i)$ strictly grows as long as the search continues and can only take a finite number of values, the search is finite. □

[1]this proof assumes that for each conflict we set that the conflict level depends of the assertion level, which has been omitted from the presented code but can be added without inconsistency.

**Algorithm 8** ANALYZE($\phi$)

```
/* $\phi$ is the false clause detected during unit propagation */
/* $\gamma$ will be the conflict clause produced by conflict analyzis */
$\gamma \leftarrow \phi$
while $\{|\{l \in \gamma \mid \lambda(l) = \lambda\}| > 1\}$ do
    /* there remains more than one literal of level $\lambda$ in $\gamma$ */
    $l \leftarrow$ LAST($\gamma, \lambda$)  /* pick the last instantiated literal of level $\lambda$ in $\gamma$ */
    $\gamma \leftarrow \gamma \otimes_{var(l)} \alpha(l)$  /* resolution of $\gamma$ and $\alpha(l)$ on the variable of $l$ */
```

**Definition 3.** *A learnt clause is non-redundant if obtained by resolving at least two clauses of the formula. A conflict is non-redundant if its analyzis produces a non-redundant learnt clause.*
*A learnt clause is useful if it becomes unit after the backtrack.*

**Proposition 4.** *All clauses learnt during a PO-CDCL are non-redundant and useful.*

*Proof.* A shown by Alg. 8, if the conflict clause is produced without any resolution, it means that the false clause $\phi$ only contained one literal of level $\lambda$. This implies that before the propagation round responsible for the conflict, either $\phi$ was already false or it was unsatisfied with only one undefined variable. Both possibilities can be ruled out using the proof of Lem. 1. Hence all clauses learnt during PO-CDCL are non-redundant.
Before the backtrack, $\gamma$ contains by definition exactly one literal at the conflict level. Since the conflict level is always undone by the backtrack, $\gamma$ is unit after the backtrack unless another decision level involved in the conflict is undone. The latter case is impossible by definition of the assertion level (see Alg. 6). Therefore $\gamma$ is useful. □

## 5 Experimental Results

In order to evaluate the practical efficiency of PO-CDCL, we implemented PO-GLUCOSE[2] as a modification of state-of-the-art solver GLUCOSE 1.0 [1]. Glucose was chosen because it has ranked as one of the most efficient solvers on application benchmarks during the last SAT competitions and races [?] and is based on MINISAT [6] which has also been a regular winner of these competitions.

Our implementation does not explicitely store the entire partial order $\Delta$; instead, we only keep track of all direct dependencies between decision levels. The algorithm only requires to find all levels depending directly or indirectly of candidate assertion levels during the ASSERTIONLEVEL procedure, which can be easily done by a few recursive traversals of the dependency tree from these levels. Maintaining the full transitive relation $\Delta$ would require a time-expensive enforcement of transitivity after each new propagation, which is much less efficient according to our preliminary tests.

For the choice of the assertion level, we kept in our experiments the basic CDCL strategy by choosing amongst candidate assertion levels the latest created one. We don't modify restarts (they still undo all instantiations except top level assertions), nor their frequency.

GLUCOSE uses phase saving by default. As we partly designed PO-CDCL as an alternative to phase saving, we disabled it in our implementation PO-GLUCOSE. Moreover, preliminary

[2]Source code of PO-GLUCOSE is available at `http://www.info2.uqam.ca/~villemaire_r/Recherche/SAT/120210partial_order_glucose.tar.gz`

Table 1: Compared performances of Glucose without phase saving (*TO*), Glucose with phase saving (*TO-phase*) and PO-Glucose (*PO*) on the set of 300 application benchmarks from the SAT 2011 competition. The first line shows the total solving time for each implementation (*tot.*), counted in days, hours and minutes. Each instance was given a time limit of one hour, the number of instances that couldn't be solved within that limit is indicated in column *#to*. The second line gives the total number of clause checks needed for solving all instances (checks are counted in billions). Limit was set to 100 billions of checks for each instance, the number of unsolved instances is again given in column *#to*.

| | TO | | TO-phase | | PO | |
|---|---|---|---|---|---|---|
| | #to | tot. | #to | tot. | #to | tot. |
| time (d:hh:mm) | 122 | 6d02h05m | 111 | 5d15h47m | 144 | 7d03h23m |
| clause checks (Bn) | 113 | 13 911 | 103 | 12 869 | 127 | 15 200 |

experiments indicated us that enabling phase saving in PO-Glucose almost always caused a signification degradation of performances. In order to make sure that performance differences were not solely caused by disabling phase saving, PO-Glucose was compared with the original Glucose implementation including phase saving, but also with a slight variant where phase saving was disabled. Experiments were conducted on a 3.16 GHz Intel Core 2 Duo CPU with 3 GB of RAM, running a Ubuntu 11.10 OS.

Our tests confirm that in practice the PO-CDCL algorithm is able to save instantiations compared to regular CDCL during the solving of any non-trivial benchmark, although the average number of instantiations saved per conflict varies a lot amongst benchmarks (from less than one to several thousands).

In order to test the behaviour of PO-Glucose on a wide range of SAT benchmarks, we ran it on the set of 300 application benchmarks from the SAT 2011 Competition. Results are summarized in Table 1. They clearly show that in general PO-Glucose tends to degradate solving performances compared to Glucose, no matter if phase saving is enabled or not. If we compare PO-Glucose with Glucose with phase saving (the best performing of both Glucose variants), only 26 of the 300 instances are solved faster by PO-Glucose, while 153 are to the contrary solved slower than by Glucose. Glucose is globally slightly less efficient when phase saving is disabled, but even then it still clearly outperforms PO-Glucose.

This counterperformance is partly due to the cost of maintaining and handling dependencies during solving. As we pointed it out, unit propagation is one of the most frequent operation performed during SAT solving and is often responsible for the largest part of the solving time. For all propagations, PO-CDCL requires to ensure that the current decision levels depends on the decision levels of all variables in the antecedent clause. This task is relatively lightweight, but as it occurs very frequently it results in a sensibly slower solving: on average, PO-Glucose performs about 30% less clause checks than Glucose in the same time, and in some extreme cases this decrease can reach 75%. Our implementation of PO-CDCL thus starts with a handicap over the regular CDCL and has to drastically reduce the solving trace in order to outperform it in terms of solving time.

Also, PO-CDCL actually follows a longer search path than CDCL on many instances, despite our original intuition. For instance, amongst the 153 instances on which PO-Glucose takes more time than Glucose, it also performs more clause checks on 143 of them. Since the CDCL algorithm is very sensitive to variations, the partial order may have negative side-effects on some aspects of the algorithm, for instance on the dynamic VSIDS heuristic used to choose decision

Table 2: Compared solving time of Glucose without phase saving (*TO*), Glucose with phase saving (*TO-phase*) and PO-Glucose (*PO*) on some example instances. For each instance, *direct dep.* gives the average direct dependency density $\delta(\Delta_{\text{dir}})$, a lower bound of the actual density $\delta(\Delta)$, during the execution of PO-Glucose. AProVE07-03, homer14.shuffled, post-c32s-gcdm16-23 and k2fix_gr_rcs_w9.shuffled are taken from the application benchmarks of the SAT 2011 competition. 7pipe_k and 12pipe_bug4 are two microprocessor formal verification benchmarks taken respectively from the pipe_unsat_1.0 and pipe_sat_1.0 series.

| | TO | TO-phase | PO | direct dep. |
|---|---|---|---|---|
| AProVE07-03 | 6m24s | 7m16s | 16m57s | 69.13% |
| homer14.shuffled | 7m51s | 10m51s | 25m14s | 39.47% |
| post-c32s-gcdm16-23 | 1m18s | 1m20s | 3m41s | 33.36% |
| k2fix_gr_rcs_w9.shuffled | >1h00m00s | 30m20s | 9m13s | 4.51% |
| 7pipe_k | 23m36s | >1h00m00s | 3m07s | 3.92% |
| 12pipe_bug4 | >1h00m00s | 18m49s | 4m11s | 2.07% |

variables. We think the issue is that on many instances the advantages gained from using a partial order are outweighted by these drawbacks.

The principle of PO-CDCL being to take advantage of some independence between decision levels, the obvious question is whether this is a frequent phenomenon in SAT solving. During the solving of a problem, if decision levels often depend on all or most previously created levels, PO-CDCL will behave very similarly to CDCL. In that case the overhead of PO-CDCL obviously comes with little benefit. The independence between decision levels can be measured by the density of the partial order $\Delta$.

At any point of the search, let $l$ be the current number of decision levels (not including level 0). We will define the cardinality of $\Delta$ as $|\Delta| = |\{(i,j),\, i <_\Delta j\}|$, i.e. the number of dependencies between decision levels. The maximal cardinality for $l$ decision levels is $|\Delta|_{\max}(l) = (l-1)(l-2)$; it is reached iff $\Delta$ is a total order on the $l$ levels. The current density of $\Delta$ is then defined by $\delta(\Delta) = \frac{|\Delta|}{|\Delta|_{\max}(l)}$. A low density (near 0) means that there are very few dependencies between decision levels compared to the maximum possible number of dependencies given the current number of decision levels. Conversely, a value of $\delta(\Delta)$ approaching 1 denotes a high amount of dependencies and means that $\Delta$ is close to defining a total order on decision levels. Considering the previous discussion, we expect PO-Glucose to perform better on instances with a low average value of $\delta(\Delta)$ during its execution.

Table 2 shows this average value on some example instances, or more exactly a lower bound of it: the average value of $\delta(\Delta_{\text{dir}}) = \frac{|\Delta|_{\text{dir}}}{|\Delta|_{\max}(l)}$ where $\Delta_{\text{dir}}$ is the set of direct dependencies between decision levels. These examples seem to validate our intuition that PO-Glucose has more chances to ameliorate performances on instances with low level dependencies. Instances that PO-Glucose solves significantly faster than both Glucose variants often have only around 5% or less of the maximum possible direct dependencies. On the contrary, PO-Glucose tends to generally degradate the solving performance on instances having an average direct density of 30% or more. Partial order CDCL thus has indeed more chances to be efficient on instances where decision levels interact moderately with each other.

Although most SAT instances we thoroughly examined have little independence during the search, we identified at the opposite some benchmark series where all instances share a low dependency level, resulting in most cases in significant solving speedups. For instance,

Table 3: Solving performances of total order Glucose without (TO) and with (TO-phase) phase saving and PO-Glucose (PO) on two benchmark families of formal verification of microprocessors. All tests were run with a time limit of 1 hour. For each test the necessary amounts of time (in seconds) and of clause checks (in millions of checks) is given, and the best performance amongst the three solvers is printed in bold. Average direct density of $\Delta$ is respectively 1.5% on pipe_sat_1.0 and 5% on pipe_unsat_1.0. Some instances of pipe_unsat_1.0 have been ommited: 2pipe_k, which in solved in less then 1s and 1M clause checks by all solvers, and 10pipe_k to 14pipe_k, which all 3 solvers are unable to solve within the time limit.

| | | time (s) | | | checked clauses (M) | | |
|---|---|---|---|---|---|---|---|
| | | TO | TO-phase | PO | TO | TO-phase | PO |
| pipe_sat_1.0 | bug1 | >3 600 | 15 | **9** | >68 766 | 427 | **20** |
| | bug2 | >3 600 | 722 | **17** | >30 290 | 12 122 | **117** |
| | bug3 | 1 875 | **178** | 2 246 | 22 776 | **5 344** | 30 485 |
| | bug4 | >3 600 | 1 702 | **251** | >42 762 | 52 933 | **3 236** |
| | bug5 | 115 | 34 | **25** | 2 261 | 1 181 | **265** |
| | bug6 | 1 750 | 354 | **138** | 35 695 | 10 056 | **1 525** |
| | bug7 | >3 600 | 783 | **389** | >21 687 | 21 393 | **3 902** |
| | bug8 | >3 600 | **1 569** | 3 230 | >84 337 | 35 203 | **31 314** |
| | bug9 | >3 600 | **5** | 13 | >66 840 | **73** | 82 |
| | bug10 | **8** | 1 525 | 282 | **145** | 36 226 | 3 089 |
| | **total** | >25 348 | 6 887 | **6 601** | >375 562 | 174 962 | **74 034** |
| pipe_unsat_1.0 | 3pipe_k | **0** | 2 | 1 | **13** | 82 | 15 |
| | 4pipe_k | **6** | 22 | 15 | **217** | 876 | 385 |
| | 5pipe_k | **13** | 68 | 37 | **520** | 2 604 | 911 |
| | 6pipe_k | 23 | 77 | **9** | 847 | 2 709 | **173** |
| | 7pipe_k | 1 416 | 4 727 | **187** | 56 905 | 228 327 | **3 977** |
| | 8pipe_k | 3 538 | 4 059 | **1 058** | 139 673 | 94 965 | **27 288** |
| | 9pipe_k | 174 | 258 | **150** | 5 948 | 7 187 | **2 006** |
| | **total** | 5 171 | 9 212 | **1 456** | 204 123 | 336 750 | **34 756** |

table 3 shows detailed statistics obtained on two benchmark sets from formal verification of microprocessors [21]. These benchmarks have particularly low dependency between decision levels, as shown in the caption of Table 3 and on a couple of examples in Table 2, and PO-Glucose significantly outperforms both versions of Glucose on most instances. Moreover, the management of dependency structures is particularly time-expensive on these instances. Thus the performance of PO-Glucose is even more significant when purely algorithmic indicators are considered, such as the total number of checked clauses: speedups up to one or even two orders of magnitude are common. This means that on these instances partial ordering CDCL consistently manages to explore the search space much more efficiently than the regular CDCL algorithm. Moreover, one family contains satisfiable benchmarks and the other unsatisfiable benchmarks. Thus PO-CDCL can be efficient not only for reaching quickly a model of the instance, but also for pruning the search space.

## 6 Conclusion

In this paper, we addressed the issue of information loss in CDCL algorithms during conflict-directed backtracks. We designed a variation of CDCL that defines a partial order on decision levels, and showed this order allows to undo less instantiations during backtracks, while keeping all essential properties of the algorithm. Finally, we implemented our algorithm in a state-of-the-art SAT solver and evaluated its efficiency. We noticed that PO-CDCL performs particularly well on benchmarks where the partial order as a low average density during the search. Moreover, some series of benchmarks are characterized by a consistently low density and can be solved significantly faster by PO-CDCL.

We are currently exploring some avenues to further ameliorate performances on instances that we already identified as relevant to partial order CDCL. For instance, the choice of the assertion level was set rather arbitrary in the experiments presented above, but using more relevant strategies to choose this level can lead to even better performances on the formal verification instances on which we focussed in this paper.

## References

[1] Gilles Audemard and Laurent Simon. Predicting learnt clauses quality in modern SAT solvers. In Craig Boutilier, editor, *IJCAI 2009, Proceedings of the 21$^{st}$ International Joint Conference on Artificial Intelligence*, pages 399–404, 2009.

[2] Ateet Bhalla, Inês Lynce, José T. de Sousa, and João Marques-Silva. Heuristic-based backtracking relaxation for propositional satisfiability. *Journal of Automated Reasoning*, 35(1–3):3–24, October 2005.

[3] Per Bjesse, James H. Kukula, Robert F. Damiano, Ted Stanion, and Yunshan Zhu. Guiding SAT diagnosis with tree decompositions. In Giunchiglia and Tacchella [9], pages 315–329.

[4] Christian Bliek. Generalizing partial order and dynamic backtracking. In *AAAI/IAAI '98 Proceedings*, pages 319–325. AAAI Press / The MIT Press, 1998.

[5] Vijay Durairaj and Priyank Kalla. Exploiting hypergraph partitioning for efficient boolean satisfiability. In *Ninth IEEE International High-Level Design Validation and Test Workshop, 2004*, pages 141–146. IEEE Computer Society, 2004.

[6] Niklas Eén and Niklas Sörensson. An extensible SAT-solver. In Giunchiglia and Tacchella [9], pages 502–518.

[7] Matthew L. Ginsberg. Dynamic backtracking. *Journal of Artificial Intelligence Research*, 1:25–46, August 1993.

[8] Matthew L. Ginsberg and David McAllester. GSAT and dynamic backtracking. In Alan Borning, editor, *PPCP'94 Proceedings*, volume 874 of *Lecture Notes in Computer Science*, pages 243–265. Springer, 1994.

[9] Enrico Giunchiglia and Armando Tacchella, editors. *Theory and Applications of Satisfiability Testing – 6$^{th}$ International Conference, SAT 2003. Santa Margherita Ligure, Italy, May 5–8, 2003, Selected Revised Papers*, volume 2919 of *Lecture Notes in Computer Science*. Springer, 2004.

[10] Jinbo Huang and Adnan Darwiche. A structure-based variable ordering heuristic for SAT. In Georg Gottlob and Toby Walsh, editors, *IJCAI-03, Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence*, pages 1167–1172. Morgan Kaufmann, 2003.

[11] Philippe Jégou and Cyril Terrioux. Hybrid backtracking bounded by tree-decomposition of constraint networks. *Artificial Intelligence*, 146(1):43–75, 2003.

[12] Wei Li and Peter van Beek. Guiding real-world SAT solving with dynamic hypergraph separator decomposition. In *16$^{th}$ IEEE International Conference on Tools with Artificial Intelligence (ICTAI 2004)*, pages 542–548. IEEE Computer Society, 2004.

[13] João P. Marques-Silva, Ines Lynce, and Sharad Malik. Conflict-driven clause learning SAT solvers. In Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors, *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*, chapter 4, pages 131–153. IOS Press, 2009.

[14] João P. Marques-Silva and Karem A. Sakallah. GRASP: A search algorithm for propositional satisfiability. *IEEE Transactions on Computers*, 48(5):506–521, May 1999.

[15] David A. McAllester. Partial order backtracking. Research note, Artificial Intelligence Laboratory, MIT, 1993.

[16] Anthony Monnet and Roger Villemaire. Scalable formula decomposition for propositional satisfiability. In *$C^3S^2E$ '10 Proceedings*, pages 43–52. ACM, 2010.

[17] Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik. Chaff: engineering an efficient SAT solver. In *Proceedings of the $38^{th}$ Design Automation Conference (DAC 2001)*, pages 530–535. ACM Press, 2001.

[18] Alexander Nadel and Vadim Ryvchin. Assignment stack shrinking. 6175:375–381, 2010.

[19] Knot Pipatsrisawat and Adnan Darwiche. A lightweight component caching scheme for satisfiability solvers. In João P. Marques-Silva and Karem A. Sakallah, editors, *Theory and Applications of Satisfiability Testing - SAT 2007, $10^{th}$ International Conference*, volume 4501 of *Lecture Notes in Computer Science*, pages 294–299. Springer, 2007.

[20] Neil Robertson and Paul D. Seymour. Graph minors. II. Algorithmic aspects of tree-width. *Journal of Algorithms*, 7(3):309–322, September 1986.

[21] Miroslav N. Velev and Randal E. Bryant. Effective use of boolean satisfiability procedures in the formal verification of superscalar and VLIW microprocessors. *Journal of Symbolic Computation*, 35(2):73–106, February 2003.

[22] Lintao Zhang. *Searching for Truth: Techniques for Satisfiability of Boolean Formulas.* PhD thesis, Princeton University, June 2003.

# qbf2epr: A Tool for Generating EPR Formulas from QBF*

Martina Seidl[1,2], Florian Lonsing[1,3] and Armin Biere[1]

[1] Institute for Formal Models and Verification,
Johannes Kepler University, Austria
`firstname.lastname@jku.at`

[2] Business Informatics Group
Vienna University of Technology, Austria

[3] Knowledge-Based Systems Group
Vienna University of Technology, Austria

**Abstract**

We present the tool qbf2epr which translates quantified Boolean formulas (QBF) to formulas in effectively propositional logic (EPR). The decision problem of QBF is the prototypical problem for PSPACE, whereas EPR is NEXPTIME-complete. Thus QBF is embedded in a formalism, which is potentially more succinct. The motivation for this work is twofold. On the one hand, our tool generates challenging benchmarks for EPR solvers. On the other hand, we are interested in how EPR solvers perform compared to QBF solvers and if there are techniques implemented in EPR solvers which would also be valuable in QBF solvers and vice versa. Furthermore, we provide an improved encoding of QBF in EPR based on dependency schemes which are a powerful concept in QBF solving.

## 1 Motivation

*Propositional logic* has proven itself to be an extremely valuable formalism for solving a wide range of reasoning problems of industrial scale. With its decision problem (SAT) being the prototypical problem for the complexity class NP, propositional logic serves not only as the host language for a wide range of application problems like planning and verification, but also is an enabling technology for large reasoning frameworks [20]. Building around this success story, research has been focused on related formalisms which have the same expressive power as SAT, but whose additional language features allow exponentially smaller problem encodings.

Two of such formalisms are *quantified Boolean formulas* (QBF) [17] and *effectively propositional logic* (EPR) [14]. While QBF extends the language of propositional logic with quantifiers over propositional variables, EPR is a syntactically restricted subset of first-order logic. As consequence, the majority of recent QBF solvers extend techniques found in modern SAT solvers, whereas EPR solvers are strongly inspired by first-order theorem provers. From a practical point of view, it would be interesting to know if there are benefits in transferring the reasoning techniques of one formalism to the other formalism. In order to directly compare QBF and EPR solvers, benchmarks are required which can be run by both kinds of solvers. One way of obtaining such benchmarks is embedding the "weaker" QBF in the "stronger" EPR. To this end, we present the tool qbf2epr which performs such an encoding. Therefore, we shortly revisit the basics of EPR and QBF in the next section, which are required for the embedding of QBF in EPR presented in Section 3. This embedding is implemented in the tool qbf2epr. Experiments

*This work was partially funded by the Vienna Science and Technology Fund (WWTF) under grant ICT10-018 and by the Austrian Science Fund (FWF) under NFN Grants S11408-N23 and S11409-N23 (RiSE).

performed with recent QBF and EPR solvers are presented in Section 4. For improving the embedding of QBF in EPR, we then consider *dependency schemes* in Section 5. Dependency schemes are a powerful concept in QBF solving, which provide a relaxed notion of quantifier dependencies. We conclude this paper with a discussion of future research directions.

## 2 Background

In the following, we introduce the concepts and terminology used in this paper necessary to describe the transformation of a QBF to an equisatisfiable EPR formula. In particular, we recap the formalisms EPR and QBF. We assume familiarity with propositional logic as well as with first-order logic.

**Effectively Propositional Logic (EPR).** The formulas of EPR (also known as Bernays-Schoenfinkel class) form a subset of first-order predicate logic (FOL) consisting of formulas with the structure

$$\exists X \forall Y.\, \rho \;\equiv\; \exists X \forall Y.\, \bigwedge_{i=0}^{n} \bigvee_{j=0}^{m_i} l_{ij}$$

where $X$ and $Y$ are disjoint sets of variables and $\rho$ is a function free first-order formula in conjunctive normal form over $X$ and $Y$. When transforming such a formula to Skolem normal form, the existentially quantified variables are simply replaced by new constants. If the universally quantified variables are grounded by all combinations of these constants, then the resulting formula is an exponentially larger representation of the EPR formula in propositional logic. In this paper, we use standard first-order semantics, which is specified over a domain $\mathcal{D}$ and an interpretation function $\iota : FOL \rightarrow \{\mathbb{T}, \mathbb{F}\}$. For our purposes, the domain of interest is binary. Several important use cases for EPR have been identified, including LTL bounded model checking [18] or reasoning with quantified bit vectors [9, 27].

**Quantified Boolean Formulas (QBF).** QBF extend propositional logic with quantifiers over propositional variables, i.e., a QBF may contain a subformula $\forall^q x.\psi$ or $\exists^q x.\psi$. The formula $\psi$ is again a QBF and is called *scope* of variable $x$. In this paper, we mainly consider QBF $\Pi.\psi$ in prenex conjunctive normal form (PCNF) with $\Pi = Q_1 X_1 \ldots Q_n X_n$, where the $X_i$ are disjoint sets of variables, $i$ denotes the quantification level of the variables in $X_i$, $Q_j \in \{\forall^q, \exists^q\}$, and $\psi$ is a propositional formula in conjunctive normal form. A formula in conjunctive normal form consists of a conjunction of clauses. A clause is a disjunction of literals. A literal is a variable or the negation of a variable. Note that we annotate the connectives, quantifiers, and truth constants with the superscript $^q$ in order to distinguish them from the corresponding symbols in EPR. A QBF $\forall^q x \Pi.\psi$ is satisfiable iff $\Pi.\psi[x \backslash \bot^q]$ and $\Pi.\psi[x \backslash \top^q]$ is true. Respectively, a QBF $\exists^q x \Pi.\psi$ is true iff $\Pi.\psi[x \backslash \bot^q]$ or $\Pi.\psi[s \backslash \top^q]$ is true. QBF find their application for instance in various verification scenarios [3].

## 3 From QBF to EPR

In this section, we discuss the translation of QBFs in prenex conjunctive normal form to EPR formulas. This translation is based on the results presented in [2, 21]. Benedetti [2] introduced the Skolemization-based QBF solver sKizzo which rewrites QBF to a first-order formula, before reducing it to a propositional formula. Piskac et al. illustrate the encoding of quantified formulas with equality over a finite domain to EPR [21] which we specialize to standard QBF. The translation is straightforward, but optimized with respect to specific QBF properties.

## 3.1 Transformation of a QBF to a First-Order Formula

In this first step, we reformulate the QBF problem as an instance of first-order predicate logic as follows.

**Definition 1.** *The embedding $[\![]\!]_p : QBF \to FOL$ with respect to the unary predicate symbol p is given by*

$$\begin{aligned}
[\![\exists^q x.\phi]\!]_p &= \exists x.[\![\phi]\!]_p & \qquad [\![\forall^q x.\phi]\!]_p &= \forall x.[\![\phi]\!]_p \\
[\![\phi \vee^q \psi]\!]_p &= [\![\phi]\!]_p \vee [\![\psi]\!]_p & [\![\phi \wedge^q \psi]\!]_p &= [\![\phi]\!]_p \wedge [\![\psi]\!]_p \\
[\![x]\!]_p &= p(x) & [\![\neg^q x]\!]_p &= \neg p(x) \\
[\![\top^q]\!]_p &= p(true) & [\![\bot^q]\!]_p &= p(false)
\end{aligned}$$

The embedding wraps the predicate $p$ around the variables and QBF truth constants $\top^q$ and $\bot^q$ are mapped to the dedicated function symbols *true* and *false*. Whereas in predicate logic variables and in consequence the quantifiers operate on the term level, the situation is different in QBF. Here the variables incarnate predicates. To lift the variables to term level, we introduce a predicate $p$ of arity one, for which it holds that $\iota(p(true)) = \mathbb{T}$ and $\iota(p(false)) = \mathbb{F}$.

**Lemma 1.** *Let $\phi$ be a QBF and let p be a unary predicate symbol. Then $\phi$ is satisfiable iff the first-order formula $([\![\phi]\!]_p \wedge p(true) \wedge \neg p(false))$ is satisfiable.*

Lemma 1 can be easily shown by induction over the formula size. It describes the embedding of arbitrary structured QBF to FOL. In the following, we consider QBFs in prenex conjunctive normal form only. This is no severe restriction, because the transformation to PCNF is polynomial and most QBF benchmarks are only available in PCNF anyhow. We impose no restriction on the number of quantifier alternations. If a QBF $\phi$ has a prefix with $n$ quantifier alternations, then the FOL formula $[\![\phi]\!]_p$ obviously has $n$ quantifier alternations as well.

## 3.2 Elimination of Existential Quantifiers

In order to obtain a FOL formula with the prefix structure $\exists X \forall Y$, we substitute existentially quantified variables which are in the scope of universally quantified variables $a_1, \ldots, a_m$ by a Boolean function with $a_1, \ldots, a_m$ as arguments. This technique of symbolic representation of quantifier dependencies is know as Skolemization [24]. In automated theorem proving, Skolemization is used to eliminate one type of quantifiers in a satisfiability preserving manner.

**Definition 2.** *Let $\chi = \exists x_1 \ldots \exists x_n \forall a_1 \ldots \forall a_m \exists y \Pi.\rho$ be a FOL formula in PCNF. Then the quantifier $\exists y$ can be eliminated as follows*

$$\exists x_1 \ldots \exists x_n \forall a_1 \ldots \forall a_m \exists y \Pi.\rho \rightsquigarrow \exists x_1 \ldots \exists x_n \forall a_1 \ldots \forall a_m \Pi.\rho[y \backslash f_y(a_1, \ldots, a_m)]$$

*where $f_y$ is a function symbol not occurring in $\chi$. The function $\mathsf{sk}(\chi)$ applies this substitution on FOL formula $\chi$ until fixpoint.*

Obviously, in Definition 2 an existential variable is replaced by a function with all universal variables as arguments which have a lower quantification level. In Section 5, we will provide an optimization of this substitution in order to obtain functions with less parameters.

**Lemma 2.** *Let $\rho$ be a FOL formula in prenex conjunctive normal form and let $\rho' = \mathsf{sk}(\rho)$. Then it holds that (i) the prefix of $\rho'$ contains at most one quantifier alternation, (ii) if $\rho'$ has one quantifier alternation, then it has the structure $\exists X \forall Y$, (iii) $\rho'$ is satisfiable iff $\rho$ is satisfiable.*

Points (i) and (ii) of Lemma 2 follow from the construction of $\mathsf{sk}(.)$, and point (iii) is an application of Skolemization [24]. With the introduction of function symbols, we move further away not only from the language of QBF but also from EPR, although we have now the required structure of the quantifier prefix. To obtain an EPR formula, the function symbols must be eliminated.

## 3.3 Removal of Function Symbols

In the last step, we have introduced function symbols which allowed us to symbolically represent quantifier dependencies and to reduce the prefix to the required structure having only one alternation. Since the language of EPR does not allow function symbols, these have to be removed from the formula again.

**Definition 3.** *The function $\mathsf{func}(\rho)$ applies the following rewriting rule*

$$p(f(a_1, \ldots, a_m)) \rightsquigarrow p_f(a_1, \ldots, a_m)$$

*on FOL formula $\rho$ until fixpoint where $p_f$ is a predicate and $f$ is a function symbol.*

We are interested in interpretations of the functions over a two valued domain, hence function symbols can also be seen as predicates. Therefore we can remove the enclosing predicate symbol for such functions.

**Lemma 3.** *Over a two valued domain, the rewriting function $\mathsf{func}(\rho)$ preserves satisfiability.*

Finally, we have all the building blocks required to transform a QBF to an EPR formula which manifests in the following proposition.

**Proposition 1.** *Let $\phi$ be a formula in prenex conjunctive normal form and let $p$ be a symbol not occurring in $\phi$. Then $(\mathsf{func}(\mathsf{sk}([\![\phi]\!]_p)) \wedge p_{true} \wedge \neg p_{false})$ is an EPR formula which is equisatisfiable to $\phi$.*

Proposition 1 follows from Lemma 1, Lemma 2, and Lemma 3. The following example illustrates the translation of a QBF to an EPR formula.

**Example 1.** *Given the QBF $\phi = \exists^q a \exists^q b \forall^q x \forall^q y \exists^q c \exists^q d.(a \vee^q x \vee^q c) \wedge^q (a \vee^q b) \wedge^q (b \vee^q y \vee^q d)$ the following three steps have to be performed.*

1. Transformation to FOL.
   *We embbed $\phi$ in first-order logic w.r.t. predicate $p$ and obtain*

   $$[\![\phi]\!]_p = \exists a \exists b \forall x \forall y \exists c \exists d.(p(a) \vee p(x) \vee p(c)) \wedge (p(a) \vee p(b)) \wedge (p(b) \vee p(y) \vee p(d)).$$

2. Elimination of Existential Quantifiers.
   *In order to obtain the required prefix structure, we apply Skolemination and eliminate all existential quantifiers not occurring in the first quantifier block, i.e., $\exists c$ and $\exists d$. Then*

   $$\mathsf{sk}([\![\phi]\!]_p) = \exists a \exists b \forall x \forall y.(p(a) \vee p(x) \vee p(f_c(x, y))) \wedge (p(a) \vee p(b) \wedge (p(b)) \vee p(y) \vee p(f_d(x, y))).$$

3. Elimination of Function Symbols.
   *Finally, we lift the function symbols to predicats resulting in the formula*

   $$\mathsf{func}(\mathsf{sk}([\![\phi]\!]_p)) = \exists a \exists b \forall x \forall y.(p(a) \vee p(x) \vee f_c(x, y)) \wedge (p(a) \vee p(b) \wedge (p(b)) \vee p(y) \vee f_d(x, y)).$$

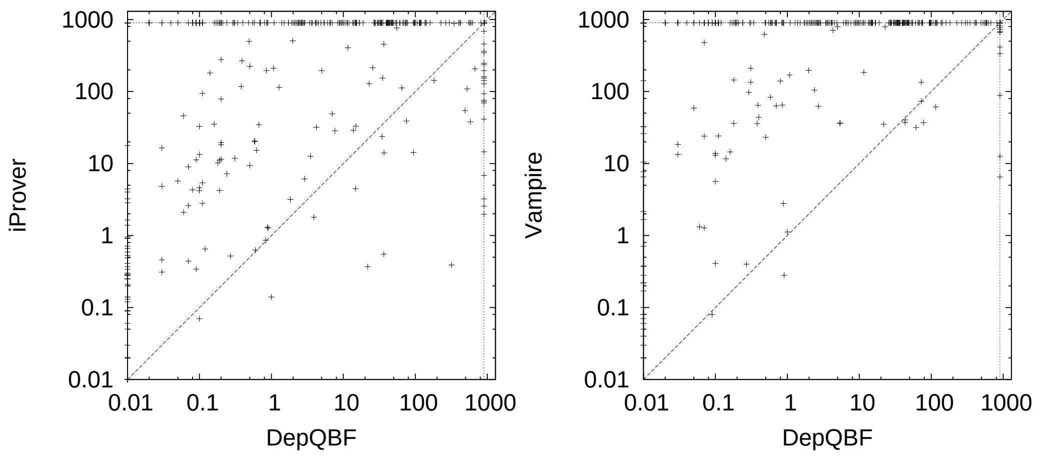


Figure 1: Comparison DepQBFwith iProver and Vampire.

## 4 Evaluation

The embedding of QBF in EPR as described in the previous section is implemented in the tool qbf2epr [1], with input format QDIMACS, the standard format for QBFs in prenex conjunctive normal form [10]. The produced formulas are formulated in the TPTP language [25]. We investigated how a state-of-the-art EPR solver performs on the QBF benchmark set of the evaluation 2010 [19]. As EPR solver we used iProver [12], which won the EPR devision of the CASC competition [26] in the last years. As a reference QBF solver, we ran DepQBF [16]. DepQBF is a search-based solver relying on the QDPLL algorithm for QBF [7]. QDPLL is a QBF-specific variant of the DPLL algorithm for propositional logic. Given a QBF in QDIMACS format, DepQBF processes the formula without any modifications whereas iProver operates on the EPR embedding generated by qbf2epr. All considered QBFs are in PCNF already, hence there was no need for explicit clausification in iProver in our experiments. Table 1 summarizes our main results.[1] The benchmark set contains 568 formulas.

As probably expected, the QBF solver DepQBF outperforms iProver in terms of the number of solved instances as well as in terms of time and memory requirements. DepQBF solved more than twice as many instances in half of average time spent by iProver on the instances translated by qbf2epr (cf. left subfigure of Figure 1). The translation to EPR failed on four out of 568 formulas due to limited time or memory. This does not influence the overall picture as none of these four formulas were solved by DepQBF. On average, translation time spent by qbf2epr was 13.33 seconds (in parentheses next to 673.50, the average run time of iProver). Average memory usage of DepQBF is lower by a factor of 40. Note that iProver ran out of memory on 237 instances, which also contributes to the median time of 900 seconds, as we treat these situations like running out of time.

If we focus on instances which were solved by both DepQBF and iProver, then again instances were solved faster and with less memory by DepQBF in their native QBF encoding. The performance of iProver is closest to DepQBF on unsatisfiable instances, where iProver spends 50% more time on average. This observation is also related to 21 instances which were solved

[1]Setup: 64-bit Ubuntu Linux 9.04, Intel®Q9550 2.83 GHz with 900 seconds / 7 GB total time and memory limit. Exceeding the memory limit is counted as a time out. We used iProver version 0.8.1 and an internal version of DepQBF. Binaries of all tools and log-files are available online [1].

| | *Solved* | *SAT* | *UNSAT* | *Avg. Time* | *Med. Time* | *Avg. Mem.* | *Med. Mem.* |
|---|---|---|---|---|---|---|---|
| DepQBF | 372 | 166 | 206 | 334.60 | 29.92 | 93.3674 | 22.2 |
| iProver | 155 | 51 | 104 | 673.50 (13.33) | 900 | 3924.48 | 3673.0 |
| Only formulas solved by both iProver and DepQBF | | | | | | | |
| DepQBF | 134 | 44 | 90 | 28.08 | 0.09 | 19.056 | 2.55 |
| iProver | | | | 54.87 | 3.62 | 432.5 | 85.60 |
| Only satisfiable formulas solved by both iProver and DepQBF | | | | | | | |
| DepQBF | 44 | 44 | – | 2.60 | < 0.01 | 4.02 | < 0.01 |
| iProver | | | | 36.94 | 0.48 | 120.548 | 29.10 |
| Only unsatisfiable formulas solved by both iProver and DepQBF | | | | | | | |
| DepQBF | 90 | – | 90 | 40.54 | 0.18 | 26.4067 | 7.35 |
| iProver | | | | 63.63 | 5.54 | 585.01 | 167.15 |
| 21 formulas solved by iProver but not by DepQBF | | | | | | | |
| iProver | 21 | 7 | 14 | 166.43 | 127.89 | 1505.07 | 616.9 |

Table 1: Performance comparison of DepQBF and iProver.

by iProver but *not* by DepQBF. From those 21 instances, 14 are unsatisfiable. Successful QBF preprocessing techniques like blocked clause elimination [5] showed only little improvements on the performance of the EPR solver.

Apart from the QDPLL-based solver DepQBF, we considered the QBF solver Quantor [4], which is based on quantifier elimination. Quantor solved 203 instances (99 satisfiable, 104 unsatisfiable) in 590.15 seconds average time. Although Quantor solves fewer instances than DepQBF, the results indicate that QDPLL and quantifier elimination, the two major approaches for QBF solving, perform better on QBF than iProver does on translated instances.

Note that iProver solved 37 unsatisfiable instances which were *not* solved by Quantor. A closer look at the set of formulas exclusively solved by iProver (and by none of the two QBF solvers) seems to suggest that techniques applied in iProver are particularly beneficial for solving unsatisfiable QBFs. Many of these exclusively solved formulas encode problems of black box bounded model checking (BMC) [11] (family `biu*` in the benchmark set). Solving BMC encodings in QBF and EPR is in general considered to be challenging [3, 9].

Additionally, we were interested how a first-order theorem prover handles the translated QBFs. We therefore ran the same experiment with the state-of-the-art theorem prover Vampire [22]. Vampire solved 65 formulas (24 satisfiable, 41 unsatisfiable). A comparison to DepQBF is shown in the right part of Figure 1. Again several instances of black box bounded model checking are included in the set of solved formulas.

## 5 From QBF to EPR with Dependencies

The embedding of QBF in EPR as introduced in Section 3 always included *all* universal variables of lower quantification level in the Skolem function of an existential variable, even if they were independent of each other. In order to reduce number of arguments the Skolem functions, we clarify the notion of (in)dependence based on a QBF specific concept called *dependency scheme* and evaluate the impact of this optimization on the runtime of the EPR solver.

## 5.1 QBF Dependency Schemes

Most state-of-the-art QBF solvers use prenex conjunctive normal form as introduced in Section 2 as input format. On the one hand, PCNF supports sophisticated reasoning techniques and allows for highly optimized data structures, but on the other hand structural information which might be valuable for the solving process is blurred by the transformation to PCNF. For example, in general it is not given that a formula consists of a quantifier prefix and the propositional matrix. To obtain the required structure, *prenexing* has to be performed which refers to the satisfiability preserving transformation of shifting all quantifiers outside the formula.

Whereas in the original formula the quantifiers are arranged in a tree established by the nesting of the scopes, prenexing flattens this tree into a linear list. So a much stronger order is imposed on the variables, which is restrictive for the solving process. Since prenexing is not deterministic, multiple prenexing strategies are often applicable. It has been shown that the selection of the wrong strategy adversely influences the solving time [8]. Therefore, solvers are strongly dependent on the normalform transformation tool, if the prefix is simply processed from left to right.

To overcome this restriction, dependency schemes [23] have been introduced which allow for relaxing the prefix ordering without changing the truth value of a formula. A dependency scheme $D$ is a binary relation over the variables of a QBF expressing (in)dependence of two variables. A variable $y$ depends on variable $x$ iff $(x, y) \in D$. Consider the QBF $\phi = \forall^q x \exists^q y.\psi$ with $(x, y) \notin D$. Then $\phi$ is equivalent to $\exists^q y \forall^q x.\psi$, i.e., if two variables are independent, then their quantifiers may be swapped. The prenex directly implies a trivial dependency scheme $P$ where $(x, y) \in P$ iff variable $x$ occurs before variable $y$ in the prefix. As in the case of $P$, dependency schemes may contain spurious dependencies. These spurious dependencies may be eliminated by a stronger dependency scheme, but dependencies crucial for the truth value of a formula may never be omitted. Less spurious dependencies contained in a dependency scheme imply more freedom for variable selection. It can be shown that there exists one unique optimal dependency scheme, but its computation is in PSPACE. Since the computation of the optimal dependency scheme is as hard as solving a QBF, it is not practically feasible. Therefore, non-optimal, but nevertheless powerful dependency schemes are used for QBF solving. For a detailed survey on theory and practice of dependency schemes in QBF solver, we kindly refer to [15].

## 5.2 Translation to EPR with Dependency Schemes

In the following, we extend Definition 2 with a dependency scheme in order to reduce the number of arguments of the Skolem functions.

**Definition 4.** *Let $\phi$ be a QBF in PCNF with a dependency scheme $D$. Further, let $\chi = [\![\phi]\!]_p = \exists X \forall A \exists y \Pi.\rho$ be an embedding of $\phi$ in FOL, where $X$ and $A$ are sets of variables (and $\Pi$ the rest of the quantifier prefix). Then the quantifier $\exists y$ can be eliminated as follows*

$$\chi \rightsquigarrow \exists X \forall A \Pi.\rho[y \backslash f_y(a_1, \ldots, a_k)]$$

*where $(a_i, y) \in D$, $|\{a_i | (a_i, y) \in D\}| = k$, and $f_y$ is a function symbol not occurring in $\chi$. The function $\mathsf{sk}^{\mathsf{D}}(\chi)$ applies this substitution on FOL formula $\chi$ until fixpoint.*

Note that Definition 4 does not refer to any specific dependency scheme. If we would use the dependency scheme induced by the quantifier prefix, then we would obtain the same translations as with Definition 2.

**Lemma 4.** *Let $\rho$ be a FOL formula in prenex conjunctive normal form obtained from a QBF $\phi$ with dependency scheme $D$. Further, let $\rho' = \mathsf{sk}^{\mathsf{D}}(\rho)$. Then it holds that (i) the prefix of $\rho'$ contains at most one quantifier alternation, (ii) if $\rho'$ has one quantifier alternation, then it has the structure $\exists X \forall Y$, (iii) $\rho'$ is satisfiable iff $\rho$ is satisfiable.*

Whereas (i) and (ii) directly follow from the construction of $\mathsf{sk}^{\mathsf{D}}()$, (iii) holds because it can be shown that two QBFs in PCNF are equivalent if they have the same matrix and the same quantifiers, and the quantifier ordering of both obey the same dependency scheme.

**Example 2.** *The QBF $\exists^q a \exists^q b \forall^q x \forall^q y \exists^q c \exists^q d.(a \vee^q x \vee^q c) \wedge^q (a \vee^q b) \wedge^q (b \vee^q y \vee^q d)$ with dependency scheme $D = \{(a,x),(x,c),(b,y),(y,d)\}$ is equisatisfiable to the EPR formula*

$$\exists a \exists b \forall x \forall y.(p(a) \vee p(x) \vee f_c(x)) \wedge (p(a) \vee p(b)) \wedge (p(b) \vee p(y) \vee f_d(y)).$$

## 5.3 Implementation and Evaluation

The QBF solver DepQBF [16] which we already applied for the evaluation in Section 4 uses the *standard dependency schema* [23] in a DPLL-based decision procedure and provides a very efficient implementation for its calculation. The standard dependency scheme is calculated by analyzing the structure of a formula in terms of connections between variables in sequences of clauses. The dependency scheme of Example 2 is a standard dependency scheme. There are dependencies between $a$ and $x$ as well as between $x$ and $c$, because these variables occur in the same clause. The same holds for $b$ and $y$ and $y$ and $d$. However, there is no dependency between $d$ and $x$ as well as $c$ and $y$. In consequence, we may replace $c$ by $f_c(x)$ and $d$ by $f_d(y)$. A formal description of the standard dependency scheme can be found in [23].

We extended DepQBF in such a way that it provides a dependency service, i.e., it can be called with a formula as argument, and then compute pairs of dependent variables. qbf2epr uses this service to obtain the universal variables on which a given existential variable depends. Then only these variables are included in the Skolem function.

With the improved translation, we performed the same evaluation as before using iProver as EPR solver. With the dependency scheme enabled, 104 unsatisfiable formulas as well as 51 satisfiable formulas were solved, e.g. exactly the same number as before without dependencies. However, the usage of dependency schemes reduced the over all runtime by more than 1000 seconds. Details are available at [1].

## 6 Conclusion and Future Work

We showed how to translate QBF to EPR resulting in challenging benchmarks for testing and evaluating EPR solver. The tool qbf2epr is available at [1]. EPR as well as QBF are promising formalisms for the encoding of a multitude of verification problems [9, 18, 20, 27] for which probably no succinct encoding in SAT can be found. Although both are closely related to SAT, QBF and EPR solving approaches differ profoundly. In first-order theorem proving for example, instantiation-based approaches result quite naturally in a decision procedure for EPR [13] which might also be interesting for QBF. In our experiments we observed that there exist formulas in the QBF standard benchmark set which can be solved by the EPR solver iProver but not by specialized QBF solvers.

This points out that there might be techniques used in EPR solving which might also be valuable for QBF solvers and confirms observations of [27] where a similar behavior for quantified bit vector formulas was experienced. Therefore, more experiments have to be conducted and the implemented inference techniques have to be compared rigorously. QBF solvers quite substantially outperform EPR solvers on the considered QBF benchmarks, and thus it would be interesting to investigate if and how EPR solving can benefit from QBF solving techniques also on other instances. Besides the additional benchmarks, qbf2epr offers an additional benefit to the developers of EPR solvers. Now QBF specific development tools like fuzzers and delta-debuggers [6] can be directly used for the development of EPR solvers.

We further showed, how the presented embedding is extended to take independencies between variables into account. First, we considered only the order of the variables imposed by the prefix. In an improved variant of the embedding, we use dependency schemes for reducing the number of arguments of the Skolem functions.

However, the comparison between the QBF and EPR solvers was not fair in the following sense. EPR allows for a potentially more succinct encoding of application problems than QBF due to the richer language. Nevertheless, we provided a direct translation of the QBF encoding to the EPR solver, which does not benefit from EPR language features. In future work, it would be therefore interesting to investigate if more sophisicated translations of the QBFs are possible. For a fair evaluation, application problems shall be directly encoded in QBF and EPR allowing a direct comparison of the solvers.

*Acknowledgements.* The authors would like to thank Robert Aistleitner and Gregor Dorfbauer for implementing the original version of the presented tool as part of a student project.

## References

[1] qbf2epr Project Page. Website. `http://fmv.jku.at/qbf2epr/`.

[2] M. Benedetti. sKizzo: a QBF Decision Procedure based on Propositional Skolemization and Symbolic Reasoning. Tech.Rep. 04-11-03, ITC-irst, 2004.

[3] M. Benedetti and H. Mangassarian. Qbf-based formal verification: Experience and perspectives. *JSAT*, 5(1-4):133–191, 2008.

[4] A. Biere. Resolve and Expand. In *SAT'04*, pages 59–70, 2004.

[5] A. Biere, F. Lonsing, and M. Seidl. Blocked Clause Elimination for QBF. In *CADE'11*, volume 6803 of *LNCS*, pages 101–115. Springer, 2011.

[6] R. Brummayer, F. Lonsing, and A. Biere. Automated Testing and Debugging of SAT and QBF Solvers. In *Proc. of SAT 2010*, volume 6175 of *LNCS*, pages 44–57. Springer, 2010.

[7] M. Cadoli, A. Giovanardi, and M. Schaerf. An Algorithm to Evaluate Quantified Boolean Formulae. In *AAAI/IAAI'98*, pages 262–267, 1998.

[8] U. Egly, M. Seidl, H. Tompits, S. Woltran, and M. Zolda. Comparing Different Prenexing Strategies for Quantified Boolean Formulas. In *Proc. of SAT 2003*, volume 2919 of *LNCS*, pages 214–228. Springer, 2004.

[9] M. Emmer, Z. Khasidashvili, K. Korovin, and A. Voronkov. Encoding industrial hardware verification problems into effectively propositional logic. In *FMCAD'10*, pages 137–144. IEEE, 2010.

[10] E. Giunchiglia, M. Narizzano, and A. Tacchella. Quantified Boolean Formulas satisfiability library (QBFLIB), 2001. `www.qbflib.org`.

[11] M. Herbstritt and B. Becker. On Combining 01X-Logic and QBF. In *EUROCAST'07*, volume 4739 of *LNCS*, pages 531–538. Springer, 2007.

[12] K. Korovin. iProver - An Instantiation-Based Theorem Prover for First-Order Logic (System Description). In *IJCAR'08*, pages 292–298, 2008.

[13] K. Korovin. Instantiation-based automated reasoning: From theory to practice. In *CADE'09*, volume 5663 of *LNCS*, pages 163–166. Springer, 2009.

[14] H.R. Lewis. Complexity results for classes of quantificational formulas. *Journal of Computer and System Sciences*, 21(3):317–353, 1980.

[15] F. Lonsing. *Dependency Schemes and Search-Based QBF Solving: Theory and Practice*. PhD thesis, JKU Linz, 2012.

[16] F. Lonsing and A. Biere. DepQBF: A Dependency-Aware QBF Solver. *JSAT*, 7(2-3):71–76, 2010.

[17] A.R. Meyer and L.J. Stockmeyer. The equivalence problem for regular expressions with squaring requires exponential space. In *SWAT'72*, pages 125–129. IEEE, 1972.

[18] J.A. Navarro Pérez and A. Voronkov. Encodings of Bounded LTL Model Checking in Effectively Propositional Logic. In *CADE'07*, volume 4603 of *LNCS*, pages 346–361. Springer, 2007.

[19] C. Peschiera, L. Pulina, A. Tacchella, U. Bubeck, O. Kullmann, and I. Lynce. The Seventh QBF Solvers Evaluation (QBFEVAL'10). In *SAT*, volume 6175 of *LNCS*, pages 237–250. Springer, 2010.

[20] M. R. Prasad, A. Biere, and A. Gupta. A survey of recent advances in SAT-based formal verification. *STTT*, 7(2):156–173, 2005.

[21] R. Piskac and L. de Moura and N. Bjørner. Deciding Effectively Propositional Logic with Equality. Technical Report: MSR-TR-2008-181.

[22] A. Riazanov and A. Voronkov. The design and implementation of vampire. *AI Commun.*, 15(2-3):91–110, 2002.

[23] Marko Samer and Stefan Szeider. Backdoor sets of quantified boolean formulas. *J. Autom. Reasoning*, 42(1):77–97, 2009.

[24] Th. Skolem. *Logisch-kombinatorische Untersuchungen über die Erfüllbarkeit und Beweisbarkeit mathematischen Sätze nebst einem Theoreme über dichte Mengen.* Translation in: From Frege to Gödel, van Heijenoort, Harvard Univ. Press, 1971.

[25] G. Sutcliffe. The TPTP Problem Library and Associated Infrastructure: The FOF and CNF Parts, v3.5.0. *Journal of Automated Reasoning*, 43(4):337–362, 2009.

[26] G. Sutcliffe and C. Suttner. The State of CASC. *AI Comm.*, 19(1):35–48, 2006.

[27] C.M. Wintersteiger, Y. Hamadi, and L. de Moura. Efficiently solving quantified bit-vector formulas. In *FMCAD'10*, pages 239–246. IEEE, 2010.

# MetTeL2: Towards a Tableau Prover Generation Platform

Dmitry Tishkovsky, Renate A. Schmidt, Mohammad Khodadadi*
The University of Manchester, UK

**Abstract**

This paper introduces MetTeL2, a tableau prover generator producing Java code from the specification of a logical syntax and a tableau calculus. It is intended to provide an easy to use system for non-technical users and allow technical users to extend the generated implementations.

## 1 Introduction

Building a platform for automatically generating provers from the definition of a logic is a challenging task. As the problem of generating a deduction calculus from the definition of a logic is highly undecidable, the best that we can hope for is technology for solving the problem for certain restricted cases. The tableau method was introduced in the 1950s by Beth and Hintikka [8, 16]. With origins in the work of Gentzen in the 1930s [13] and thoroughly studied by Smullyan in the 1960s [29], it has become one of the most popular deduction approaches in automated reasoning. Tableau methods in numerous forms exist for various logics and many implementations of tableau provers are available.

Based on the collective experience in the area our recent research has been concerned with trying to develop a framework for synthesising tableau calculi from the specification of a logic. The tableau synthesis framework introduced in [27] effectively describes a class of logics for which tableau calculus synthesis can be done automatically. This class includes many modal, description, intuitionistic and hybrid logics. Our long-term goal is to synthesise not only tableau calculi, but also implementations of the tableau calculi as tableau provers.

As a step towards this goal we have implemented a tool, called MetTeL2, for automatically generating code of a tableau prover from user-defined specifications of a syntax and a set of tableau rules for a logical theory. The syntax and tableau rule specification languages of MetTeL2 are designed to be as simple as possible for the user and to be as close as possible to the traditional notation used in logic and automated reasoning textbooks. At the moment the syntax specification language is limited to multi-sorted propositional languages with finitary connectives. The tableau calculus specification language covers different types of tableau calculi that fit the traditional representation of tableau rules of the form $X_0/X_1 \mid \cdots \mid X_m$, where the $X_i$ denote finite sets of expressions of the given logical theory. $X_0$ is a set of premises and $\{X_1, \ldots, X_m\}$ is a finite set of branches of the rule. Many labelled semantic tableau calculi for modal, description, hybrid and superintuitionistic logics belong to this paradigm.

MetTeL2 is complementary to the mentioned tableau synthesis framework [27]. The framework provides a theoretical foundation for sound, complete and terminating implementations of tableau procedures for a wide class of logics with first-order representable semantics and, in particular, for many logics which can be specified in MetTeL2. The scope of MetTeL2 extends however that of tableau calculi derived in the framework and is not limited to semantic or labelled tableau calculi.

*This research is supported by EPSRC research grant EP/H043748/1.

METTEL$^2$ is the successor of the METTEL system [31, 1]. METTEL is a tableau prover for a large class of propositional modal-type logics, including various traditional modal logics, dynamic modal logics, description logics, hybrid logics, intuitionistic logic and logics of metrics and topology. It does already allow users to specify their own tableau calculi and then use METTEL as a prover for the specified calculus. Though flexible, the specification language of METTEL is based on a *fixed* set of logical operators common to the mentioned logics. This means there is no facility in the specification language to allow the user to define their own set of logical operators unrelated to operators of modal-type logics.

The functionality of METTEL$^2$ considerably extends that of the METTEL prover. METTEL$^2$ generates JAVA code for a tableau prover to parse problems in the user-defined syntax to solve satisfiability problems. In order to come closer to the vision of a powerful prover generation tool, METTEL$^2$ is equipped with a flexible specification language for users to define their logic or logical theory with syntactic constructs as they see fit. Thus no logical operators are predefined in METTEL$^2$.

Compared with the previous METTEL system, the tableau reasoning core of METTEL$^2$ has been completely reimplemented and several new features have been added, the most important being: dynamic backtracking [14] and conflict-directed backjumping [11, 25], and ordered forward and backward rewriting for operators declared to be equality and equivalence operators. There is support for different search strategies. The tableau rule specification language in METTEL$^2$ now allows the specification of rule application priorities thus providing a flexible and simple tool for defining rule selection strategies. To our knowledge, METTEL$^2$ is the first system with full support of these techniques for *arbitrary* logical syntax.

The aim of the current implementation is to provide an easy to use prover generator with basic specification languages without sophisticated meta-programming features that might overwhelm non-technical users. For technical users, the generated code consists of a thoroughly designed hierarchy of public JAVA classes and interfaces that can be extended and integrated with other systems.

The paper is structured as follows. We introduce the syntax and tableau specification languages of METTEL$^2$ in Sections 2 and 3. Section 4 describes a common scenario of using METTEL$^2$ as a prover generator. Details on implementation, prover generation, integrated optimisations, and features for controlling derivations are given in Sections 5, 6, and 7. We show how generated provers can be run and discuss their output in Section 8. The scope of METTEL$^2$ and further features are illustrated in Section 9. In Section 10 we discuss possible applications of METTEL$^2$ and our experience of using the system. Sections 11 and 12 give an overview of related work and further directions for development of the system.

## 2 Language specification

The language of METTEL$^2$ for specifying the syntax of a logical theory, is in line with the many-sorted object specification language of the tableau synthesis framework defined in [27]. We give a simple 'non-logical' example for describing and comparing lists to illustrate how the language of a logical theory can be defined in METTEL$^2$.

```
specification lists;
syntax lists{
    sort formula, element, list;
    list empty = '<>' | composite = '<' element list '>';
    formula elementInequality = '[' element '!=' element ']';
```

```
        formula listInequality = '{' list '!=' list '}';
    }
```

The first line starting with the keyword **specification** defines lists to be the name of the user-defined logical language. The **syntax** lists{...} block consists of the declaration of the sorts and definitions of logical operators in simplified BNF notation. Here, the specification is declared to have three sorts. For the sort element no operators are defined. This means that all element expressions are atomic. The second line defines two operators for the sort list: a nullary operator <> (to be used for the empty list) and a binary operator <..> (used to inductively define non-empty lists). composite is the name of the operator <..>, which could have been omitted. The next two lines define how expressions of sort formula can be formed. For example, the line formula listInequality = '{'list '!='list '}'; defines an inequality operator on lists, while the previous line defines an inequality operator on elements (note the difference in notation via the brackets). This means formulae can be two types of inequality expressions. The first mentioned sort in a declaration, in our case formula, is the *main sort* of the defined language. Several declarations of connectives for the same sort are equivalent to a joint statement which is composed from these declarations by means of the operator |. For example, the two statements for formula are equivalent to the following statement:

```
formula elementInequality = '[' element '!=' element ']' | listInequality = '{' list '!=' list '}';
```

Another example we consider is three-valued Łukasiewicz logic Ł$_3$. The syntax of the logic includes the standard connectives $\bot$, $\top$, $\neg$, $\vee$, $\wedge$, and $\rightarrow$. Truth values are considered over three-element Łukasiewicz algebra over the set $\{0, \frac{1}{2}, 1\}$. The elements of the algebra are naturally ordered. The algebra operations correspond to the connectives of the logic and are defined for any elements $a$ and $b$ from the algebra as follows [20, 17, 30].

$$\bot \overset{\text{def}}{=} 0 \qquad a \wedge b \overset{\text{def}}{=} \min\{a, b\} \qquad \neg a \overset{\text{def}}{=} 1 - a$$
$$\top \overset{\text{def}}{=} 1 \qquad a \vee b \overset{\text{def}}{=} \max\{a, b\} \qquad a \rightarrow b \overset{\text{def}}{=} \min\{1, 1 - a + b\}$$

The syntax specification for the logic Ł$_3$ with respect to the three-valued Łukasiewicz algebra is the following:

```
    specification Lukasiewicz3;
    syntax Lukasiewicz3{
        sort valuation, formula;
        valuation true = 'T' formula | unknown = 'U' formula | false = 'F' formula;
        formula true = 'true' | false = 'false';
        formula negation = '~' formula;
        formula conjunction = formula '&' formula;
        formula disjunction = formula '|' formula;
        formula implication = formula '->' formula;
    }
```

Thus, in the specification we denote truth value 0 as `F` ('false'), truth value $\frac{1}{2}$ as `U` ('unknown'), and truth value 1 as `T` ('true'). In order to indicate that a formula $\phi$ has truth value $a$ in the Łukasiewicz algebra we prepend $\phi$ with $a$, that is **`U (p -> q)`** means that the formula **`p -> q`** has 'unknown' truth value.

## 3 Tableau calculus specification

The tableau rule specification language of MetTeL2 is loosely based on the tableau rule specification language of MetTeL, but extends it in significant ways. The premises and conclusions of a rule are separated by / and each rule is terminated by $;. Branching rules can have more than two sets of conclusions and are separated by $| symbols. Premises and conclusions are expressions in the user-defined logical language. Additionally, the user can annotate a rule with a *priority value*. The default priority value of any rule with unspecified priority is 0. Roughly speaking (see Section 7), smaller priority values imply a rule is applied earlier.

Turning back to the examples of the previous section, tableau rules for list comparison might be defined as follows.

```
[a != a] / priority 0$;
{L != L} / priority 0$;
{<a L0> != <b L1>} / [a != b] $| {L0 != L1} priority 2$;
```

The first two rules are closure rules since the right hand sides of / are empty. They reflect that inequality is irreflexive. The last rule is a branching rule. As the parsing of rule specifications is context-sensitive the various identifiers (a, L, L0, etc) are recognised as symbols of the appropriate sorts. Thus *sorts* of identifiers are distinguished by their contextual position within the rule and not their symbolic representation.

A tableau calculus for the three-valued Łukasiewicz logic is given in [17]. The following is its specification in MetTeL$^2$ syntax.

```
T false /   priority 0$;                       U false /   priority 0$;
U true /   priority 0$;                        F true /   priority 0$;
T P  F P /   priority 0$;                      T P  U P /   priority 0$;
U P  F P /   priority 0$;                      U P  F P /   priority 0$;
T ~P / F P  priority 1$;    U ~P / U P  priority 1$;    F ~P / T P  priority 1$;
T (P & Q) /  T P  T Q  priority 2$;            F (P & Q) /  F P  $|  F Q  priority 1$;
U (P & Q) /  T P  U Q  $|  U P  T Q  $|  U P  U Q  priority 3$;
T (P | Q) /  T P $| T Q  priority 2$;          F (P | Q) /  F P  F Q  priority 1$;
U (P | Q) /  F P  U Q  $|  U P  F Q  $|  U P  U Q  priority 3$;
F (P -> Q) /  T P  F Q  priority 1$;           U (P -> Q) / U P  F Q  $|  T P  U Q  priority 2$;
T (P -> Q) /  T Q  $|  F P  $|  U P  U Q  priority 3$;
```

The first eight rules are closure rules which detect contradictions between truth values of formulae. All of them have the highest priority (priority value 0). The rest of the rules reflect the truth tables for the connectives of the logic (in the Łukasiewicz algebra) and are given priorities proportional to their branching factors. This means that the higher the branching factor of a rule, the less often the rule is applied in tableau derivations.

## 4 Using MetTeL$^2$

The binary version of MetTeL$^2$ is distributed as a **jar**-file and requires Java Runtime Environment, Version 1.6.0 or later. MetTeL$^2$ can be called from the command line as follows.

```
>java −jar mettel2.jar [−i <sf>] [−t <tf>] [−d <od>] [−p <pf>]
```

A file with the syntax specification can be given using the **−i** option. A file with the specification of the tableau rules can be given with the **−t** option. If the **−t** option is specified MetTeL$^2$ attempts to do everything for the user by generating Java source code, compiling it and producing a final executable **jar**-file of the prover. In this case, Java Development Kit, Version 1.6.0 or later is required. The directory where the generated Java source code is placed can be given using the **−d** option.

With the **−p** option the user can specify the name of a standard Java property file where currently a small number of properties can be configured. Figure 1 lists the properties currently

| | |
|---|---|
| **`tableau.rule.delimiter`** | Terminator of tableau rules. Default: **`$;`** |
| **`tableau.rule.branch.delimiter`** | Separator between branches in branching rules. Default: **`$\|`** |
| **`tableau.rule.premise.delimiter`** | Separator of premises and conclusions in rules. Default: **`/`** |
| **`branch.bound`** | An expression for computing an apriori bound on the maximal number of expressions in a branch. Default: empty, this means the feature is disabled |

Figure 1: Properties accepted by MetTeL².

supported by MetTeL². In order to be able to handle logics with eventualities a non-standard feature to realise the 'avoid huge branch strategy' (cf., [10, 28]) is the **`branch.bound`** property. For example, this line in the Java property file

```
branch.bound = ((int)(java.lang.Math.pow(2,%l)))
```

configures the generated prover so that any branch is discarded once it contains more than $2^{\%l}$ expressions, where **`%l`** is the parameter for the length of the input expression. **`%l`** is the only pattern variable available in the current MetTeL² implementation but we plan to introduce several other patterns, e.g., pattern variables that reflect the number of atomic expressions of each sort in the input. The property file is also reserved for other non-standard features, flag settings, and definitions that may be required for advanced tuning of the prover generation process.

All the options are optional. In the case that the −**`i`** option is omitted, MetTeL² waits for a language specification from standard input. If the −**`t`** option is not given, MetTeL² will not generate a **`jar`**-file of the prover. In this case only Java code for the prover is generated. This is useful, for example, if the user is going to amend the code with the aim of tailoring the prover for performance or defining a non-standard feature, or simply wishes to compile the code by a Java compiler provided by a different vendor. Whenever −**`d`** is not specified the default directory for output of Java code is the subdirectory **`output`** of the current directory. If the −**`p`** option is omitted the default values are used.

## 5 Prover generation

The parser for the specification of the user-defined logical language is implemented using the ANTLR parser generator. The specification is parsed and internally represented as an abstract syntax tree (AST). The internal ANTLR format for the AST is avoided for performance purposes. The created AST is passed to the generator class which processes the AST and produces the following files: (i) a hierarchy of Java classes representing the user-defined logical language, (ii) an object factory class managing the creation of the language classes, (iii) classes representing substitution and replacement, (iv) an ANTLR grammar file for generating a parser of the user-specified language and the tableau language, (v) a main class for the prover parsing command line options and initiating the tableau derivation process, and (vi) JUnit test classes for testing the parsers and testing the correctness of tableau derivations. In the current version, for testing purposes, most of the classes related to the derivation process are combined in a separate library. In future versions, more and more classes from this library and their extensions will migrate to the generated parts. This will allow the production of faster provers tailored for particular application areas.

The generated Java classes for syntax representation and algorithms for rule application follow the same paradigm as in the previous MetTeL system [31]. All generated Java classes for the syntax representation are specialisations of the basic MettelExpression interface. For efficiency reasons, each kind of expression is represented as a separate Java class, which is not parameterised by operators. At runtime, the creation of expression objects is managed by means of a factory pattern generated as a specialisation of the base interface MettelObjectFactory and ensures that each expression is represented by a single object. Each generated expression class implements several methods. The two most important are: (i) a method for matching the current object with the expression object supplied as a parameter. This method returns the substitution unifying the current expression with the parameter. (ii) The second method returns an instance of the current expression with respect to a given substitution.

Every node of the tableau is represented as a tableau state object comprising of a set of formulae associated with the node and methods for manipulating the formulae and realising rule applications.

The application of rules is implemented as follows. Every rule is applied within a tableau state. A tuple of formulae from the set of active formulae associated with the tableau state is selected and the formulae in the tuple are matched with the premises of the chosen rule. Since matching is computationally expensive, it is performed only once for any given formula and each premise of a rule. This is achieved by maintaining sets of all the substitutions obtained from matching the selected formula with the rule premises. All the selected formulae are discarded from the set of active formulae associated with the rule. If the tuple of the selected formulae match the premises of the rule, the resulting substitution object is passed to the conclusions of the rule. The final result of a rule application is a set of branches, which are sets of formulae obtained by applying the substitution to the conclusions of the rule.

## 6 Built-in optimisations

MetTeL$^2$ implements two general techniques for reducing the search space in tableau derivations: dynamic backtracking [14] and conflict directed backjumping [11, 25]. Dynamic backtracking avoids repeating the same rule applications in parallel branches by keeping track of rule applications common to the branches. Conflict-directed backjumping derives conflict sets of expressions from a derivation. This causes branches with the same conflict sets to be discarded. Since MetTeL$^2$ is a prover generator, dynamic backtracking and backjumping needed to be represented and implemented in a generic way completely independent of any specific logical language and tableau rules. Although dynamic backtracking and backjumping are known for some time and have been used in many tableau provers they are usually defined for a particular tableau procedure which is based on some fixed syntax and fixed tableau calculus. The implementation in MetTeL$^2$ is especially involved for the case of backjumping where calculations of conflicting sets are closely tied to the rules of the tableau calculus. To the best of our knowledge, MetTeL$^2$ is the first system which implements these techniques in a generic way for any logical syntax and any calculus. The performance achieved by these optimisations has not been specially tested, but, due to these optimisations, the total execution time of the generated provers on examples developed for testing the MetTeL$^2$ generator and the generated provers decreased more than 100-fold.

The provers generated by MetTeL$^2$ come with support for ordered backward and forward rewriting with respect to equalities appearing in the current branch. In the language specification, equality expressions can be identified with one of the built-in keywords **equality**, **equivalence** or **congruence**. For example, the declaration formula **equivalence** = formula '<->'formula; in the logic specification defines the binary operator `<->`. The keyword **equivalence** signals that reasoning with this operator should be realised by rewriting.

Each Java class representing a tableau node keeps a rewrite relation completed with respect to all equality expressions appearing in a branch. Since only ground expressions are allowed in a branch, the rewrite relation operates only on ground expressions. Every equality expression is oriented by a lexicographic path ordering $\prec$ based on the order of creation of atomic expressions in the branch. Hence, any (ground) rewrite system based on any such ordering $\prec$ is terminating. Thus, if an equality expression $\alpha \leftrightarrow \beta$ appears on the branch one of the rewrite rules $\alpha \xrightarrow{\mathcal{R}} \beta$ or $\beta \xrightarrow{\mathcal{R}} \alpha$ is added to the rewrite relation depending on whether $\beta \prec \alpha$ or vice versa.

Once an equality expression is added within a tableau node, backward rewriting is applied. This means the rewrite relation is rebuilt with respect to the newly added equality, and all expressions of the node are rewritten with respect to the rewrite relation. Forward rewriting (with respect to the current rewrite relation) is applied to all new expressions added to the branch during the derivation.

In future we plan to provide an implementation of a completion procedure for added equalities which takes a care about dependencies of derived rewrite rules for more efficient backtracking.

## 7 Controlling derivations and blocking

The core tableau engine of MetTeL2 provides various ways for controlling derivations. The default search strategy is depth-first left-to-right search which is implemented as a MettelSimpleLIFOBranchSelectionStrategy request to the MettelSimpleTableauManager. Breadth-first search is implemented as a MettelSimpleFIFOBranchSelectionStrategy request and can be used after a small modification in the generated Java code. A user can also implement their own search strategy and pass it to MettelSimpleTableauManager. In future more search strategies will be implemented (e.g., strategies with iterative deepening) and the choice of strategy will be configurable at the generation stage.

The rule selection strategy can be controlled by specifying priority values for the rules in the tableau calculus specification. The rule selection algorithm checks the applicability of rules and returns a rule that can be applied to some expressions on the current branch according to the rule priority values.

First, the algorithm selects a group of rules with the same priority value. Selection of a group with higher priority value is made only if no rules with smaller priority values are applicable. That is, if several rules are applicable preference is given to rules from groups with smaller priority values.

Second, rules with the same priority values are checked for applicability sequentially. Usually, fair application of rules is a necessary condition to achieve completeness of a tableau derivation. An application of rules during derivation is *fair* if every rule which is applicable to some expressions in a branch is eventually applied to these expressions or a contradiction is detected beforehand. To ensure fairness for rules within the same priority group all rules within the group are checked for applicability an equal number of times. For example, given a single closure rule, to achieve that the closure rule is applied immediately after any new information is added to a branch the user can assign to the closure rule the priority value 0 and to all other rule values higher than 0, e.g., 1. If, however, several rules are assigned the same priority value as the closure rule it can happen that several tableau expansion rules are applied before the branch is checked for contradictions via the closure rule. Furthermore, if the closure rule has a priority value strictly greater than other rules, then the branch is checked for contradiction via the closure rule only after it has been fully expanded, i.e., no expansion rule is applicable to it. More subtle control for application of the closure rules can be also achieved, e.g., to ensure that the closure rules are applied only after particular expansion rules. On the side we remark that it is not necessarily a good idea to give closure rules highest priority in a group by themselves as then performance may degrade because contradiction testing dominates the search.

Again the user could implement their own rule selection strategy and modify the generated code.

To achieve termination for semantic tableau approaches some form of blocking is usually necessary. To generate a prover with blocking the user can specify a blocking rule similar to the unrestricted blocking rule from [28] as one of the rules of the tableau calculus. If the definition of the rule involves equality operators then rewriting is triggered (see above), and, based on the results in [27, 28], the blocking rule can be used to achieve termination for logics with the finite model property.

Consider, for example, the following declarations which might be part of the language specification for a description (or hybrid) logic.

```
sort concept, individual;
concept at = '@' individual concept | negation = '~' concept;
concept equality = '[' individual '=' individual ']';
```

This defines respectively the sorts concept and individual and two operators @ and ~. The last line defines an equality operator = on individuals which is handled by rewriting. The unrestricted blocking rule can now be defined by the following tableau rule.

```
@i p @j q / [i = j] $| ~[i = j] $;
```

The purpose of the two premises here is domain predication so that, on application of the rule, the variables i and j are instantiated by individuals which are present in the current tableau node (cf. [27]), because symbols that do not occur in premise positions are not instantiated. In essence the rule causes individuals occurring in expressions of the form @i p to be systematically set to be equal. If this does not lead to a model in the left subtableau, then the right branch is explored.

The idea of unrestricted blocking rule is to ensure termination of sound and complete tableau calculus in case the specified logic has the finite model property (cf. [27, 28]) and find finite models. The first of the two termination conditions in [27, 28] is automatically true because the generated provers are equipped with ordered rewriting. The second termination condition can be satisfied by using appropriate priority values for tableau rules of the tableau calculus. Currently, it is not yet possible to emulate standard blocking techniques such as subset and equality blocking but by varying the specification of the blocking rule it is possible to perform blocking more selectively [4, 18].

## 8 Using the generated provers

The generated prover **jar**-file can be run via the command line as follows.

```
>java −jar <prover_name>.jar [−i <if>] [−o <of>] [−t <tf>]
```

**<prover_name>** is the name of the syntax specification. An input file **<if>** can be specified via the **−i** option. If the option is not specified then input is expected from the standard input. The input file must contain a list of expressions of the main sort (the *first* specified sort in the syntax specification) separated by space characters. The prover will output the result to the file `<of>`, if the option `-o` is given, or to the standard output stream, otherwise. With the `-t` option the user can specify a file with an alternative definition of a tableau calculus. If the option is omitted then the calculus specified at generation is used. If no tableau calculus was specified at generation then a tableau calculus definition must be provided now, which can be done with the `-t` option.

The generated provers return the answers **`Satisfiable`** or **`Unsatisfiable`**. If the answer is **`Unsatisfiable`** and the prover is able to extract the input expressions needed for deriving the contradiction they are printed. If the answer is **`Satisfiable`** then all the expressions within the completed open branch are output as a **`Model`**.

Considering our list example, the user can run the prover generated from the syntax and tableau specifications in Sections 2 and 3 as follows.

```
>java -jar lists.jar
```

Since the `-i` option is not specified the prover will wait for input from the terminal. Suppose `{<a (<b L>)> != <a (<b L>)>}` is typed (and finished by pressing `<Ctrl-D>`). The output is

```
Unsatisfiable.
Contradiction: [({(<a (<b L>)>) != (<a (<b L>)>)})]
```

For the input `{<a (<b L0>)> != <a (<b L1>)>}` the output is

```
Satisfiable.
Model: [({(<a (<b L0>)>) != (<a (<b L1>)>)}), ({(<b L0>) !=
(<b L1>)}), ({L0 != L1})]
```

In order to test validity of a formula in the three-valued Łukasiewicz logic $Ł_3$ we have to run the prover two times with truth values `U` and `F` for the formula (cf. [17]). For example, to show that `p -> (q -> p)` is a theorem in $Ł_3$ we run the prover two times for each expression `U p -> (q -> p)` and `F p -> (q -> p)` using the following command.

```
>java -jar Lukasiewicz3.jar
```

In the case if the input is `U p -> (q -> p)` we get the following output from the prover:

```
Unsatisfiable.
Contradiction: [( U ( p -> ( q -> p ) ) )]
```

For the input `F p -> (q -> p)` we also obtain a contradiction:

```
Unsatisfiable.
Contradiction: [( T p ), ( F p ), ( F ( p -> ( q -> p ) ) ),
( F ( q -> p ) )]
```

Thus, there is no interpretation of variables `p` and `q` in the three element Łukasiewicz algebra that makes the truth value of the formula `p -> (q -> p)` different from `T`. That is, `p -> (q -> p)` is a theorem of $Ł_3$.

## 9 Illustration of scope and further features

MetTeL2 is designed for propositional logics and not supposed to deal with first-order languages. Nevertheless, the syntax specification language of MetTeL2 has enough expressive power to represent languages of first-order theories with a finite number of predicate and functional symbols. Predicate and functional symbols of such a theory can be defined as connectives of formula sort and term sort respectively. For example, the following is a specification of a language of a first-order theory which contains a constant c, a unary function symbol f, a binary function symbol g, a unary *constant* predicate symbol P, and a binary *constant* predicate symbol Q.

```
specification fotheory;
syntax SomeFOTheory{
    sort formula,term,var;
    term var = '!' var;
    term c = 'c';
    term f = 'f' '(' term ')';
    term g = 'g' '(' term ',' term ')';
    formula true = 'true' | false = 'false';
    formula P = 'P' '(' term ')';
    formula Q = 'Q' '(' term ',' term ')';
    formula negation = '~' formula;
    formula conjunction = formula '&' formula;
    formula disjunction = formula '|' formula;
    formula implication = formula '->' formula;
    formula equivalence = formula '<->' formula;
    formula universalQuantifier = 'forall' var formula;
    formula existentialQuantifier = 'exists' var formula;
}
```

The additional connective ! is required to embed sort of variables var into the sort of terms term. The following expression is an example of a well-formed formula in the specified syntax.

```
forall x (forall y (~ P(g(f(!x),!y))) | (exists y Q(f(!y),!x)))
```

Notice that predicate symbols P and Q are constant predicate symbols. In fact, under the above approach to syntax specification, every predicate symbol of a given theory is a constant symbol. The effect of this becomes apparent in interpretation tableau rules where substitutions for these symbols will be forbidden. While the rules for Boolean connectives do not pose a problem and remain the same as for Boolean logic, the standard rules for quantifiers must be appropriately instantiated for every predicate symbol. Correct specification of a complete tableau calculus in such cases is tedious and even an impossible task for some theories.

A more promising approach to deal with first-order theories in MetTeL2 is to represent the given finitely defined first-order theory as a deductive system of formula schemes [32]. Since every deductive system in the sense of [32] is a propositional multi-modal logic, it can be naturally specified in MetTeL2.

For the curious reader we also give an example of how to specify in MetTeL2 a rule for the $\Diamond$ operator of standard modal logics. The syntax specification has to include a definition of an additional connective which corresponds to a Skolem function. For example, the syntax of a (hybrid) modal logic can include the following lines.

```
sort formula, individual;
formula singleton = '{' individual '}';
formula at = '@' individual formula;
formula diamond = '<>' formula;
individual SkolemTerm = 'f' '(' individual ',' formula ')';
```

Thus, the symbolic representation of the $\Diamond$ operator is <> and f(.,.) is a new binary connective such that f(i,p) is an individual for any individual i and formula p. In this syntax, the standard $\Diamond$ rule is as follows.

```
@i(<> p) / @i(<> f(i,p)) @f(i,p) p $;
```

That is, for every instance of the formula @i(<>p) in current tableau branch, a new individual term f(i,p) is created (where i and p are appropriately instantiated) and two formulae @i(<> f(i,p)) and @f(i,p) p are added to the branch. The formula @i(<> f(i,p)) states that the individual f(i,p) is accessible from i.

Using Skolem terms instead of newly generated individual constants avoids the need to perform again some inference steps on the same branch and in combination with ordered rewriting can considerably reduce the search space. Furthermore, Skolem terms add flexibility to tableau specification language because Skolem functions are legal not only in conclusions of tableau rules but also in their premises.

## 10 Application areas and experiences so far

Software to generate code for provers is useful anywhere where automated reasoning is needed. The provers generated by MetTeL2 also output models for for satisfiable problems on termination, so can be used for model generation purposes.

With MetTeL2 a quick implementation of a tableau prover can be obtained and changes can be made without programming a single line of code. Prover generation is useful for obtaining provers for newly defined logics or new combinations of logics. This is particularly pertinent to an area such as multi-agent systems where the logics are staggering complex. In ongoing work we are using MetTeL2 in combination with the tableau synthesis framework to develop provers for multi-agent interrogative epistemic logics [21]. For these logics and related dynamic epistemic logics there are almost no implemented reasoning tools. Therefore being able to generate tableau provers is very useful especially to researchers without the resources or expertise to implement automated reasoning tools themselves.

We have found MetTeL2 useful for analysing tableau calculi under development whose properties are not known yet. For example, in research conducted for [19] we used MetTeL2 to determine the refinability or unrefinability of tableau rules for a modal logic with global counting quantifiers operators. MetTeL2 can also be used to compare the effectiveness of different sets of tableau rules for the same logic. For example, with minimal effort it is possible to compare the effectiveness of standard tableau calculi with calculi following the KE approach where disjunction is handled by an analytic cut rule and a unit propagation rule.

Concrete case studies we have undertaken with MetTeL2 include implementing unlabelled tableau calculi for Boolean logic and three-valued Łukasiewicz logic, labelled tableau calculi for standard modal logics K, KT, S4, description logic $\mathcal{ALCO}$, a hybrid logic $\mathrm{K}(E_n)$ with global counting operators [19], a multi-agent interrogative epistemic logic [21], and internalised tableau calculi for hybrid and description logics. We used MetTeL2 to implement a tableau decision procedure for $\mathcal{ALBO}^{\mathsf{id}}$, a description logic with the same expressive power as the two-variable fragment of first-order logic. Some of these test cases, including the Lukasiewicz3 example and the lists example from this paper (as well as an extended version of the lists example with a concatenation operator) are available at the MetTeL website [1].

## 11 Related work

A fast and robust method to obtain a prover for a given logic is to translate the logic into a more expressive target logic for which an automated reasoning tool is available. For instance, translation of modal and description logics into first-order logic has been extensively researched, and specialised translation approaches have been developed which use first-order resolution theorem provers [9, 22, 26] or provers for higher-order logic such as LEO II [6] (see [5] for the approach). Translation approaches require however the user is familiar with the target logic. In addition it requires knowledge of the deduction approach and prover for the target logic so that the user can appropriately use the prover and understand its output. Experience shows it is unrealistic to expect users to learn a new language, a new theory, and capabilities and flag settings of the prover being used.

Several dedicated systems exist for developing and prototyping tableau provers for modal-type logics. The Logics Workbench [15] implements a suite of generic decision procedures for propositional logic and numerous non-classical logics and includes a full programming language. In the eighties, work on developing languages for programming in non-classical logics has evolved into the generic tableau prover development platform LoTREC [12]. The Tableau Workbench (TWB) [2] provides a generic prover development platform for modal logics. Both LoTREC and the TWB give users the possibility to program their own tableau prover for modal-type logics using meta-programming languages for building and manipulating formulae, and controlling the tableau derivation process. The Logics Workbench, LoTREC, the TWB and also MetTeL differ in various ways, for example, in the kind of tableau approach used, the specification language provided, the way blocking is performed and configured, and the possibilities to control the way the search performed. Although they have notable features compared to MetTeL$^2$ none of them have the facility for the user to define their own set of logical operators unrelated to the built-in operators.

Support for user-definable languages and rule sets can be found in logical frameworks such as PVS [23], Isabelle/HOL [24] and Coq [7]. These are based on higher-order logic and can be used for prototyping calculi and deductive systems. There are also formal software development tools such as the KeY system [3] which allow the specification of logical theories via a taclet mechanism containing not only rule declarations, but also usage pragmatics.

All these systems are however not designed to produce executable code for a prover but rather act as virtual machines that perform derivations (in some cases interactively but often fully automatically). Even though some incorporate various support tools and are extensible, within a virtual machine it is not possible to accommodate all imaginable requirements for new provers without giving the user appropriate flexibility in the specification language. On the other hand, any specification language necessarily restricts the user. This is actually useful, since it also reduces the potential number of specification errors. MetTeL$^2$ addresses this inescapable dilemma by generating prover code that is ready for possible modifications by an experienced user who may wish to incorporate, for example, specialised simplification routines and optimisation techniques for better performance, or who may want to incorporate the prover into a larger systems requiring automated reasoning, or add other features not supported by the prover generator, the prover engineering platform or the logical framework being used.

## 12 Conclusion

MetTeL$^2$ is a prototypical system intended for experimenting with tableau calculi and prover generation for various logics. It is a small but essential step to the very ambitious goal to create a reliable and easy to use prover generation platform which implements the automated synthesis framework [27]. In line with this goal we intend to expand the system in various ways. In particular, we will give the user more flexibility in controlling derivations by specifying various search heuristics as well as reduction orderings for the ordered rewriting at the generation stage. This will allow the user to contribute to the improvement of the generated provers. We are going to implement a selection of standard blocking mechanisms and also various generic blocking mechanisms based on specialisations of the unrestricted blocking rule. These will help to improve the performance of the generated provers, especially for non-compact logics such as temporal and dynamic logics. Finally, a comparison with other provers and the design of benchmarking suites is necessary in order to estimate the performance of generated provers and to indicate directions for further development of the prover generator.

MetTeL$^2$ can be downloaded from [1]. A web-interface for MetTeL$^2$ is also provided, where a user can input their specifications in syntax aware textareas and generate provers. The user can either download the generated prover as a **`jar`**-file or directly run the generated prover in the interface.

## References

[1] MetTeL website. `http://www.mettel-prover.org`.

[2] P. Abate and R. Goré. The Tableau Workbench. *Electronic Notes in Theoretical Computer Science*, 231:55–67, 2009.

[3] W. Ahrendt, T. Baar, B. Beckert, R. Bubel, M. Giese, R. Hähnle, W. Menzel, W. Mostowski, A. Roth, S. Schlager, and P. H. Schmitt. The KeY tool. *Software and System Modeling*, 4:32–54, 2005.

[4] P. Baumgartner and R. A. Schmidt. Blocking and other enhancements for bottom-up model generation methods, 2008. Manuscript, short version published in *Proc. IJCAR 2006*.

[5] C. Benzmüller and L. C. Paulson. Multimodal and intuitionistic logics in simple type theory. *Logic Journal of the IGPL*, 18(6):881–892, 2010.

[6] C. Benzmüller, L. C. Paulson, F. Theiss, and A. Fietzke. LEO-II: A cooperative automatic theorem prover for classical higher-order logic. In *Proc. IJCAR'08*, vol. 5195 of *LNCS*, pp. 162–170. Springer, 2008.

[7] Y. Bertot and P. Castéran. Interactive theorem proving and program development. Coq'Art: The calculus of inductive constructions. In *Texts in Theoretical Computer Science*, vol. 35. Springer, 2004.

[8] E. W. Beth. *The Foundations of Mathematics.* North-Holland, 1959.

[9] H. De Nivelle, R. A. Schmidt, and U. Hustadt. Resolution-based methods for modal logics. *Logic J. IGPL*, 8(3):265–292, 2000.

[10] C. Dixon, B. Konev, R. A. Schmidt, and D. Tishkovsky. A labelled tableau approach for temporal logic with constraints. Manuscript, `http://www.mettel-prover.org/papers/dkst12.pdf`, 2012.

[11] J. Gaschnig. *Performance Measurement and Analysis of Certain Search Algorithms.* PhD thesis, Carnegie-Mellon University, 1979.

[12] O. Gasquet, A. Herzig, D. Longin, and M. Sahade. LoTREC: Logical tableaux research engineering companion. In *Proc. TABLEAUX'05*, vol. 3702 of *LNCS*, pp. 318–322. Springer, 2005.

[13] G. Gentzen. Untersuchungen über das logische Schliessen. *Mathematische Zeitschrift*, 39:176–210, 1934.

[14] M. L. Ginsberg and D. A. McAllester. GSAT and dynamic backtracking. In *Proc. KR'94*, pp. 226–237, 1994.

[15] A. Heuerding, G. Jäger, S. Schwendimann, and M. Seyfried. The Logics Workbench LWB: A snapshot. *Euromath Bull.*, 2(1):177–186, 1996.

[16] J. Hintikka. Form and content in quantification theory. In *Two Papers on Symbolic Logic*, vol. 8 of *Acta Philosophica Fennica*, pp. 7–55. 1955.

[17] R. Hähnle. Tableaux and related methods. In *Handbook of Automated Reasoning*, pp. 101 – 178. North-Holland, 2001.

[18] M. Khodadadi, R. A. Schmidt, and D. Tishkovsky. An abstract tableau calculus for the description logic $\mathcal{SHOI}$ using unrestricted blocking and rewriting. In *Proc. DL-2012*, 2012. To appear.

[19] M. Khodadadi, R. A. Schmidt, D. Tishkovsky, and M. Zawidzki. Terminating tableau calculi for modal logic K with global counting operators. Manuscript, `http://www.mettel-prover.org/papers/KEn12.pdf`, 2012.

[20] J. Łukasiewicz and A. Tarski. Investigations into the sentential calculus. *Logic, Semantics, Metamathematics*, pp. 38–59, 1956.

[21] Ş. Minică, M. Khodadadi, R. A. Schmidt, and D. Tishkovsky. Synthesising and implementing tableau calculi for interrogative epistemic logics, 2012. In these Proceedings.

[22] H. J. Ohlbach, A. Nonnengart, M. de Rijke, and D. Gabbay. Encoding two-valued nonclassical logics in classical logic. In *Handbook of Automated Reasoning*, pp. 1403–1486. Elsevier, 2001.

[23] S. Owre, J. M. Rushby, and N. Shankar. PVS: A prototype verification system. In *Proc. CADE-11*, vol. 607 of *LNAI*, pp. 748–752. Springer, 1992.

[24] L. C. Paulson. *Isabelle: A Generic Theorem Prover (with a contribution by T. Nipkow)*, vol. 828 of *LNCS*. Springer, 1994.

[25] P. Prosser. Hybrid algorithms for the constraint satisfaction problem. *Computational Intelligence*, 9:268–299, 1993.

[26] R. A. Schmidt and U. Hustadt. First-order resolution methods for modal logics. To appear in *Volume in memoriam of Harald Ganzinger*, 2006.

[27] R. A. Schmidt and D. Tishkovsky. Automated synthesis of tableau calculi. *Log. Methods Comput. Sci.*, 7(2:6):1–32, 2011.

[28] R. A. Schmidt and D. Tishkovsky. Using tableau to decide description logics with full role negation and identity. Manuscript, `http://www.mettel-prover.org/papers/ALBOid.pdf`, 2011.

[29] R. M. Smullyan. *First Order Logic*. Springer, 1971.

[30] D. Tishkovsky. On Beth property in extensions of Lukasiewicz logics. *Siberian Mathematical Journal*, 43:147–150, 2002.

[31] D. Tishkovsky, R. A. Schmidt, and M. Khodadadi. MetTeL: A tableau prover with logic-independent inference engine. In *Proc. TABLEAUX'11*, vol. 6793 of *LNCS*, pp. 242–247. Springer, 2011.

[32] D. E. Tishkovsky. On algebraic semantics for superintuitionistic predicate logics. *Algebra Logic*, 38(1):36–50, 1999.

# Satisfiability Checking and Query Answering for large Ontologies

Christoph Weidenbach[1] and Patrick Wischnewski[1][2]

[1] Max Planck Institute for Informatics,
Saarbrücken, Germany
[2] Universität des Saarlandes,
Saarbrücken, Germany
{weidenbach,wischnew}@mpi-inf.mpg.de

**Abstract**

In this paper we develop a sound, complete and terminating superposition calculus plus a query answering calculus for the BSH-Y2 fragment of the Bernays – Schönfinkel Horn class of first-order logic. BSH-Y2 can be used to represent expressive ontologies. In addition to checking consistency, our calculus supports query answering for queries with arbitrary quantifier alternations. Experiments on BSH-Y2 (fragments) of several large ontologies show that our approach advances the state of the art.

## 1 Introduction

In addition to research in description logics [1], reasoning in ontologies has recently drawn a lot of attention in automated theorem proving [4, 19, 16, 7] as well as database theory [9, 3]. The approaches differ in the expressiveness of the considered logics, the supported reasoning tasks as well as the quality of the reasoning procedures. A focus in description logics is on the sound and complete computation of the concept hierarchy, whereas theorem proving and data base approaches typically consider existentially quantified queries. Concerning the theorem proving approaches we can distinguish complete methods from incomplete ones. Whereas the former guarantee completeness and consistency of the ontology [19], the latter consider very expressive ontologies [4, 16, 7] and aim at providing useful query answering.

The approach of this paper is to keep completeness plus consistency checking, but to push the border of expressivity. Previously [19] we have shown effective satisfiability testing for the language *BSH-Y1* consisting of clauses of the form

| | | | |
|---|---|---|---|
| $\to P(a_1,\ldots,a_n)$ | Ground Fact | $R(x,y),R(y,z)\to R(x,z)$ | Transitivity |
| $S(x)\to T(x)$ | Subsort Relation | $R(x,y),R(x,z)\to y\approx z$ | Functionality |

where all function symbols are constants enjoying the unique name assumption. The language covers the YAGO ontology. We have shown that we can decide satisfiability of the YAGO ontology consisting of 10m clauses over 2m constants of the above form in about one hour by superposition based saturation. Existentially quantified queries with respect to the saturated YAGO ontology can then typically be answered in the range of seconds.

In this paper we consider an extended language, called *BSH-Y2*, where in addition to the above clauses we consider clauses of the form

| | |
|---|---|
| $P_1(t_{11},\ldots,t_{1n_1}),\ldots,P_k(t_{k1},\ldots,t_{kn_k})\to$ | Negative Clauses |
| $P_1(t_{11},\ldots,t_{1n_1}),\ldots,P_k(t_{k1},\ldots,t_{kn_k})\to P(s_1,\ldots,s_m)$ | Defined Relations |

where the $t_{ij}, s_j$ are either constants or variables and all variables of the $s_j$ show up in some $t_{ij}$ (range restriction). Due to a further developed superposition calculus (Section 3) and its implementation, the extended BSH-Y2 language is still suitable to decide large ontologies. We tested our implementation on three large ontologies: the extension YAGO++ of the YAGO ontology fully covered by BSH-Y2 (10m clauses), the BSH-Y2 fragment of the SUMO ontology as it appears in the TPTP library [12], serving about 90% of the TPTP SUMO version (82k clauses, SUMO-Y2), and on the CYC ontology as it appears in the TPTP library [13], serving about 30% of the TPTP CYC version (1m clauses, CYC-Y2). We considered the SUMO ontology and the CYC ontology from the CASC–23 competetion [20]. SPASS-Y2 can check consistency for the YAGO++ and SUMO-Y2 ontologies, where we found 2 logical inconsistencies in SUMO-Y2. So far we have not been able to check consistency of CYC-Y2. We stopped after finding and debugging 35 logical inconsistencies. For further details, see Section 5.

Furthermore, we provide complete reasoning support for queries with arbitrary quantifier alternations, introduced in Section 4. Again queries are typically answered in the range of seconds with respect to the minimal model of a saturated ontology via a special query answering calculus. Note that then, completeness turns into soundness for queries with quantifier alternations. For example, answering a query of the form $\exists x\ \forall y\ \Phi$ requires complete reasoning for the universal quantifier in order to obtain a sound result for the overall query. In Section 5, we provide experimental data for query answering with respect to YAGO++ and SUMO-Y2. The paper ends with a short summary and further discussion of related work. The proofs of the theorems are available in a technical report [23].

## 2 Preliminaries

In this paper we follow the notations from [21]. We assume a first-order language over a signature $\Sigma$ as usual. We use $x$, $y$, $z$ to denote variables, $a$, $b$ and $c$ to denote constants, $s$, $t$, $l$, $r$ to denote terms $P$, $Q$, $S$ to denote predicate symbols, $A$, $B$ to denote atoms $C$, $D$ to denote clauses and $N$ to denote a set of clauses. Let vars be the function returning all variables of a term, an atom, and a clause, respectively. We write $\sigma$ for a substitution.

We only consider Horn clauses which we write in implication form $\Gamma \rightarrow \Delta$ with $\Gamma$ is a multiset of literals and $\Delta$ is either the empty set or a singleton set containing one atom. We call a positive ground unit clause ($\rightarrow A$) a fact and a negative ground unit clause ($A \rightarrow$) a negative fact. For the empty clause we write $\square$.

An inference is a rule of the form

$$\frac{C_1 \quad \ldots \quad C_n}{D}$$

where the clause $D$ can be derived from the premises $C_1, \ldots, C_n$. We say an inference is ground iff all clauses $C_1, \ldots, C_n$ and $D$ are ground. An inference system is a collection of inference rules. As usual for the superposition calculus, inferences will be restricted to maximal positive literals and negative literals that are either maximal or selected.

We say that a ground clause $C$ is *redundant* with respect to a set of clauses $N$ if there exists a set $\{C_1, \ldots, C_k\}$ of ground instances of clauses from $N$ such that $C$ is true in every model of $\{C_1, \ldots, C_k\}$ and $C \succ C_j$, for all $j$ with $1 \leq j \leq k$. A non-ground clause is called *redundant* if all its ground instances are.

A ground inference $\pi$ is *redundant* with respect to $N$ if either one of its premises is redundant in $N$, or else there exists a set $\{C_1, \ldots, C_k\}$ of ground instances of clauses from $N$ such that the conclusion of $\pi$ is true in every model of $\{C_1, \ldots, C_k\}$ and $C \succ C_j$, for all $j$ with $1 \leq j \leq k$, where $C$ is the maximal premise of $\pi$. A non-ground inference is called *redundant* if all its ground instances are redundant.

We say that a set of clauses $N$ is *saturated up to redundancy* with respect to some inference system, if all inferences from $N$ are redundant.

A reduction rule reduces the search space by deleting clauses or by reducing clauses to simpler ones. A reduction is denoted as

$$\mathcal{R}\,\frac{C_1 \quad \dots \quad C_n}{\begin{array}{c} D_1 \\ \vdots \\ D_m \end{array}}$$

where the clause above the bar $C_1, \dots, C_n$ are replaced by the clauses below the bar $D_1, \dots, D_m$. A reduction rule implements a special redundancy criteria.

A *Herbrand* interpretation $I$ is a set of ground atoms. Each ground atom $A$ is called true in $I$ if $A \in I$. It is called false in $I$ if $A \notin I$. A negated atom $\neg A$ is true in $I$ if $A \notin I$. A ground clause $C$ is called true in $I$ if one of its literals is true in $I$. We write $I \models C$ in this case.

## 2.1 Admissible Term Ordering

For reasoning in large domain problems efficiently handling transitivity is important due to the fact that the standard superposition approach is too prolific in this context; it computes the whole transitive closure. The chaining calculus [2] has been designed for efficiently dealing with transitivity in general by avoiding the computation of the transitive closure in many cases. We have integrated the chaining calculus into our new reasoning calculus.

The chaining calculus is defined in terms of an extension of the usual reduction ordering on terms. This extension is called admissible and defined as follows.

An ordering $\succ$ on ground terms and literals is called *admissible* if

- it is well-founded and total on ground terms and literals,
- it is *compatible with reduction on maximal subterms*, i.e. $L \succ L'$ whenever $L$ and $L'$ contain the same transitive predicate symbol $Q$, and the maximal subterm of $L'$ is strictly smaller than the maximal subterm of $L$,
- it is *compatible with goal reduction*, i.e.
  - $\neg A \succ A$ for all ground atoms $A$,
  - $\neg A \succ B$ whenever $A$ is an atom $Q(s,t)$ and $B$ is an atom $Q(s',t')$, such that $Q$ is a transitive predicate and $max(s,t) \succeq max(s',t')$,
  - $\neg A \succ \neg B$ whenever $A$ is an atom $Q(s,s)$ and $B$ atom $Q(s,t)$ or $Q(t,s)$, where $Q$ is a transitive predicate and $s \succ t$.

An ordering on ground clauses is called *admissible* if it is the multiset extension of an admissible ordering on literals.

For implementing an actual admissible ordering a triple $(max_L, p_L, min_L)$ is associated with each literal $L$. Two literals are compared by lexicographically comparing their associated triples. For the comparison of the first and last component of the triples the superposition term ordering $\succ$ is used and for comparing the middle component the ordering $1 > 0$ is used. The individual members of the triples are defined as follows: If $L$ is of the form $Q(s,t)$ for a transitive predicate

$Q$ we set $max_L$ to the maximum of $s$ and $t$, and $min_L$ to the minimum of the two terms (with respect to $\succ$). If $L$ is of the form $A$ or $\neg A$ for some atom $A$ the top symbol of which is not a transitive predicate, we set $max_L = A$ and $min_L = \top$, where $\top$ is special symbol minimal in the term ordering $\succ$. We set $p_L = 1$, if $L$ is negative, and 0 otherwise.

## 2.2 Minimal Model

The chaining calculus assumes a given clause set $N$ which does not contain any transitivity axioms. Instead, it assumes that the respective predicates are marked as transitive. These predicates are treated specially by the chaining rules. The candidate model for a set of Horn clauses that entails the transitive closure is defined in terms of rewrite proofs. The rewrite proof from the term $l$ to $r$ via the transitive predicate $Q$, is denoted as $l \Downarrow_Q^{R_C} r$. A detailed introduction to transitive rewrite proofs can be found in [2].

The following defines a minimal candidate model for a set $N$ of Horn clauses which is also a model of the transitive closure of $N$. Assume that $N$ does not contain any transitivity axioms and assume that the respective transitive predicates are marked as transitive.

**Definition** (Candidate Interpretation)**.** *Let $N$ be a set of clauses from the BSH-Y2 without transitivity axioms such that the transitive predicates of $N$ are in the set* Tr*. Further, let $\succ$ be an admissible ordering. The following defines a candidate interpretation for $N$ and* Tr*. Let $C = \Gamma \rightarrow A$ be a ground instance of a clause from $N$. Suppose $E_{C'}$ and $R_{C'}$ have been defined for all ground clause $C'$ with $C \succ C'$. Then*

$$R_C = \bigcup_{C \succ C'} E_{C'}$$

*if (i) $A \succ \Gamma$, (ii) $A \notin R_C^*$, (iii) $\Gamma \subseteq R_C^*$, and (iv) no literal is selected in $C$ then*

$$E_C = \{A\}$$

*otherwise $E_C = \emptyset$. If $E_C \neq \emptyset$, we say that $C$ is productive and produces $A$.*

$$R_C^* = R_C \cup \{Q(l,r) : l \Downarrow_Q^{R_C} r \wedge Q \in \mathrm{Tr}\}$$

*The interpretation $N_I$ of $N$ is defined as $N_I = \bigcup_C R_C^*$.*

Note, this definition is also defined for sets containing non-ground Horn clauses via the lifting lemma which is a standard result of the superposition framework.

# 3 Superposition for BSH-Y2

In this section we define the BSH-Y2 class and present our new superposition calculus for BSH-Y2.

## 3.1 The BSH-Y2 Class

The set BSH-Y2 is a subset of the Bernays–Schönfinkel Horn fragment with equality. It is able to represent the YAGO ontology as well as large parts of the ontologies SUMO (SUMO-Y2) and CYC (CYC-Y2). It is defined below.

We call a clause $D = \Gamma \rightarrow P(t_1, \ldots, t_n)$ a *definition* for the predicate $P$ if it is range restricted, i.e. $\mathrm{vars}(P(t_1, \ldots, t_n)) \subseteq \mathrm{vars}(\Gamma)$. We also say that $P$ is *defined* by $D$. A predicate

$Q$ is called *$P$–dependent* in a clause set $N$ if $Q$ is defined by a clause $D = \Gamma \to Q(t_1, \ldots, t_n)$ in $N$ and (i) $P(s'_1, \ldots, s'_{n'}) \in \Gamma$ or (ii) there is a predicate $R$ with $R(s''_1, \ldots, s''_{n''}) \in \Gamma$ that is P-dependent. If $Q$ is not $P$–dependent we call it *$P$–independent.* A definition $D = \Gamma \to P(t_1, \ldots, t_n)$ is *acyclic* in a clause set $N$ if $P$ is $P$–independent. A predicate $Q$ is called *transitive dependent* in $N$ iff (i) $Q$ is transitive or (ii) there is a definition $C \in N$ with $C = \Gamma \to Q(t_1, \ldots, t_n)$ and there is a transitive dependent predicate $P$ with $P(s_1, \ldots, s_n) \in \Gamma$. Otherwise, we call $Q$ *transitive independent.*

The *BSH-Y2* class consists of the following types of BSH clauses:

| | |
|---|---|
| $\to P(a, b)$ | facts |
| $R(x, y), R(x, z) \to y \approx z$ | functionality axioms |
| $R(x, y), R(y, z) \to R(x, z)$ | transitivity axioms |
| $P_1(t_{11}, \ldots, t_{1n_1}), \ldots, P_k(t_{k1}, \ldots, t_{kn_k}) \to$ | negative clauses |
| $P_1(t_{11}, \ldots, t_{1n_1}), \ldots, P_k(t_{k1}, \ldots, t_{kn_k}) \to P(s_1, \ldots, s_m)$ | acyclic definitions |
| $S_1(x) \to S_2(x)$ | subsort relations |

where the subset of subsort relations is acyclic and we further assume the unique name assumption for the BSH-Y2 meaning that different constants represent different domain elements.

## 3.2 Calculus for BSH-Y2

In general, verifying the satisfiability in the Bernays–Schönfinkel Horn fragment is EXPTIME complete. Therefore, standard reasoning procedures are too prolific for reasoning in such large ontologies; the experiments in Section 5 confirm this. In [19], we have developed a sound and complete calculus which uses hyperresolution together with the chaining calculus. The resulting reasoning procedure saturates the YAGO ontology in less than one hour. However, this calculus is not able to saturate clause sets containing defined relations of BSH-Y2 in acceptable time. The reason for this observation is that a non-ground transitive atom that occurs in a defined relation causes the chaining calculus to inspect the whole transitive closure of this predicate. This problem arises already if one only adds the following clause to the YAGO ontology:

$$\mathrm{bornIn}(x, y), \mathrm{locatedIn}(y, z) \to \mathrm{bornInTr}(x, z) \tag{1}$$

where locatedIn is transitive.

Therefore, we have developed a new calculus for BSH-Y2 that performs a two layered-reasoning. It separates reasoning about non-transitive predicates from reasoning about transitive predicates via dedicated inference rules. These rules are depicted in the following.

The calculus is defined with respect to a clause set $N$ containing clauses from BSH-Y2. Let $\mathcal{S}_N$ be the sort theory contained in $N$. We switch from the simple clause notation introduced in Section 2 to a clause notation $\Theta \parallel \Gamma \to \Delta$ where $\Theta$ contains solely the sort atoms (monadic atoms) interpreted as negated sort atoms. This notation helps in defining the below rules as it explicitly separates sort atoms from others. During the saturation the sort atoms are treated independently from all other atoms via the rules *Empty Sort* , *Sort Simplification*, and *Static Soft Typing.* Actually, this simulates a particular ordering and selection strategy for these atoms on the standard calculus [5]. More precisely, $T \succ S$ for all subsort declarations $S(x) \to T(x)$. This ordering is well-defined because there are no cycles in $\mathcal{S}_N$. Whenever a clause has an unsolved constraint, this constraint is selected.

An unsolved constraint $\Theta$ of a clause $\Theta \parallel \Gamma \to \Delta$ either contains an atom $T(x)$ such that $x \notin \mathrm{vars}(\Gamma \cup \Delta)$ or an atom $T(a)$ for some constant $a$. Finally, all sort predicates $S$ occurring in $N$ are smaller than any other predicate occurring in $N$. The fact that the monadic predicates are treated separately allows more efficient implementations for their calculus rules.

The sort theory $\mathcal{S}_N$ is static [5] meaning that $N_I \models S(a)$ iff $\mathcal{S}_N \models S(a)$ and this property is invariant on the saturation of $N$ while fixing $\mathcal{S}_N$ from the beginning. This is due to the fact that all positive sort atoms in $N$ occur either as facts or as subsort declarations. Hence, when deriving a clause $S(a), \Theta \,\|\, \Gamma \rightarrow \Delta$ with $\mathcal{S}_N \not\models S(a)$, the clause is a tautology and can be deleted. This is exploited by the implementation of our new calculus. Note also that the relations $\mathcal{S}_N \models S(a)$ and $\mathcal{S}_N \models \exists x\, S_1(x) \wedge \ldots \wedge S_n(x)$ can be efficiently decided by specific algorithms [21].

The chaining calculus [2] assumes that all transitivity axioms are deleted from the clause set $N$ and the respective atoms are marked as transitive. We assume the set Tr that contains all transitive predicate symbols of $N$.

The rule *OECut* [18] ensures that the minimal model $N_I$ respects the unique name assumption, namely $N_I \models a \not\approx b$ for two different constants $a$ and $b$ occurring in $N$. So the disequations are not explicitly added to the clause set $N$.

The superposition calculus for the BSH-Y2 is the following set of inference and reduction rules.

## Non-Transitive Reasoning

### Ordered Hyperresolution for BSH-Y2 (HyperY2)

$$\frac{(1 \le i \le n) \quad \Theta_i \,\|\, \Gamma_i \rightarrow A_i \quad \Theta \,\|\, T_1, \ldots, T_m, B_1, \ldots, B_n \rightarrow \Delta}{(\Theta, \Theta_1, \ldots, \Theta_n \,\|\, T_1, \ldots, T_m, \Gamma_1, \ldots, \Gamma_n \rightarrow \Delta)\sigma},$$

where $n \ge 1$, $T_1, \ldots, T_m$ are transitive atoms, $\Theta_1, \ldots, \Theta_n, \Theta$ are solved, $\Gamma_1, \ldots, \Gamma_n$ contain only transitive atoms, $B_1, \ldots, B_n$ are non-transitive atoms, $\sigma$ is the simultaneous most general unifier of $A_i$ and $B_i$ for all $i \in \{1, \ldots n\}$, respectively, and $A_i\sigma$ are strictly maximal in $(\Theta_i \,\|\, \Gamma_i \rightarrow A_i)\sigma$.

### Object Equality Cutting (OECut)

$$\frac{\| \rightarrow a \approx b}{\square},$$

where $a$ and $b$ are two different constants.

## Transitive Reasoning

### Ordered Chaining for BSH-Y2 (OChainY2)

$$\frac{\Theta_1 \,\| \rightarrow Q(l, s) \quad \Theta_2 \,\| \rightarrow Q(t, r)}{(\Theta_1, \Theta_2 \,\| \rightarrow Q(l, r))\sigma}$$

where $Q \in \mathrm{Tr}$ is a transitive predicate, $\sigma$ is the most general unifier of $s$ and $t$, $\Theta_1$ and $\Theta_2$ are solved, $Q(t, r)\sigma$ is strictly maximal in $(\Theta \,\|\, \Gamma \rightarrow Q(t, r))\sigma$, $l\sigma \not\preceq s\sigma$, $r\sigma \not\preceq t\sigma$, and there are only transitive literals in $\Gamma$.

**Negative Chaining for BSH-Y2 (NChainY2)**

$$\frac{\Theta_1 \,\|\, \to Q(l,s) \quad \Theta_2 \,\|\, \Gamma, Q(t,r) \to}{(\Theta_1\Theta_2 \,\|\, \Gamma, Q(s,r) \to)\sigma}$$

where $Q \in \mathrm{Tr}$ is a transitive predicate, $\sigma$ is the most general unifier of $l$ and $t$, $\Theta_1$ and $\Theta_2$ are solved, $s\sigma \not\preceq l\sigma$, $r\sigma \not\preceq t\sigma$, $Q(t,r)\sigma$ is maximal with respect to $(\Theta \,\|\, \Gamma, Q(t,r) \to)\sigma$, and there are only transitive literals in $\Gamma$.

$$\frac{\Theta_1 \,\|\, \to Q(l,s) \quad \Theta_2 \,\|\, \Gamma, Q(t,r) \to}{(\Theta_1, \Theta_2 \,\|\, \Gamma, Q(t,l) \to)\sigma}$$

where $Q \in \mathrm{Tr}$ is a transitive predicate, $\sigma$ is the most general unifier of $s$ and $r$, $\Theta_1$ and $\Theta_2$ are solved, $l\sigma \not\preceq s\sigma$, $t\sigma \not\prec r\sigma$, $Q(t,r)\sigma$ is maximal with respect to $(\Theta \,\|\, \Gamma, Q(t,r) \to)\sigma$, and there are only transitive literals in $\Gamma$.

**Ordered Resolution for BSH-Y2 (OReY2)**

$$\frac{\Theta_1 \,\|\, \to Q(t_1,t_2) \quad \Theta_2 \,\|\, \Gamma, Q(s_1,s_2) \to}{(\Theta_1, \Theta_2 \,\|\, \Gamma \to)\sigma},$$

where $Q \in \mathrm{Tr}$ is a transitive predicate, $\sigma$ is the most general unifier of $Q(t_1,t_2)$ and $Q(s_1,s_2)$, $\Theta_1$ and $\Theta_2$ are solved, $Q(s_1,s_2)\sigma$ is strictly maximal in $(\Theta \,\|\, \Gamma, Q(s_1,s_2) \to)\sigma$, and there are only transitive literals in $\Gamma$.

## Sort Reasoning

**Empty Sort**

$$\frac{S(x), \Theta \,\|\, \Gamma \to \Delta}{(\Theta \,\|\, \Gamma \to \Delta)\sigma},$$

if $\sigma$ is a substitution with $S(x\sigma)$ is ground, $x \notin \mathrm{vars}(\Gamma \cup \Delta)$, and $S_N \models S(x\sigma)$.

**Sort Simplification**

$$\mathcal{R}\frac{S(a), \Theta \,\|\, \Gamma \to \Delta}{\Theta \,\|\, \Gamma \to \Delta},$$

if $\mathcal{S}_N \models S(a)$. In the sort theory of a clause set from the BSH-Y2 sort simplification coincides with sort resolution.

**Static Soft Typing**

$$\mathcal{R}\frac{S(x), \Theta \,\|\, \Gamma \to \Delta}{},$$

if $S_N \not\models \exists x\, S(x)$.

**Theorem** (Decison Procedure for BSH-Y2). *The BSH-Y2 calculus is sound, complete and terminating for BSH-Y2.*

### 3.3 Implementation

For successfully saturating a clause set from BSH-Y2, an efficient implementation of the new calculus rules is essential. In our implementation we avoid to generate clauses that are redundant and can, therefore, immediately be removed from the search space.

In particular, every time an application of hyperresolution derives a clause $\Theta, S(a) \parallel \Gamma \rightarrow A$ an application of sort simplification becomes possible on the ground sort instance $S(a)$. If $\mathcal{S} \models S(a)$ the clause is reduced to $\Theta \parallel \Gamma \rightarrow A$. If $\mathcal{S} \not\models S(a)$ then the clause is a tautology and can be deleted. Therefore, we have integrated this sort reasoning in the implementation of hyperresolution. This means that the clause $\Theta \parallel \Gamma \rightarrow A$ where $\Theta$ does not contain any further ground sort instances is derived.

Additionally, our implementation of the calculus relies on the efficient term indexing data structure *Filtered context trees* that we have introduced in [19]. We have integrated the implementation of our new calculus together with the data structures in the theorem prover SPASS [22]. We call the resulting version SPASS-Y2.

# 4 Query Answering

In this section we present a query language with arbitrarily many quantifier alternations and the corresponding sound and complete query answering procedure. This procedure answers queries with respect to the minimal model of a clause set from BSH-Y2.

### 4.1 Query Language

Consider the language $\Phi$ defined in terms of the below syntax

$$\Phi \quad := \quad \Gamma \quad | \quad \forall x(\Gamma \rightarrow \Phi) \quad | \quad \exists x(\Gamma \wedge \Phi) \quad | \quad \top$$

where $\Gamma$ is a conjunction of atoms.

In order to guarantee completeness of our new query answering procedure, we require further restrictions on the query language. We call a formula of the language $\Phi$ *variable shielded* iff it is either ground or it is of the form $\exists x(\Gamma \wedge \Phi')$ or $\forall x(\Gamma \rightarrow \Phi')$ and all variables occurring under a transitive dependent predicate in $\Gamma$ or occurring freely in $\Phi'$, also occur under a non-transitive dependent predicate or a sort predicate in $\Gamma$.

We call a formula $\varphi$ a *query* if it is a variable shielded sentence from the language $\Phi$. Further, we assume that a variable is bound by at most one quantifier in a query.

Note, that shielding of the variables is not a real restriction because it can always be achieved via a special predicate entity, s.t. for every constant $c$ of the signature entity$(c)$ is entailed by the minimal model of the respective ontology.

### 4.2 Query Answering Calculus

Let $N$ be the saturation of a set of clauses from BSH-Y2 with respect to the superposition calculus of Section 3 and $\square \notin N$. Further, let $\mathcal{S}_N$ be the sort theory contained in $N$. Our query answering calculus is composed of deterministic rule system with respect to $N$ and $\mathcal{S}_N$ and consists of three calculus rules; one for each type of query: *existential query*, *universal query* and *ground query*. The efficiency of our rule system is based on the following observation.

**Lemma.** *Let $N$ be the saturation of a clause set from the BSH-Y2 with $\square \notin N$ and let $A$ be a transitive independent ground atom. Then $N_I \models A$ iff there is a ground substitution $\sigma$ and a clause $\Theta \parallel \rightarrow B \in N$ with $B\sigma = A$ and $\mathcal{S}_N \models \Theta\sigma$.*

Note, all transitive independent definitions are ground instantiated during the saturation. For all rules, we assume that $A_i$ are transitive independent atoms, $T_i$ are transitive dependent atoms and $S_i$ are sort atoms. Note, each subquery $\Phi'\sigma$ derived from our calculus, is again a query because all variables of $\Phi$ are shielded. Likewise, for all transitive dependent atoms $T_i$ of a query $\Phi$ and a substitution $\sigma$, it holds that $T_i\sigma$ is ground if $\sigma$ is grounding for all transitive independent atoms and all sort atoms of $\Phi$. Because of the previous lemma, it is sufficient to consider only clauses of the form $\Theta \parallel \rightarrow A$ from $N$ as the right premisses.

Verifying the side-conditions of the query answering calculus rules requires to perform entailment operations. The sort entailment of condition 1 and condition 3 is a well-sorted check which is quasi-linear [15]. The entailment check in condition 4 is performed by exhaustively applying the saturation calculus of Section 3 with a set of support strategy. Note, our new calculus is a decision procedure for this minimal model reasoning problem because of Theorem 3.2 and [8].

**Existential query**

$$\frac{\Phi = \exists \overline{x}(\, S_1 \wedge \cdots \wedge S_{n_1} \wedge A_1 \wedge \cdots \wedge A_{n_2} \wedge T_1, \ldots, T_{n_3} \wedge \Phi') \quad \Theta_i \parallel \rightarrow A_i'}{\Phi'\sigma}$$

if there is a grounding substitution $\sigma$ such that

1. $\mathcal{S}_N \models S_i\sigma$ for all $i \in \{1, \ldots, n_1\}$
2. $A_i\sigma = A_i'\sigma$ for all $i \in \{1, \ldots, n_2\}$
3. $\mathcal{S}_N \models \Theta_i\sigma$ for all $i \in \{1, \ldots, n_2\}$
4. $N_I \models T_i\sigma$ for all $i \in \{1, \ldots, n_3\}$

**Universal query**

$$\frac{\Phi = \forall \overline{x}(\, S_1 \wedge \cdots \wedge S_{n_1} \wedge A_1 \wedge \cdots \wedge A_{n_2} \wedge T_1 \wedge \cdots \wedge T_{n_3} \rightarrow \Phi') \quad \Theta_i \parallel \rightarrow A_i'}{\Phi'\sigma}$$

if there is a grounding substitution $\sigma$ such that

1. $\mathcal{S}_N \models S_i\sigma$ for all $i \in \{1, \ldots, n_1\}$
2. $A_i\sigma = A_i'\sigma$ for all $i \in \{1, \ldots, n_2\}$
3. $\mathcal{S}_N \models \Theta_i\sigma$ for all $i \in \{1, \ldots, n_2\}$
4. $N_I \models T_i\sigma$ for all $i \in \{1, \ldots, n_3\}$

**Ground query**

$$\frac{\Phi = S_1 \wedge \cdots \wedge S_{n_1} \wedge A_1 \wedge \cdots \wedge A_{n_2} \wedge T_1 \wedge \cdots \wedge T_{n_3} \quad \Theta_i \parallel \ \to A'_i}{\text{true}}$$

if there is a grounding substitution $\sigma$ such that

1. $\mathcal{S}_N \models S_i$ for all $i \in \{(1, \ldots, n_1\}$
2. $A_i = A'_i\sigma$ for all $i \in \{1, \ldots, n_2\}$
3. $\mathcal{S}_N \models \Theta_i$ for all $i \in \{1, \ldots, n_2\}$
4. $N_I \models T_i\sigma$ for all $i \in \{1, \ldots, n_3\}$

## 4.3 Query Answering Procedure

If $N$ is the saturation of a clause set from BSH-Y2 in terms of the calculus of Section 3 then Algorithm 1 implements the query answering procedure which is sound and complete with respect to the minimal model $N_I$. The strategy corresponds to a quantifier elimination over finite domains.

**Algorithm 1**: AnswerQuery

```
Input: Query $\Phi$, saturated clause set $N$
if $\Phi = \top$ then return true
else if $\Phi = \exists \overline{x}.\Gamma \wedge \Phi'$ then
    foreach $\Phi'\sigma \in \text{ext}(\Phi, \text{N})$ do
        if AnswerQuery($\Phi'\sigma$,N) then return true;
    end
    return false;
else if $\Phi = \forall \overline{x}.\Gamma \to \Phi'$ then
    foreach $\Phi'\sigma \in \text{unv}(\Phi, N)$ do
        if ¬AnswerQuery($\Phi'\sigma, N$) then return false;
    end
    return true;
else if $\text{gnd}(\Phi, N) = \text{true}$ then return true
else return false
```

The algorithm expects as its input a query $\Phi$ and the clause set $N$. First, the algorithm checks whether the given query $\Phi$ is an existential quantified, a universally quantified or a ground query. Then it computes the set of all subqueries obtained by applying the respective calculus rule. The set $\text{ext}(\Phi, N)$ is the set of all subqueries from applying the rule *Existential query* to $\Phi$ and $N$. Likewise, the set $\text{unv}(\Phi, N)$ is the set of all subqueries from applying the rule *Universal query* to $\Phi$ and $N$. Finally, $\text{gnd}(\Phi, N)$ is the result of the application of *Ground query.* If $\text{gnd}(\Phi, N)$ is true then the algorithm returns true otherwise it returns false. Our algorithm processes a query from the outer query to the inner subquery. Checking if $N_I \models \forall x(\Gamma \to \Phi')$ requires to check if each ground instance of $\Gamma$ is also contained in the set of instances of $\Phi'$. Since, we have to compute the ground instances of $\Gamma$ anyway, we process the queries from the outer to the inner query.

Our implementation of the query answering calculus follows exactly Algorithm 1. The rules are implemented following the implementation of hyperresolution style rules. One exception to the straight forward implementation is the condition 4 that checks if a transitive dependent atom $A$ is entailed by $N_I$. We do not use the whole reasoning engine of SPASS-Y2 for this purpose. Instead, we have implemented a procedure that simulates several derivation steps in hyperresolution style macro steps while keeping an efficient implicit representation of the query clause.

**Theorem.** *Let $\Phi$ be a query, $N$ the saturation of a clause set from the BSH-Y2 with respect to the saturation calculus of Section 3. If $N_I$ is the minimal model of $N$, then*

$$N_I \models \Phi \Leftrightarrow \text{AnswerQuery}(\Phi, N) = \text{true}$$

## 5 Experiments

We have extended the automated theorem prover SPASS 3.8 [22] with the saturation and query answering calculus presented in this paper. We call this new version SPASS-Y2. SPASS-Y2 is sound, complete and terminating for BSH-Y2 and additionally provides a query answering engine for queries with quantifier alternations.

For our experiments we used the SUMO and CYC ontologies from the CASC–23 [20] competition. In order to obtain the clause set SUMO-Y2, we extracted all clauses belonging to the BSH-Y2 language from the SUMO ontology file CSR003+2.ax of the TPTP. The clause set SUMO-Y2 contains 82,064 which is about 90% of CSR003+2.ax. The remaining 10% cannot be expressed in the BSH-Y2 fragment. In particular, the relations s__instance and s__subclass can be expressed in BSH-Y2. Likewise, for CYC-Y2 we extracted the BSH-Y2 clauses from the base knowledge of the CYC TPTP file CSR002+5.ax. The clause set CYC-Y2 contains about 1,033,447 clauses out of 3,341,996. We consider only the BSH-Y2 fragment of the base knowledge of CYC in CYC-Y2 because this has already a high level of inconsistency. In other words, we do not consider the microtheories of CYC for our experiments. The YAGO++ ontology that we used for our experiments includes the first-order representation of the YAGO ontology plus further axioms. For example, we added the following definition which is an refinement of the relation locatedIn: $\text{locatedIn}(x, y) \rightarrow \text{locatedInTr}(x, y)$, removed the transitivity axiom for locatedIn and added a transitivity axiom for locatedInTr. This allows us to check the relation locatedIn for additional properties like functionality and antisymmetry. The relation locatedInTr together with the respective transitivity axioms represents the transitive closure of the original locatedIn relation. The resulting clause set YAGO++ contains 9,918,724 clauses over $2m$ constants.

We ran our experiments on a 2 x Intel Xeon Processor X5660 (12 MB Cache, 2.80 GHz) Debian Linux machine with 96 GB RAM with SPASS-Y2 compiled as 64 bit binary. We require a 64 bit architecture for our experiments because SPASS-Y2 needs to address around 20 GB RAM for the saturation of the YAGO++ ontology.

### 5.1 Saturation Procedure

In our experiments we compare clasp 2.0.4 with the grounder gringo [6], DLV [11], Vampire 1.8 [14], E 1.4 [17], iProver 0.8.1 [10] and SPASS 3.8 [22] with SPASS-Y2. SPASS 3.8 contains already some of our data structure from our previous work [19] but not an implementation of the calculi presented here. All provers were called using the recommended default settings with a time limit of 100 min. We ran each of these tools with YAGO++, SUMO-Y2 and CYC-Y2.

| | YAGO++ | | | SUMO-Y2 | | | CYC-Y2 | | |
|---|---|---|---|---|---|---|---|---|---|
| Tool | Derived | Result | Time | Derived | Result | Time | Derived | Result | Time |
| clasp | $1,118,858,572$ | kbs | 70 min | $1,322,070$ | sat | 20 sec | | unsat | 1 min |
| DLV | | t.o. | 100 min | | sat | 30 sec | | t.o. | 100 min |
| Vampire | | kbs | 1 min | | kbs | 25 min | | unsat | 26 sec |
| E | | kbs | 6 min | | t.o. | 100 min | | kbs | 3 min |
| iProver | | kbs | 1 min | $967,678$ | t.o. | 100 min | | kbs | 8 sec |
| Spass3.8 | $49,848,842$ | sat | 60 min | $1,530,025$ | t.o. | 100 min | $18,907,803$ | t.o. | 100 min |
| Spass-Y2 | $2,722,246$ | sat | 16min | $790,691$ | sat | 45min | 328,904 | unsat | 1 min |

Figure 1: Evaluation of Spass-Y2

The results are depicted in Figure 1. The first column shows the tool and the second the results for the respective ontology. The column derived shows the number of newly generated formulas during problem processing. This column contains empty entries because this information was not always available when the prover timed out (t.o.) after 100 min or was killed by operating system/self killed (kbs), depicted in the result column. We have also tested the model finders FIMO 0.2, E-Darwin 1.4, and Paradox 0.4, but none of these tools could find a model for any of the three ontologies within 100 minutes. Except for Spass-Y2, none of these tools could saturate all three ontologies and find inconsistencies. clasp performed nicely on SUMO-Y2 and CYC-Y2, but it could not find a model of YAGO++ because it run out of main memory (96 GB).

## 5.2 Query Answering Procedure

We have tested the query answering abilities of Spass-Y2 in the standard first-order semantics as well as in minimal model semantics. For the evaluation in terms of the standard first-order semantics, we have tested the 20 queries of the SUMO category of the CASC-23 competition. Before answering the queries we have saturated the SUMO-Y2 ontology and removed the logical inconsistencies. We have identified two logical inconsistencies of the SUMO ontology as used in CASC-23. Then we applied the saturation procedure of Spass-Y2 with a, in this case, complete set of support strategy in order to find a proof for the respective query. This approach terminates on 13 problems with a proof and on further five with a consistent saturated set. The latter result is due to the fact that SUMO-Y2 does not contain all SUMO clauses. All results were obtained within one second. The conjectures of the remaining two problems cannot be formulated in the BSH-Y2 language. Spass-Y2 could have answered all of these questions in terms of the inconsistent SUMO ontology (principle of explosion). After identifying and fixing 35 inconsistencies in the base knowledge of CYC, we did not consider CYC for further experiments because it is questionable what an answer in terms of an inconsistent ontology means.

We have tested the query answering procedure of Section 4 of Spass-Y2 by running the procedure on the following queries with respect to the saturated clause set of the YAGO++ ontology.

Each of the following queries regard a particular feature of our query language or the BSH-Y2 language. This includes quantifier alternations, transitive dependent and transitive independent definitions.

$$
\begin{aligned}
Q_1 &= \exists x(\mathrm{politician}(x) \wedge \mathrm{physicist}(x))\\
Q_2 &= \exists x, y, z(\mathrm{hasSuccessor}(x, \mathrm{GeorgeWBush}) \wedge \mathrm{graduatedFrom}(x, z) \wedge\\
&\qquad \mathrm{graduatedFrom}(y, z) \wedge \mathrm{isMarriedTo}(x, y))\\
Q_3 &= \exists x, y(\mathrm{bornIn}(\mathrm{Angela\_Merkel}, y) \wedge \mathrm{locatedIn}(x, y) \wedge \mathrm{country}(y))\\
Q_4 &= \exists x, y(\mathrm{bornIn}(x, y) \wedge \forall z.\, \mathrm{hasChild}(x, z) \rightarrow \mathrm{bornIn}(z, y))\\
Q_5 &= \exists x(\mathrm{bornIn}(x, y) \wedge \mathrm{politician}(x) \wedge \mathrm{locatedIn}(x, \mathrm{Europe}) \wedge \mathrm{physicist}(x))\\
Q_6 &= \exists x(\mathrm{bornIn}(x, \mathrm{Hamburg}) \wedge \mathrm{politician}(x) \wedge \mathrm{physicist}(x) \wedge\\
&\qquad \mathrm{hasSuccessor}(\mathrm{Helmut\_Schmidt}, x))\\
Q_7 &= \forall x(\mathrm{politicianOf}(x, \mathrm{Germany}) \rightarrow \exists y, z.\, \mathrm{hasSuccessor}(y, x) \wedge \mathrm{bornIn}(y, z) \wedge\\
&\qquad \mathrm{locatedIn}(z, \mathrm{Germany}))\\
Q_8 &= \exists x(\mathrm{politician}(x) \wedge \mathrm{bornInCountry}(x, \mathrm{Germany}))
\end{aligned}
$$

The time that SPASS-Y2 needed to answer this queries are depicted in the below table.

| $Q_1$ | $Q_2$ | $Q_3$ | $Q_4$ | $Q_5$ | $Q_6$ | $Q_7$ | $Q_8$ |
|---|---|---|---|---|---|---|---|
| 0:00.79 | 0:01.13 | 1:10.28 | 0:00.33 | 0:09.20 | 0:00.00 | 0:03.36 | 0:29.95 |

The automated theorem proving systems that participated in the CASC-23 LTB division are not suitable in order to answer these queries. They are incomplete because of their axiom selection strategy. In the case of a quantifier alternation, completeness is required for soundness. Furthermore, these systems do not provide a minimal model query answering procedure.

SPASS-Y2 answers most of the queries in a few seconds with respect to minimal model semantics. Our implementation returns "Yes" or a counter example for universal queries and "No" or a complete set of answers for existential queries. For example the query $Q_6$ returns $\{x \mapsto \mathrm{AngelaMerkel}\}$. SPASS-Y2 together with YAGO++, SUMO-Y2, CYC-Y2, and the queries are available from the SPASS homepage `http://www.spass-prover.org/` in section prototypes and experiments. There is also a prototype of a web frontend accessible from `http://spassyago.spass-prover.org/`.

## 6 Conclusion

We have presented a sound and complete superposition calculus for BSH-Y2 covering YAGO++ and large portions of SUMO and CYC. The implementation SPASS-Y2 can effectively decide satisfiability for all three ontologies, where all other systems we have tested fail on at least one input set. clasp performed nicely on SUMO-Y2 and CYC-Y2 but failed on YAGO++ due to the 2m constants and transitive relations preventing efficient grounding. Our results on SUMO-Y2 show that winning the respective CASC competition category can be easily done by focusing on one of the logical inconsistencies. Our results on CYC show, where we stopped after finding and debugging 35 inconsistencies, that it is highly inconsistent. So keeping completeness, but further developing theory and implementation in order to be able to effectively check consistency for large problems can lead to useful insights.

In addition, we provide a new calculus for query processing supporting queries with arbitrary quantifier alternations with respect to consistent and saturated clause sets of YAGO++ and SUMO-Y2. Typical query response times for complex queries are in the range of seconds. There is currently no other implementation of an automated reasoning procedure that supports queries with quantifier alternations with respect to large ontologies out of the BSH-Y2 fragment.

## References

[1] Franz Baader, Diego Calvanese, Deborah McGuinness, Daniele Nardi, and Peter Patel-Schneider. *The Description Logic Handbook: Theory, Implementation and Applications.* Cambridge University Press, March 2003.

[2] Leo Bachmair and Harald Ganzinger. Ordered chaining calculi for first-order theories of transitive relations. *J. ACM*, 45(6):1007–1049, November 1998.

[3] Andrea Calì, Georg Gottlob, and Thomas Lukasiewicz. A general datalog-based framework for tractable query answering over ontologies. In *Proceedings of the twenty-eighth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, PODS '09, pages 77–86. ACM, 2009.

[4] Ulrich Furbach, Ingo Glöckner, Hermann Helbig, and Björn Pelzer. Loganswer - a deduction-based question answering system (system description). In Alessandro Armando, Peter Baumgartner, and Gilles Dowek, editors, *IJCAR*, volume 5195 of *LNCS*, pages 139–146. Springer, 2008.

[5] Harald Ganzinger, Christoph Meyer, and Christoph Weidenbach. Soft typing for ordered resolution. In William McCune, editor, *CADE 14*, volume 1249 of *LNCS*, pages 321–335. Springer, 1997.

[6] M. Gebser, R. Kaminski, B. Kaufmann, T. Schaub, M. Schneider, and S. Ziller. A portfolio solver for answer set programming: Preliminary report. In *LNAI*, volume 6645, pages 352–357. Springer, 2011.

[7] Krytof Hoder and Andrei Voronkov. Sine qua non for large theory reasoning. In Nikolaj Bjørner and Viorica Sofronie-Stokkermans, editors, *CADE-23*, volume 6803 of *LNCS*, pages 299–314. Springer, 2011.

[8] Matthias Horbach and Christoph Weidenbach. Superposition for fixed domains. *ACM Trans. Comput. Log.*, 11(4), 2010.

[9] Ullrich Hustadt, Boris Motik, and Ulrike Sattler. Reasoning in Description Logics by a Reduction to Disjunctive Datalog. *Journal of Automated Reasoning*, 39(3):351–384, 2007.

[10] K. Korovin. iProver – an instantiation-based theorem prover for first-order logic (system description). In A. Armando, P. Baumgartner, and G. Dowek, editors, *IJCAR*, volume 5195 of *LNCS*, pages 292–298. Springer, 2008.

[11] Nicola Leone, Gerald Pfeifer, Wolfgang Faber, Thomas Eiter, Georg Gottlob, Simona Perri, and Francesco Scarcello. The DLV system for knowledge representation and reasoning. *ACM Trans. Comput. Log.*, 7(3):499–562, 2006.

[12] Adam Pease and Geoff Sutcliffe. First order reasoning on a large ontology. In Geoff Sutcliffe, Josef Urban, and Stephan Schulz, editors, *ESARLT*, volume 257 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2007.

[13] Deepak Ramachandran, Pace Reagan, and Keith Goolsbey. First-orderized researchcyc: Expressivity and efficiency in a common-sense ontology. In *In Papers from the AAAI Workshop on Contexts and Ontologies: Theory, Practice and Applications*, 2005.

[14] Alexandre Riazanov and Andrei Voronkov. Vampire 1.1. In Rajeev Gor, Alexander Leitsch, and Tobias Nipkow, editors, *IJCAR*, volume 2083, pages 376–380–380–376–380–380. Springer, 2001.

[15] Manfred Schmidt-Schauß. *Computational Aspects of an Order-Sorted Logic with Term Declarations*, volume 395 of *LNCS*. Springer, 1989.

[16] Michael Schneider and Geoff Sutcliffe. Reasoning in the owl 2 full ontology language using first-order automated theorem proving. In Nikolaj Bjørner and Viorica Sofronie-Stokkermans, editors, *CADE-23*, volume 6803 of *LNCS*, pages 461–475. Springer, 2011.

[17] Stephan Schulz. E - a brainiac theorem prover. *Ai Communications*, 15(2-3):111–126, 2002.

[18] Stephan Schulz and Maria Paola Bonacina. On Handling Distinct Objects in the Superposition Calculus. In B. Konev and S. Schulz, editors, *Proc. of the 5th International Workshop on the Implementation of Logics, Montevideo, Uruguay*, pages 66–77, 2005.

[19] Martin Suda, Christoph Weidenbach, and Patrick Wischnewski. On the Saturation of YAGO. In Jrgen Giesl and Reiner Hähnle, editors, *IJCAR*, volume 6173 of *LNCS*, pages 441–456. Springer, 2010. An extended version of this article can be found in the Technical Report MPI-I-2010-RG1-001.

[20] Geoff Sutcliffe. The cade-23 automated theorem proving system competition - casc-23. *AI Commun.*, 25(1):49–63, 2012.

[21] Christoph Weidenbach. Combining superposition, sorts and splitting. In Alan Robinson and Andrei Voronkov, editors, *Handbook of Automated Reasoning*, volume 2, chapter 27, pages 1965–2012. Elsevier, 2001.

[22] Christoph Weidenbach, Dilyana Dimova, Arnaud Fietzke, Rohit Kumar, Martin Suda, and Patrick Wischnewski. Spass version 3.5. In *CADE-22*, volume 5663, pages 140–145. Springer, 2009.

[23] Christoph Weidenbach and Patrick Wischnewski. Satisfiability checking and query answering for large ontologies. Technical Report MPI-I-2011-RG1-001, Max Planck Institute for Informatics, 2011.