Christoph Benzmüller
Jens Otten (Eds.)

Automated Reasoning in Quantified Non-Classical Logics

3rd International Workshop, ARQNL 2018, Oxford, United Kingdom, July 18th, 2018

Proceedings

Also appeared as

CEUR Workshop Proceedings, Volume 2095
CEUR-WS.org/Vol-2095

Preface

This volume contains the proceedings of the Third International Workshop on Automated Reasoning in Quantified Non-Classical Logics (ARQNL 2018), held July 18th, 2018, in Oxford, United Kingdom. The workshop was affiliated and co-located with the International Joint Conference on Automated Reasoning (IJCAR 2018), which was part of the Federated Logic Conference (FLoC 2018). The aim of the ARQNL 2018 Workshop has been to foster the development of proof calculi, automated theorem proving (ATP) systems and model finders for all sorts of quantified non-classical logics. The ARQNL workshop series provides a forum for researchers to present and discuss recent developments in this area.

Non-classical logics — such as modal logics, conditional logics, intuitionistic logic, description logics, temporal logics, linear logic, multivalued logic, dynamic logic, deontic logic, fuzzy logic, paraconsistent logic, relevance logic, free logic, natural logic — have many applications in AI, Computer Science, Philosophy, Linguistics, and Mathematics. Hence, the automation of proof search in these logics is a crucial task. For many propositional non-classical logics there exist proof calculi and ATP systems. But proof search is significantly more difficult than in classical logic. For first-order and higher-order non-classical logics the mechanization and automation of proof search is even more difficult. Furthermore, extending existing non-classical propositional calculi, proof techniques and implementations to quantified logics is often not straightforward. As a result, for most quantified non-classical logics there exist no or only few (efficient) ATP systems. It is in particular the aim of the ARQNL workshop series to initiate and foster practical implementations and evaluations of such ATP systems for non-classical logics.

The ARQNL 2018 Workshop received eight paper submissions. Each paper was reviewed by at least three referees, and following an online discussion, six research papers were selected to be included in the proceedings. The ARQNL 2018 Workshop also included invited talks by Larry Moss and Giles Reger. Additionally, one research paper was selected for presentation at the workshop.

We would like to sincerely thank the invited speakers and all authors for their contributions. We would also like to thank the members of the Program Committee of ARQNL 2018 for their professional work in the review process. Furthermore, we would like to thank the IJCAR Workshop Chair Alberto Griggio and the Organizing Committee of FLoC 2018. Finally, many thanks to all active participants of the ARQNL 2018 Workshop.

Luxembourg and Oslo, July 2018

Christoph Benzmüller
Jens Otten

Organization

Program Committee

Christoph Benzmüller	University of Luxembourg & FU Berlin, Germany – co-chair
José Luiz Fiadeiro	Royal Holloway University of London
Marcelo Finger	University of São Paulo, Brazil
Didier Galmiche	Université de Lorraine - LORIA, France
Rajeev Goré	The Australian National University, Australia
Andreas Herzig	IRIT-CNRS, France
Sven Linker	University of Liverpool, UK
Aniello Murano	Università di Napoli "Federico II", Italy
Hans De Nivelle	Nazarbayev University, Kazakhstan
Jens Otten	University of Oslo, Norway – co-chair
Valeria De Paiva	Nuance Communications, UK
Xavier Parent	University of Luxembourg, Luxembourg
Revantha Ramanayake	Vienna University of Technology, Austria
Giselle Reis	Carnegie Mellon University, Qatar
Leila Ribeiro	Universidade Federal do Rio Grande do Sul, Brazil
Bruno Woltzenlogel Paleo	Vienna University of Technology, Austria

Workshop Chairs

Christoph Benzmüller
University of Luxembourg (and Freie Universität Berlin)
Avenue de l'Université, L-4365 Esch-sur-Alzette, Luxembourg
E-mail: christoph.benzmueller@uni.lu

Jens Otten
University of Oslo
PO Box 1080 Blindern, 0316 Oslo, Norway
E-mail: jeotten@ifi.uio.no

Contents

Implementations of Natural Logics 1–10
Lawrence S. Moss

Some Thoughts About FOL-Translations in Vampire 11–25
Giles Reger

Pseudo-Propositional Logic 26–33
Ahmad-Saher Azizi-Sultan

A Simple Semi-automated Proof Assistant for First-order Modal Logics 34–48
Tomer Libal

Labelled Connection-based Proof Search for Multiplicative Intuitionistic Linear Logic 49–63
Didier Galmiche and Daniel Méry

Labelled Calculi for Quantified Modal Logics with Non-rigid and Non-denoting Terms 64–78
Eugenio Orlandelli and Giovanna Corsi

System Demonstration: The Higher-Order Prover Leo-III 79–85
Alexander Steen and Christoph Benzmüller

Evidence Extraction from Parameterised Boolean Equation Systems 86–100
Wieger Wesselink and Tim A.C. Willemse

Implementations of Natural Logics

Lawrence S. Moss

Indiana University, Bloomington, IN 47405, USA lmoss@indiana.edu

Abstract

We discuss what is known about implementations of logical systems whose syntax is either a small fragment of natural language, or alternatively is a formal language which looks more like natural language than standard logical systems. Much of this work in this area is now carried out under the name of *natural logic*. Just as in modal logic or description logic, there are many systems of natural logic; indeed, some of those systems have features reminiscent of modal logic and description logic. For the most part, quantification in natural logics looks more like description logic or term logic (i.e., syllogistic logic) than first-order logic. The main design criteria for natural logics are that (1) one can be able to use them to carry out significant parts of reasoning; and (2) they should be decidable, and indeed algorithmically manageable. All of the questions that we ask about the implementation of any logics can be asked about natural logics. This paper surveys what is known in the area and mentions open questions and areas.

1 Introduction

This workshop is concerned with Automated Reasoning in Quantified Non-Classical Logics. The specific contribution in this paper might be called *automated reasoning pertaining to human reasoning, but done in a setting that deviates from the normative frameworks in logic*. The specific area of application is natural language semantics, especially done with an eye towards automated reasoning in formal languages which approximate natural language. Admittedly, these days formal logic in computer science is primarily a tool in areas like verification; connections of logic to AI are less prominent. Nevertheless, the original and primary motivation for logic is (arguably) the study of *inference in language*, from Aristotelian syllogisms onward. This is the target area of this paper.

The natural logic program This is a summary of the program of natural logic, taken from papers and talks:

1. Show the aspects of natural language inference that can be modeled at all can be modeled using logical systems which are decidable.

2. To make connections to proof-theoretic semantics, and to psycholinguistic studies about human reasoning.

3. Whenever possible, to obtain *complete axiomatizations*, because the resulting logical systems are likely to be interesting, and also *complexity results*.

4. To implement the logics, and thus to have running systems that directly work in natural language, or at least in formal languages that are closer to natural language than to traditional logical calculi.

5. To re-think aspects of natural language semantics, putting inference at the center of the study rather than at the periphery.

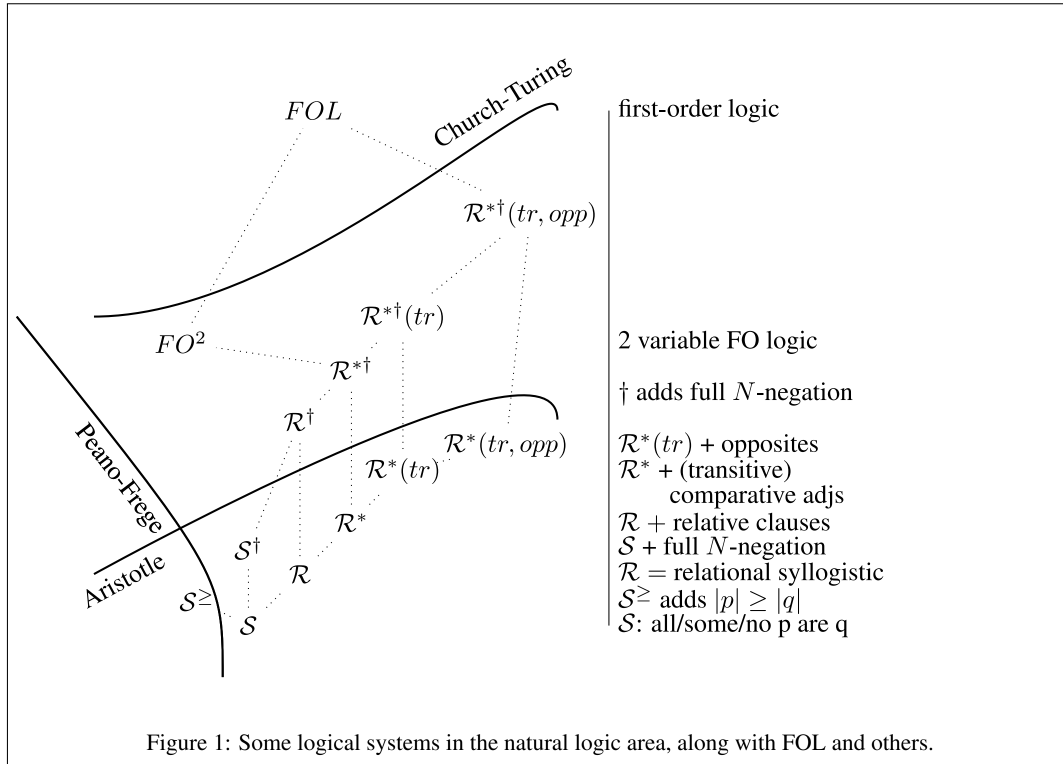


Figure 1: Some logical systems in the natural logic area, along with FOL and others.

Goal (4) is the topic of this paper, but goal (3) is also relevant. Most of the work on natural logic has pursued goals (1) and (3). Concerning (3), there are now a myriad of logical systems which are complete and decidable and which have something to do with natural language. Some of them are listed in a chart in Figure 1. Here is a guide to those systems, starting with the three boundary lines.

The line called "Church-Turing" at the top separates the logics which are undecidable (above the line) from those which are not. FOL is first-order logic. FO^2 is two-variable first-order logic; this is well-known as a decidable logic. This logic is not directly relevant to our study, but since many logics can be expressed in FO^2, they inherit the decidability.

The line called Peano-Frege separates the sub-logics of FOL (to the right) from the one logic on this chart which is not sub-logic of FOL. These logics are "non-classical", being unrelated to FOL. And they deal with "quantity" (but not "quantification" in the usual sense). I would hope that people in the ARQNL community will find them interesting. I discuss them in Section 3. It should be mentioned that there are other logics to the left of the Peano-Frege boundary. Space didn't permit a larger display.

The line called Aristotle separates the logics with a syllogistic formulation from those which lack such a formulation. This line is much less "firm" than the other lines, for two reasons. First, there is no accepted definition of what a syllogistic proof system even is. For us, it means a logic with a finite set of rules, each of which is essentially a Horn clause, and also allowing *reductio ad absurdum*. The second noteworthy feature of the Aristotle line is that sometimes a logic L which is provably above the line has a super-logic L' which is *below* the line. That is, sometimes, adding vocabulary to a logical system enables one to find a syllogistic presentation. This feature has made some doubt whether there is much of a point to the Aristotle boundary. Be that as it may, there certainly is work to do in clarifying the line which we name for Aristotle.

Specific logics We go into details on the syntax and semantics of some of the logics in the chart, starting with $\mathcal{S}$ at the bottom. This is the classical syllogistic. The syntax starts with a collection of *nouns*; we use lower-case letters like p, q, x, and y for nouns. The syntax just has sentences of the form *all x are y*, *some x are y*, and *no x are y*. These sentences are not analyzed as usual in first-order logic. The sentences of $\mathcal{S}$ do not involve recursion. Also, there are no propositional connectives in this tiny logic.

For the semantics, we consider *models* $\mathcal{M}$ consisting of a set M and subsets $[\![x]\!] \subseteq M$ for all nouns x. We then declare

$$\begin{array}{lll} \mathcal{M} \models \textit{all } x \textit{ are } y & \text{iff} & [\![x]\!] \subseteq [\![y]\!] \\ \mathcal{M} \models \textit{some } x \textit{ are } y & \text{iff} & [\![x]\!] \cap [\![y]\!] \neq \emptyset \\ \mathcal{M} \models \textit{some } x \textit{ are } y & \text{iff} & [\![x]\!] \cap [\![y]\!] \neq \emptyset \end{array}$$

Please note that we are not rendering sentences like *all x are y* into first-order logic. Instead, we are giving a semantics to *syllogistic logic*.

Then we ask some very traditional questions. For a set Γ of sentences in this language, and for another sentence φ, we say that $\Gamma \models \varphi$ if every model $\mathcal{M}$ which satisfies all sentences in Γ also satisfies φ. This is the notion of *semantic consequence* from logic. There is a matching proof system, defining a relation $\Gamma \vdash \varphi$. For example, here are two rules

$$\frac{\textit{all } x \textit{ are } y \quad \textit{all } y \textit{ are } z}{\textit{all } x \textit{ are } z}\ \text{BARBARA} \qquad \frac{\textit{all } x \textit{ are } y \quad \textit{some } x \textit{ are } z}{\textit{some } y \textit{ are } z}\ \text{DARII}$$

One feature of the logics in this area is that frequently there are many rules.

As expected, there is a Completeness Theorem. For the purposes of this paper, the important point is that the relation $\Gamma \vdash \varphi$ (for Γ a finite set) is in NLOGSPACE (see [20]), and it is straightforward to implement.

We return to the chart to briefly discuss the other logics. The logic $\mathcal{R}$ above $\mathcal{S}$ adds transitive verbs (such as *see*, *love*, and *hate*), interpreted as arbitrary binary relations on the universe. The syntax of $\mathcal{R}$ is bit complicated, so we won't go into full details. But the following are examples of sentences of it: *all x see some y* and *no x see all y*. In English, sentences like the first of these are ambiguous, and to make a proper logical study we take just the standard reading, where *all* has wide scope. Moving up, $\mathcal{R}^*$ adds *terms* in a recursive way to $\mathcal{R}$. We present the syntax of two related systems formally in Section 2. So *all (see all (love all dogs)) (hate some cat)* is a sentence of $\mathcal{R}^*$; its usual rendering in English would be *all who see all who love all dogs also hate some cat*. $\mathcal{R}^*(tr)$ adds comparative adjectives such as *taller than*, interpreted again as binary relations, but insisting that those interpretations be transitive (hence the *tr*) and irreflexive. Adding converse relations such as *shorter than* takes us up to $\mathcal{R}(tr, opp)$. For example, a logical rule in this logic would be to assume *all x are taller than all y* and conclude *all y are shorter than all x*.

The logics with the dagger † add full negation on all nouns. So in $\mathcal{S}^\dagger$, one can say *some non-x are non-y*. This sentence is not part of the classical syllogistic. We interpret "non" in a classical way, so $[\![\textit{non-}x]\!] = M \setminus [\![x]\!]$. If one wanted to be "non-classical" here, one certainly could.

More on natural logic may be found in the Handbook article [15].

Complexity A great deal is known about the complexity of the consequence relation $\Gamma \vdash \varphi$. The logics at the bottom of the chart, $\mathcal{S}$, $\mathcal{S}^\dagger$, $\mathcal{R}$, and $\mathcal{S}^{\leq}$ are in NLOGSPACE (see [20]). $\mathcal{R}^*$ is NP-COMPLETE (see [13]). The logics $\mathcal{R}^\dagger$, $\mathcal{R}^{\dagger *}$, and $\mathcal{R}^{\dagger *}(tr)$ are complete for EXPTIME. It is open to investigate approximation algorithms for these kinds of logics.

Of special interest for implementations Most of these logics have not been implemented, but of course it would be interesting to do so. Further, we do not even have approximation algorithms for

these logics. All of this would be good to do. The smallest logics in the chart have been implemented, and there are some interesting features. First, once a logic has negation, contradictions are possible. Perhaps the most natural way to handle these is by adding *reductio ad absurdum* to the proof system. However, this complicates the proof search in a big way. And in general, allowing *reductio ad absurdum* in syllogistic logics raises the complexity beyond PTIME. To circumvent *reductio ad absurdum*, one can use *ex falso quodlibet*. This principle allows us to derive an arbitrary conclusion from a contradiction.

Another interesting thing about the implementations of logics "low in the chart" is that *completeness and model-building* are closely related. Here is how this works. Suppose we are given Γ and φ and want to know whether or not $\Gamma \vdash \varphi$; if this hold, we want a proof, and if not, we want a counter-model. One proves a kind of conservativity fact about these logics: if there a derivation of $\Gamma \vdash \varphi$, then there is one with the property that all terms in it are subterms of the premises or of the conclusion. This is a distant relative of the *subformula property*, and it make proof search efficient. One simply generates *all* of the consequences of Γ and looks for φ. If φ is not found, then the sentences which are consequences of it usually gives us a model in a canonical way. The upshot is that one does not need a separate model finder.

Predecessors There are two main predecessor areas to the work that I am discussing, one from Artificial Intelligence, the other from Philosophy. In AI, there is a long tradition of thinking about inference from language. Sometimes this is even taken as a primary problem of natural language processing. But most of that work does not propose small fragments the way we are doing it here. Still, some early papers in the area do work (in effect) with fragments, such as [18, 13]. Those papers have observations that are useful even now. The other background area is Philosophy, especially to those interested in reconstruction of ancient logical systems such as the syllogistic. For this line of work, see [11, 6, 4, 12]. The main difference between it and what we have mentioned so far is that the syllogistic logics in the natural logic area are *extended* syllogistic logics; we are willing to go beyond the ancient systems in every way.

2 Tableau for Beyond Syllogistic Logic

The most prominent topic for the ARQNL workshop is *quantified non-classical logics*. I take it that "quantified" here means "having the standard quantifiers." In Section 3 below, we consider a different sense of "quantified": dealing with quantities (but doing so in a non-classical way). But first, in this section, I want to suggest another direction. Let us return to the chart in Figure 1. The logics above the line marked Aristotle can be shown to have *no syllogistic proof system, not even one that uses (*RAA*)*. The language that I want to mention in this section is $\mathcal{R}^{*\dagger}$. It is a term logic; the variables range over subsets of a model, not individuals. Its syntax comes from [20] and is found in Figure 2. (Actually, I have changed the original syntax in small ways, and my syntax is closer to that of [14].)

For the semantics, we interpret nouns by subsets of the universe M of a model (as in the previous section), verbs by relations on M, noun and verb complements classically, and then inductively interpret terms by

$$\begin{array}{lcl} [\![r\ \textit{all}\ x]\!] & = & \{n \in M : \text{for all } m \in [\![x]\!], (m,n) \in [\![r]\!]\} \\ [\![r\ \textit{some}\ x]\!] & = & \{n \in M : \text{for some } m \in [\![x]\!], (m,n) \in [\![r]\!]\} \end{array}$$

In terms of the complexity, the consequence relation for $\mathcal{R}^*$ is CO-NP COMPLETE (see [13]). For $\mathcal{R}^{*\dagger}$, the complexity jumps to EXPTIME-complete (see [20], following Pratt-Hartmann [19]). In general, the addition of full negation is responsible for a number of complexity jumps in the area.

It is possible to construct natural deduction style proof systems for this logic and related ones [14]. People interested in proof calculi for fragments of first-order logic might find this work interesting. But

Expression	Variables	Syntax
nouns	p, q, x, y	
verbs	s	
unary literal	l	$p \mid \bar{p}$
binary literal	r	$s \mid \bar{s}$
term	c, d	$l \mid r$ *some* $c \mid r$ *all* c
$\mathcal{R}^*$ sentence	φ	*some* b^+ d $\mid$ *some* d b^+ $\mid$ *all* b^+ d $\mid$ *all* d $\overline{b^+}$

Figure 2: Syntax of $\mathcal{R}^{*\dagger}$.

nobody has investigated or implemented proof search for the logics in [14]. So from the point of view of this paper, those systems are less interesting.

Wennstrom [22] provided an *analytic tableau* system for $\mathcal{R}^{*\dagger}$. His system is complete. He showed that for every finite set s of sentences in the language which is unsatisfiable, there is a (finite) closed tableau, proving the unsatisfiability of s. And if s is satisfiable, then the proof search algorithm essentially provides us with a model. In addition, Wennstrom, provided three different tableau search strategies for the language. (We should mention that the tableau system is not strongly complete.) These all have to do with witnesses to existential sentences. In order to have finiteness results of the kind we mentioned, some care is needed.

Wennstrom also implemented all three strategies. He worked in miniKanren [5], a relational programming language embedded in Scheme. He evaluated these on a number of test sets. Evaluation was done using Petite Chez Scheme on a Windows 7 (64 bit) laptop. To give an idea, here is one of the tests:

$$\{\forall(k,m), \exists(\forall(\forall(\exists(k,h),s),r), \exists(\exists(\exists(m,h),s),\bar{r}))\}$$

(The syntax here employs evident abbreviations from what we mentioned in Figure 2. For example $\exists(t,r)$ abbreviates r *some* t.) The fourfold nesting of the quantifiers means that this example was too complicated to actually run.

Other tableau systems and implementations Wennstrom [22] was influenced by Pratt-Hartmann and Moss [20], the paper that put forward most of the systems in Figure 1. But the idea of using tableau comes from Muskens [17], the primary source on tableau methods in connection with natural language reasoning. This work was taken up by Abzianidze in his papers [1, 2, 3]. These papers build on [17], but in addition they implement the ideas. To date, this is the most sophisticated and most successful implementation effort of the kind reported in this paper. The main reason not to go into more detail on it in this paper is that I cannot point to so many problems that could interest people at the ARQNL workshop.

3 Sizes of sets

This section discusses a topic that I think could be of interest at ARQNL, the logic $\mathcal{S}^{\leq}$. This logic adds to syllogistic logic some decidedly non-first order semantics, but with a simple syntax. We add to the classical syllogistic two extra kinds of sentences: *there are at least as many x as y*, and *there are more x than y*. The semantics is just what one would expect: use set theoretic models, and use the standard notion of cardinality. In this discussion, we are going to restrict attention to finite models, since this is most in the spirit of the subject. (But one can work on infinite sets; see [16].)

Again, we should emphasize that these logical systems do not have quantifiers; for that matter, they do not have propositional connectives $\wedge$, $\vee$, or $\neg$. The only sentences are the ones mentioned. In a sense, they are the cognitive module for reasoning about size comparison of sets, in the simplest possible setting.

It might come as a surprise, but reasoning about the sizes of sets does not by itself require a "big" logical system.

There is a complete proof system. It has 22 rules and so we won't present them all. But let us show several of them:

$$\frac{\textit{all } p \textit{ are } q \quad \textit{there are at least as many } p \textit{ as } q}{\textit{all } q \textit{ are } p} \ (\textsc{card-mix})$$

$$\frac{\textit{there are more } q \textit{ than } p}{\textit{there are more } \overline{p} \textit{ than } \overline{q}} \ (\textsc{more-anti}) \qquad \frac{\exists^{\geq}(p,\overline{p}) \quad \exists^{\geq}(\overline{q},q)}{\exists^{\geq}(p,q)} \ (\textsc{half})$$

(CARD-MIX) requires that the universe be finite. In the last two rules, we use $\overline{x}$ as an abbreviation of *non-x*. In (CARD-MIX), we use $\exists^{\geq}(x,y)$ as an abbreviation of *there are at least as many x as y*. So, here is an explanation of the (HALF) rule. If there are at least as many p as non-p, then the p's are at least half of the universe. So if there are at most as many q as non-q, then q's have at most half of the elements in the universe.

As we mentioned before, the logic does not have *reductio ad absurdum*, but it has the related principle of *ex falso quodlibet*. In more detail, *reductio ad absurdum* is an admissible rule, but it is not needed. One can use the following weaker forms

$$\frac{\exists^{\geq}(p,q) \quad \exists^{>}(q,p)}{\varphi} \ (\textsc{x-card}) \qquad \frac{\textit{some } p \textit{ are } q \quad \textit{no } p \textit{ are } q}{\varphi} \ (\textsc{X})$$

These basically say that anything follows from a contradiction. It is open to construct and implement meaningful natural logics which are paraconsistent.

3.1 Implementation

The logic has been implemented in Sage[1]. This implementation is on the cloud, and the author shares it. See `https://cocalc.com/`. Other syllogistic logics have been implemented in Haskell.

For example, one may enter in the CoCalc implementation the following.

```
assumptions= ['All non-a are b',
'There are more c than non-b',
'There are more non-c than non-b',
'There are at least as many non-d as d',
'There are at least as many c as non-c',
'There are at least as many non-d as non-a']
conclusion = 'All a are non-c'
follows(assumptions,conclusion)
```

[1] As a programming language, Sage is close to Python. It is mainly associated with CoCalc (`https://cocalc.com/`); CoCalc incorporates a lot of mathematical software, all of it open source. It provides Jupyter notebooks and course management tools.

We are asking whether the conclusion follows from the assumptions. This particular set of assumptions is more complicated than what most people deal with in everyday life, even when they consider the sizes of sets. This is largely due to the set complements.

As we mentioned, the proof search algorithm basically gives counter-models to non-theorems. In the case of this logic, this fact takes special work. Returning to our example, the result appears instantly. (That is, for a set of assumptions of this type, the algorithm takes less than one second, running over the web.) We get back

```
The conclusion does not follow
Here is a counter-model.
We take the universe of the model to be {0,1,2,3,4,5}
noun semantics
a     {2,3}
b     {0,1,4,5}
c     {0,2,3}
d     {}
```

So the system gives the semantics of `a`, `b`, `c`, and `d` as subsets of $\{0,\ldots,5\}$. Notice that the assumptions are true in the model which was found, but the conclusion is false.

Here is an example of a derivation found by our implementation. We ask whether the putative conclusion below really follows:

$$\frac{\textit{All non-x are x} \qquad \textit{Some non-y are z}}{\textit{There are more x than y}}$$

In this case, the conclusion does follow, and the system returns a proof.

```
Here is a formal proof in our system:

1 All non-x are x     Assumption
2 All y are x     One 1
3 All non-x are x     Assumption
4 All non-y are x     One 3
5 Some non-y are z     Assumption
6 Some non-y are non-y     Some 5
7 Some non-y are x     Darii 4 6
8 Some x are non-y     Conversion 7
9 There are more x than y   More 2 8
```

What about propositional logic? Propositional connectives make complex sentences from sentences. We did not add the propositional connectives to this particular logic to keep the complexity low, and also to illustrate the idea that one should find special-purpose logics for topics like sizes of sets. However, one certainly could add in the connectives. The resulting logic has been explored, but we lack the space to expand on this. It is open to implement such logics, or to connect the work to SAT solvers.

4 Reasoning without grammar, and forgetting about completeness

The ultimate goal of the work that I am reporting on is to have working systems that do inference in natural language, or something close. It is clear that the program of natural logic that I have discussed, where one works with ever more complicated fragments, has a long way to go. I hope that some of the languages which we have found so far will be of interest even to people outside of the area. It is equally clear that for natural language inference itself, the method of fragments has its limitation: to get started with it, one must have a language with a clear syntax and semantics.

Readers of this paper will be surprised by the title of this section. The standard assumptions in all areas of formal logic is that logical language come with a precisely defined semantics and especially a precisely defined syntax. This syntax is usually trivial in the sense that parsing a logical language is a non-issue. For natural language, the opposite is true. Parsing is a complex matter. (The semantics is even worse, of course.)

This section loosen this assumption in connection with sizes of sets. It is based on [9, 8]. It calls on some knowledge of formal and computational linguistics.

We are interested in *polarity marking*, as shown below:

More dogs$^{\downarrow}$ *than cats*$^{\uparrow}$ *walk*$^{\downarrow}$
Most$^{\uparrow}$ *dogs*$^{=}$ *who*$^{=}$ *every*$^{=}$ *cat*$^{=}$ *chased*$^{=}$ *cried*$^{\uparrow}$
Every dog$^{\downarrow}$ *scares* $^{\uparrow}$ *at least two*$^{\downarrow}$ *cats*$^{\uparrow}$

The $^{\uparrow}$ notation means that whenever we use the given sentence truthfully, if we replace the marked word w with another word which is "$\geq w$" in an appropriate sense (see below for an example), then the resulting sentence will still be true. So we have a *semantic inference*. The $^{\downarrow}$ notation means the same thing, except that when we substitute using a word $\leq w$, we again preserve truth. Finally, the $^{=}$ notation means that we have neither property in general; in a valid semantic inference statement, we can only replace the word with itself rather than with something larger or smaller. We call $\uparrow$ and $\downarrow$ *polarity indicators*.

For example, suppose that we had a collection of background facts like *cats* $\leq$ *animals*, *beagles* $\leq$ *dogs*, *scares* $\leq$ *startles*, and one $\leq$ two. This kind of background fact could be read off from WordNet, thought of as a proxy for word learning by a child from her mother. In any case, our $^{\uparrow}$ and $^{\downarrow}$ notations on *Every dog*$^{\downarrow}$ *scares* $^{\uparrow}$ *at least two*$^{\downarrow}$ *cats*$^{\uparrow}$ would allow us to conclude *Every beagle startles at least one animal*. In general, $\uparrow$ notations permit the replacement of a "larger" word and $\downarrow$ notations permit the replacement of a smaller one.

The goal of work on the *monotonicity calculus* such as [21, 10, 9] is to provide a computational system to determine the notations $\uparrow, \downarrow, =$ on input text "in the wild". That means that the goal would be to take text from a published source or from the internet, or wherever, and to then accurately and automatically determine the polarity indicators. Then using a stock of background facts, we get a very simple "inference engine," suitable for carrying out a reasonable fraction of the humanly interesting inferences. The system would handle *monotonicity inferences* [7, 21]. Such a system would not be complete at all, because many forms of inference are not monotonicity inferences.

We must emphasize that [9] is also not a complete success. The work there depends on having a correctly parsed representation of whatever sentence is under consideration, either as a premise of an argument or as the conclusion. And here is the rub: it is rather difficult to obtain semantically useable parses. We have wide-coverage parsers; that is, programs capable of training on data (probabilistically) and then giving structure to input sentences. And the parses of interest to us are those in a grammatical framework called *combinatory categorial grammar* (CCG). CCG is a descendant of categorial grammar (CG), and it is lexicalized; that is, the grammatical principles are encoded in complex lexical types rather than in top-down phrase structure rules. From our point of view, this is a double-edged sword.

On the one hand, CG and CCG connect syntax and semantics because string concatenation in the syntax is matched by function application in the semantics, or to combinators of various sorts. On the other hand, there is no real hope of writing a complete set of rules for logical systems whose syntax is so complicated as to require a grammar of this form. In effect, we give up on completeness in order to study a syntax that is better than a toy. For that matter, working with a wide-coverage parser for a framework like CCG means that some of the parses will be erroneous in the first place. And so the entire notion of deduction will have to be reconsidered, allowing for mistakes along the way.

The connection to automated reasoning There are several reasons why this topic of automated monotonicity reasoning deserves mention in a paper on natural logic for the ARQNL community. Given a set of assumptions in natural language (or even in formal logic), if one has $\uparrow$ and $\downarrow$ information on the individual words and the constituent phrases, then a "good deal" of logical inference follows "easily", by substitution. Admittedly, we are vague here; see [8]. It is a nice open question to quantify in a meaningful way the amount of logical inference that is due to very simple operations, such as monotonicity substitution. Beyond this, there are some interesting technical questions about how exactly one would compute $\uparrow$ and $\downarrow$ information from parse trees.

5 Conclusion

My feeling is that people interested in the topics like "proof calculi, automated theorem proving systems and model finders for all sorts of quantified non-classical logics" will find much to like in the topic of natural logic. Although the systems in the area will be new, and so on a technical level one would be tempted to see differences, there is a close ideological kinship. Both are about automated reasoning, both are willing to consider new systems of various types.

We close with a review of the main points in this survey paper for people in the area.

First, first-order logic (FOL) is not the only approach to the topic of quantification: extended syllogistic logics in the natural logic family offer an alternative. That alternative is not nearly as well-studied as FOL. There are numerous computational problems related to extended syllogistic logics, and there are also a few working systems.

Even if one prefers FOL for the things that it can do, FOL cannot handle some manifestly second-order phenomena such as reasoning about the sizes of sets. We mentioned logics which can cover this reasoning and showed a simple implementation.

An old effort in AI is to study inference from text using automated logic. This topic is currently under revision, due to the connection with the monotonicity calculus. From the point of view of this paper, automating monotonicity inference is an important next step in doing automated reasoning concerning quantification.

References

[1] Lasha Abzianidze. A tableau prover for natural logic and language. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing, EMNLP 2015, Lisbon, Portugal, September 17-21, 2015*, pages 2492–2502, 2015.

[2] Lasha Abzianidze. A tableau prover for natural logic and language. In *EMNLP*, pages 2492–2502, 2015.

[3] Lasha Abzianidze. Natural solution to fracas entailment problems. In *Proceedings of the Fifth Joint Conference on Lexical and Computational Semantics, *SEM 2016, Berlin, Germany, 11-12 August 2016*, 2016.

[4] John Corcoran. Completeness of an ancient logic. *Journal of Symbolic Logic*, 37(4):696–702, 1972.

[5] William E. Byrd Daniel P. Friedman and Oleg Kiselyov. *The Reasoned Schemer*. MIT Press, 2005.

[6] George Englebretsen. *Three Logicians*. Van Gorcum, Assen, 1981.

[7] Bart Geurts. Monotonicity and processing load. *Journal of Semantics*, 22(1):97–117, 2005.

[8] Hai Hu, Thomas F. Icard III, and Lawrence S. Moss. Automated reasoning from polarized parse trees. In *Proc. Natural Language in Computer Science (NLCS'18)*, 2018.

[9] Hai Hu and Lawrence S. Moss. Polarity computations in flexible categorial grammar. In M. Nissim (et al), editor, *Proceedings of The Seventh Joint Conference on Lexical and Computational Semantics, *SEM 2018, New Orleans, Louisiana*, 2018.

[10] Thomas F. Icard and Lawrence S. Moss. Recent progress on monotonicity. *Linguistic Issues in Language Technology*, 9(7):167–194, 2014.

[11] Jan Łukasiewicz. *Aristotle's Syllogistic*. Clarendon Press, Oxford, 2nd edition, 1957.

[12] John N. Martin. Aristotle's natural deduction revisited. *History and Philosophy of Logic*, 18(1):1–15, 1997.

[13] David A. McAllester and Robert Givan. Natural language syntax and first-order inference. *Artificial Intelligence*, 56:1–20, 1992.

[14] Lawrence S. Moss. Logics for two fragments beyond the syllogistic boundary. In *Fields of Logic and Computation: Essays Dedicated to Yuri Gurevich on the Occasion of His 70th Birthday*, volume 6300 of *LNCS*, pages 538–563. Springer-Verlag, 2010.

[15] Lawrence S. Moss. Natural logic. In *Handbook of Contemporary Semantic Theory, Second Edition*, chapter 18. John Wiley & Sons, 2015.

[16] Lawrence S. Moss and Selçuk Topal. Syllogistic logic with cardinality comparisons on infinite sets. *Review of Symbolic Logic*, 2018.

[17] Reinhard Muskens. An analytic tableau system for natural logic. In *Logic, Language and Meaning - 17th Amsterdam Colloquium, Amsterdam, The Netherlands, December 16-18, 2009, Revised Selected Papers*, pages 104–113, 2009.

[18] Noritaka Nishihara, Kenichi Morita, and Shigenori Iwata. An extended syllogistic system with verbs and proper nouns, and its completeness proof. *Systems and Computers in Japan*, 21(1):760–771, 1990.

[19] Ian Pratt-Hartmann. Fragments of language. *Journal of Logic, Language and Information*, 13:207–223, 2004.

[20] Ian Pratt-Hartmann and Lawrence S. Moss. Logics for the relational syllogistic. *Review of Symbolic Logic*, 2(4):647–683, 2009.

[21] Johan van Benthem. *Essays in Logical Semantics*, volume 29 of *Studies in Linguistics and Philosophy*. D. Reidel Publishing Co., Dordrecht, 1986.

[22] Erik Wennstrom. Tableau-based model generation for relational syllogistic logics. In *International Symposium on Artificial Intelligence and Mathematics, ISAIM 2014, Fort Lauderdale, FL, USA, January 6-8, 2014*, 2014.

Some Thoughts About FOL-Translations in Vampire

Giles Reger

University of Manchester, Manchester, U.K.
giles.reger@manchester.ac.uk

Abstract

It is a common approach when faced with a reasoning problem to translate that problem into first-order logic and utilise a first-order automated theorem prover (ATP). One of the reasons for this is that first-order ATPs have reached a good level of maturity after decades of development. However, not all translations are equal and in many cases the same problem can be translated in ways that either help or hinder the ATP. This paper looks at this activity from the perspective of a first-order ATP (mostly Vampire).

1 Introduction

This paper looks at the common activity of encoding problems in first-order logic and running an automated theorem prover (ATP) on the resulting formulas from the perspective of the ATP. This paper focusses on the Vampire [30] theorem prover (available at `https://vprover.github.io/`) but much of the discussion applies to other similarly constructed theorem provers (e.g. E [46] and SPASS [51]).

Over the last few years we have been looking at the application of program analysis/verification, the related encodings, and how Vampire can work with these encodings. In this paper I also mention another setting where we have begun to do some work: working with logics more expressive than first-order logic. This paper comes at the start of an activity to inspect the first-order problems coming from translations involving these logics and considering how we can make Vampire perform better on them.

Throughout the paper I use the terms encoding and translation reasonably interchangeably and lazily. Sometimes the term translation makes more sense (we are moving from one formally defined language to another) and sometimes encoding makes more sense (we are taking a description of a problem and representing it in first-order logic).

This issue of different encodings have differing impacts on how easily a problem is solved is well-known in the automated reasoning community. A standard example is the explosion in conversion to CNF in SAT when not using the Tseitin encoding. The result is obviously bad because it is much larger. In the setting of first-order theorem proving large inputs are also bad but there are other (more subtle) ways in which an encoding can impact the effectiveness of proof search.

The main points of this paper are as follows:

1. The way in which a problem is expressed can have a significant impact on how easy or hard it is for an ATP to solve it and the causes of this go beyond the size of the translation
2. It is often non-obvious whether a translation will be good; we need experiments
3. Sometimes there is no best solution i.e. different encodings may be helpful in different scenarios. In such cases it can be advantageous to move this choice within the ATP

4. Sometimes the only solution is to extend the ATP with additional rules or heuristics to make the ATP treat an encoding in the way we want it to be treated

This points are expanded (with examples) in the rest of the paper.

The rest of the paper is structured as follows. Section 2 describes the relevant inner workings of Vampire that might have a significant impact on the way that it handles different encodings. Section 3 reviews some encodings described in the literature. Section 4 attempts to make some hints about things to consider when designing a representation of a problem in first-order in logic. Section 5 gives some advice on how different encodings should be compared. Section 6 concludes.

2 The Relevant Anatomy of a First-Order ATP

In this section we review the main components of Vampire [30] that might affect how well it handles different encodings. As mentioned above, Vampire shares some elements with other well-known first-order theorem provers.

We consider multi-sorted first-order logic with equality as input to the tool. It will become clear later that Vampire accepts extensions of this in its input, however its underlying logic remains multi-sorted first-order logic with equality and extensions are (mostly) handled as translations into this.

Usually, Vampire deals with the separate notions of *axioms* and *conjecture* (or *goal*). Intuitively, axioms formalise the domain of interest and the conjecture is the claim that should logically follow from the axioms. The conjecture is, therefore, negated before we start the search for a contradiction. Conjectures are captured by the TPTP input language but not by the SMT-LIB input language where everything is an axiom.

2.1 Preprocessing

Vampire works with clauses. So before proof search it is necessary to transform input formulas into this clausal form. In addition to this process there are a number of (satisfiability, but not necessarily equivalence, preserving) optimisation steps. For more information about preprocessing see [17, 43].

Clausification. This involves the replacement of existentially quantified variables by Skolem functions, the expansion of equivalences, rewriting to negation normal form and then application of associativity rules to reach conjunctive normal form. It is well know that this process can lead to an explosion in the number of clauses.

Subformula naming. A standard approach to dealing with the above explosion is the *naming* of subformulas (similar to the Tseitin encoding from SAT). This process is parametrised by a threshold which controls at which point we choose to name a subformula. There is a trade-off here between a (possible) reduction in the size and number of resulting clauses and restricting the size of the signature. Later we will learn that large clauses and a large signature are both detrimental for proof search.

Definition inlining. Vampire detects definitions at the formula and term level. The equivalence $p(X) \leftrightarrow F[X]$ is a definition if p does not occur in formula F, similarly the unit equality

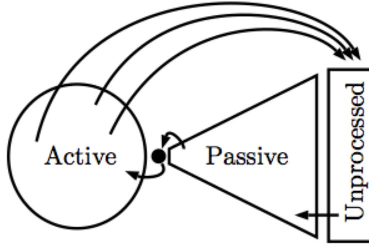


Figure 1: Illustrating the Given Clause Algorithm.

$f(X) = t$ is a definition if f does not appear in term t. Definitions may be *inlined* i.e. all occurrences of p or f are replaced by their definition. This reduces the size of the signature with the cost of a potential increase in the size of clauses. Note that it can be quite easy to break this notion of definition in encodings. For example, by introducing a guard to a definition.

Goal-based premise selection. Vampire can use a goal to make proof search more goal-directed. The point here is that if an encoding does not explicitly highlight the goal we cannot make use of these techniques. There are two techniques used in preprocessing. The first is SInE selection [19] which *heuristically* selects a subset of axioms that are likely to be used in the proof of the goal. Axioms are selected using a notion of closeness to the goal that is based on whether they can be connected to the goal via their least common symbol. The second is the set of support strategy [52] where clauses from the goal are put into a set of support and proof search is restricted so that it only makes inferences with clauses in, or derived from, this set.

2.2 Saturation-Based Proof Search

After a clause set has been produced, Vampire attempts to *saturate* this set with respect to some inference system $\mathcal{I}$. The clause set is saturated if for every inference from $\mathcal{I}$ with premises in S the conclusion of the inference is also added to S. If the saturated set S contains a contradiction then the initial formulas are unsatisfiable. Otherwise, if $\mathcal{I}$ is a complete inference system and, importantly, the requirements for this completeness have been preserved, then the initial formulas are satisfiable. Finite saturation may not be possible and many heuristics are employed to make finding a contradiction more likely.

To compute this saturation we use a set of active clauses, with the invariant that all inferences between active clauses have been performed, and a set of passive clauses waiting to be activated. The algorithm then iteratively selects a *given clause* from passive and performs all necessary inferences to add it to active. The results of these inferences are added to passive (after undergoing some processing). This is illustrated in Figure 1. An important aspect of this process is *clause selection*. Clauses are selected either based on their age (youngest first) or their weight (lightest first) with these two properties being alternated in some specified ratio.

A recent addition to this story is AVATAR [50, 40], which (optionally) performs *clause splitting* using a SAT solver. The main point here is that the success of AVATAR is driven by the observation that saturation-based proof search does not perform well with long or heavy clauses. Therefore, encodings should avoid the introduction of such clauses. As an additional point, AVATAR can only be utilised if the boolean structure of a problem is exposed at the literal-level. For example, including a predicate *implies* with associated axioms would not play

to AVATAR's strengths.

2.3 Inference Rules

Vampire uses resolution and superposition as its inference system $\mathcal{I}$ [1, 34]. A key feature of this calculus is the use of *literal selection* and *orderings* to restrict the application of inference rules, thus restricting the growth of the clause sets. Vampire uses a Knuth-Bendix term ordering (KBO) [23, 25, 32] which orders terms first by weight and then by symbol precedence whilst agreeing with a multisubset ordering on free variables. The symbol ordering is taken as a parameter but is relatively coarse in Vampire e.g. by order of occurrence in the input, arity, frequency or the reverse of these. There has been some work beginning to explore more clever things to do here [22, 38] but we have not considered treating symbols introduced by translations differently (although they will appear last in occurrence).

Understanding this is important as some translations change symbols used in terms, which can change where the term comes in the term ordering. Vampire makes use of both complete and incomplete literal selection functions [18] which combine the notion of maximality in the term ordering with heuristics such as selecting the literal with the least top-level variables.

Another very important concept related to saturation is the notion of *redundancy*. The idea is that some clauses in S are *redundant* in the sense that they can be safely removed from S without compromising completeness. The notion of saturation then becomes saturation-up-to-redundancy [1, 34]. An important redundancy check is *subsumption*. A clause A subsumes B if some subclause of B is an instance of A, in which case B can be safely removed from the search space as doing so does not change the possible models of the search space S. The fact that Vampire removes redundant formulas is good but if there is a lot of redundancy in the encoding we can still have issues as this removal can be lazy (e.g. when using the discount saturation loop that does not remove redundancies from the passive set).

Figure 2 gives a selection of inference rules used in Vampire. An interesting point can be illustrated by examining the demodulation rule. This uses unit equalities to rewrite clauses to replace larger terms by smaller terms. A similar, but more general, rewriting is performed by superposition. Ordering restrictions in inference rules make proof search practical but, as I comment later, can mean that the theorem prover does not treat encoded rules in the way that we want.

2.4 Strategies and Portfolio Mode

Vampire is a portfolio solver [36]. It implements many different techniques and when solving a problem it may use tens to hundreds of different strategies in a time-sliced fashion. Along with the above saturation-based proof search method, Vampire also implements the InstGen calculus [24] and a finite-model building through reduction to SAT [42]. In addition, Vampire can be run in a parallel mode where strategies are distributed over a given number of cores.

This is important as Vampire can try different preprocessing and proof search parameters in different strategies, meaning that it does not matter if a particular encoding does not work well with one particular strategy. Furthermore, if Vampire is allowed to do the translation itself then it can try multiple different translations during proof search.

2.5 Problem Characteristics that Matter

As summary, we can discuss the problem characteristics that matter. The main ones we usually talk about are:

Resolution

$$\frac{\underline{A} \vee C_1 \quad \underline{\neg A'} \vee C_2}{(C_1 \vee C_2)\theta},$$

Factoring

$$\frac{\underline{A} \vee A' \vee C}{(A \vee C)\theta},$$

where, for both inferences, $\theta = \mathsf{mgu}(A, A')$ *and* A *is not an equality literal*

Superposition

$$\frac{\underline{l \simeq r} \vee C_1 \quad \underline{L[s]_p} \vee C_2}{(L[r]_p \vee C_1 \vee C_2)\theta} \quad or \quad \frac{\underline{l \simeq r} \vee C_1 \quad \underline{t[s]_p \otimes t'} \vee C_2}{(t[r]_p \otimes t' \vee C_1 \vee C_2)\theta},$$

where $\theta = \mathsf{mgu}(l, s)$ *and* $r\theta \not\succeq l\theta$ *and, for the left rule* $L[s]$ *is not an equality literal, and for the right rule* $\otimes$ *stands either for* $\simeq$ *or* $\not\simeq$ *and* $t'\theta \not\succeq t[s]\theta$

EqualityResolution

$$\frac{\underline{s \not\simeq t} \vee C}{C\theta},$$

where $\theta = \mathsf{mgu}(s, t)$

EqualityFactoring

$$\frac{\underline{s \simeq t} \vee s' \simeq t' \vee C}{(t \not\simeq t' \vee s' \simeq t' \vee C)\theta},$$

where $\theta = \mathsf{mgu}(s, s')$, $t\theta \not\succeq s\theta$, *and* $t'\theta \not\succeq s'\theta$

Demodulation

$$\frac{l \simeq r \quad \cancel{L[l\theta] \vee C}}{(L[r\theta] \vee C)\theta},$$

where $l\theta \succ r\theta$

UnitResultingResolution

$$\frac{C \vee A_1 \vee \ldots A_n \quad \neg B_1 \quad \ldots \quad \neg B_n}{C\theta},$$

where $|C| \leq 1$ *and* $\theta = \bigsqcup \mathsf{mgu}(A_i, B_i)$

Figure 2: Selected inference rules.

- Number of resulting clauses. Clearly, if there are more clauses it will take longer to process them initially. However, the number of clauses is a bad proxy for effort as a small clause set can lead to many consequences, where a large clause set may have almost no consequences at all.

- Size of resulting clauses. Long and heavy clauses lead to even longer and heavier clauses when applying rules such as resolution. Some processes, such as subsumption checking, are exponential in the length of a clause.

- Size of signature. In SAT-solving the relationship with the signature is clear as each symbol represents a potential choice point. In saturation-based proof search we are effectively trying to eliminate the literals in clauses until we get an empty clause. The combination of clause selection and literal selection steers this process. The notion of maximality built into literal selection means that 'bigger' symbols are preferred, driving proof search to effectively eliminate symbols in this order. So, like in SAT solving, the larger the signature the more work we need to do.

But as we can see from the discussions of literal selection in this paper, issues with encodings can be more subtle than this.

3 Examples of Translations/Encodings

This section reviews some translations or encodings targeting first-order logic. I don't attempt to be exhaustive; in fact there are many well-established examples that I will have missed.

3.1 Simplifying things further

Not all ATPs can handle multi-sorted first-order logic with equality. There are methods that can be used to remove things that are not supported or wanted.

Removing sorts. It is not always possible to simply drop sort information from problems as the 'size' of different sorts may have different constraints [11]. Two proposed solutions are to either *guard* the use of sorted variables by a *sort predicate* that indicates whether a variable is of that sort (this predicate can be set to false in a model for all constants not of the appropriate sort) or *tag* all values of a sort using a *sort function* for that sort (in a model the function can map all constants to a constant of the given sort). Both solutions add a lot of clutter to the signature although sort predicates add more as they also require the addition of axioms for the sort predicates. Experimental evaluation [8, 42] concluded that the encodings can be complementary. Similar techniques can be used for removing polymorphism [8]. An important optimisation here is that we can simply drop sorts if they are monotonic [11].

Removing equality. Every atom $t = s$ can be replaced by $eq(t, s)$ with the addition of axioms for reflexivity, symmetry, transitivity of eq and congruence of functions and predicates. However, the result is not able to use the efficient superposition or demodulation rules. Vampire optionally makes this transformation as in some cases it can aid proof search, in particular when we do not require deep equality reasoning. It is necessary to remove equality when using InstGen as this does not support equality.

Removing functions. A well known decidable fragment of first-order logic is the Bernays-Schönfinkel fragment (also known as *effectively propositional*) where formulas contain no (non-zero arity) function symbols (even after Skolemisation). If a problem fits in this fragment then we obtain this useful property. For example, the problem of finding a finite model of a first-order formula can be reduced to a sequence of effectively propositional problems [4].

3.2 Syntactic Extensions for Program Verification

A common setting where we see problems encoded in first-order logic is for program verification. Tools such as Boogie [31] and Why3 [10] produce first-order proof obligations. There tend to be common expressions in such obligations, such as *if-then-else*, which is why languages such as TPTP [48, 49] and SMT-LIB [3] support these features. However, ATPs typically do not support these features directly but via the following translations.

Boolean sort. In first-order logic predicates are of boolean sort and functions are not. However, in programs we typically want to reason about boolean functions, which requires a first-class boolean sort in the problem. This can be encoded in first-order logic (as explained in the FOOL work [26, 28]) by the introduction of two constants *true* and *false* and two axioms $\mathit{true} \neq \mathit{false}$ and $\forall x \in \mathit{bool} : x = \mathit{true} \vee x = \mathit{false}$. This second axiom is problematic as it will unify with every boolean sorted term and assert that it is either true or false. To overcome this we introduce a specialised inference rule that captures the desired behaviour without the explosive nature [26].

If-then-else. Conditionals are a common construct in programs and it is important to model them. This is often done by extending the syntax by a term of the form $\mathsf{if}\ b\ \mathsf{then}\ s\ \mathsf{else}\ t$ for a boolean sorted term b and two terms of the same sort t and s. The question is then how this should be translated to first-order logic. I discuss three alternatives. In each case we assume we are given a formula containing an if-then-else term e.g. $F[\mathsf{if}\ b\ \mathsf{then}\ s\ \mathsf{else}\ t]$. In the first case we translate this formula into two formulas

$$b \rightarrow F[s] \quad , \quad \neg b \rightarrow F[t]$$

where b and its negation are used as guards. This has the disadvantage that it copies F. An altnernative is to produce the three formulas

$$F[g(X)] \quad , \quad b \rightarrow g(X) = s \quad , \quad \neg b \rightarrow g(X) = t$$

where g is a fresh symbol and X are the free variables of the original if-then-else term. This introduces a new symbol g. Finally, we could replace the formula by $F[\mathit{ite}_\tau(b, s, t)]$, where *ite* is a fixed function symbol of the sort $\mathit{bool} \times \tau \times \tau \rightarrow \tau$, and add the general axioms

$$(X = \mathit{true}) \rightarrow \mathit{ite}_\tau(X, s, t) = s \quad , \quad (X = \mathit{false}) \rightarrow \mathit{ite}_\tau(X, s, t) = t$$

capturing the intended behaviour of ite_τ. This only introduce a pair of axioms per set of if-then-else expressions with the same resultant sort.

We now have three different translations and the question is: which should we use? There are various observations we can make at this point. The first translation copies F, which may lead to many more clauses being introduced if F is complex. However, it does not extend the signature, whilst the second two translations do. The last translation introduces some axioms that are likely to be quite productive during proof search. We should also think about what we want to happen here; ideally the guards will be evaluated first and then the rest of the expression either selected or thrown away. Recall that *literal selection* is the mechanism for choosing which part of a clause is explored next. In general, literal selection prefers heavier literals that are more ground and dislikes positive equalities. Therefore, in the second translation it is likely that the guards will be evaluated before the equalities. In general, it is likely that the guard will be simpler than the conditional body and therefore the first translation is likely to not do what we want. However, it is difficult to draw conclusions from this; as discussed at the end of this paper, we should really run some experiments to see. Note that currently Vampire will pick between the first two depending on the subformula naming threshold [27].

Next state relations. Recent work [12] has used a tuple-based let-in expression to encode next-state relations of loop-free imperative programs. These tuple-based let-in expressions are then translated into clausal form by introducing names for the bound functions [26] and relevant axioms for the tuples (if required). The point of this translation was to avoid translation to

single static assignment form as is often done in translations of imperative programs [2]. The reason to avoid such a form is that it drastically increases the size of the signature. Another benefit of this work is that we can translate loop-free imperative programs directly into first-order logic, allowing Vampire to take such programs directly in its input.

3.3 Proofs about Programming Languages

Recent work [15, 14] has used ATPs to aid in establishing soundness properties of type systems. The idea is to translate a language specification and an 'exploration task' into first-order logic. They define and evaluate 36 different compilation strategies for producing first-order problems.

The most interesting observation to make about this work is that they reinvent a number of encodings and fail to take advantage of the encodings that already exist within ATPs. For example, they consider if-then-else and let-in expressions but do not encode these in the TPTP format but choose their own encodings. In this case they make the first choice for encoding conditionals given above and they choose an alternative way to capture let-in expressions. They also explore the inlining of definitions in their encoding, a step that Vampire already takes in preprocessing. Even though they were able to use more contextual information to decide when to inline, they found that their inlining had minimal impact on performance.

A positive point about this work is that they made a thorough comparison of the different encodings, although some of the suggested encodings were likely to hinder rather than help the ATP. In particular, those that remove information from the problem.

3.4 Higher-Order Logic

Many of the automated methods for reasoning in higher-order logic work via (a series of) encodings into first-order logic. Well-known examples are the Sledgehammer tool [9, 35] and Leo III [47]. The translation of higher-order logic to first-order logic has been well studied [33]. In the translation it is necessary to remove three kinds of higher-order constructs and we discuss the standard translation for removing these here.

Handling partial application and applied variables. In higher-order formulas it is possible to partially apply function symbols, which means that we cannot use the standard first-order term structure. The standard solution is to use the so-called *applicative form*. The general idea is to introduce a special *app* symbol (per pair of sorts in the multi-sorted setting). An application *st* is then translated as $app(s,t)$. This also addresses the issue that higher-order formulas can contain applied variables e.g. $\exists f.\forall x.f(x) = x$ now becomes $\exists f.\forall x.app(f,x) = x$.

Whilst this translation is correct it also adds a lot of clutter to clauses that causes problems for proof search. One example of this is in literal selection as applicative form can change the notion of maximal literal. For example, the literal $g(a) = a$ will be maximal in clause $f(a) = a \vee g(b) = b$ if $g \succ f$ in the symbol ordering. However, in $app_{\tau_1}(f,a) = a \vee app_{\tau_2}(g,b) = b$ the maximal literal depends on the ordering of app_{τ_1} and app_{τ_2}, which cannot consistently agree with the symbol ordering (consider h of sort τ_1 such that $h \succ g$). As a result, it seems likely that the number of maximal literals in clauses will significantly increase. Additionally, literal selection heuristics often consider the number of 'top' variables of a literal but the applicative form changes the height of variables in terms. Finally, the applicative form can decrease the efficiency of term indexing as (i) these often use function symbols to organise the tree but these have been replaced, and (ii) terms are larger making the indices larger.

However, the applicative form is only needed for supporting partial application and applied variables. If neither are used in a problem then it is not needed (although note that the combinator translation discussed below for λ-expressions introduces applied variables).

Removing λ-expressions. The standard approach to removing λ-expressions is rewriting with Turner combinators [33] and the addition of axioms defining these combinators. The translation itself can lead to very large terms, which can be mitigated by the introduction of additional axioms at the cost of further axioms and extensions to the signature. An alternative is λ-*lifting* which introduces new function symbols for every nested λ-expression, which again can significantly increase the size of the signature. To avoid combinator axioms cluttering proof search we have begun to explore replacing these with (incomplete) inference rules emulating their behaviour [6].

Translating logical operators. It is sometimes necessary to translate logical operators occurring inside λ-expressions (other embedded logical operators can be lifted via naming). These must then be specified by axioms e.g. OR could be defined by

$$app(app(\mathsf{OR}, x), y) = true \iff X = true \ \vee \ Y = true$$

which produces the clauses

$$\begin{array}{l} app(app(\mathsf{OR}, X), Y) = false \ \vee \ X = true \ \vee \ Y = true \\ app(app(\mathsf{OR}, X), Y) = true \ \vee \ X = false \\ app(app(\mathsf{OR}, X), Y) = true \ \vee \ Y = false \end{array}$$

Alternatively, the last clause could be dropped and replaced by

$$app(app(\mathsf{OR}, X), Y) = app(app(\mathsf{OR}, Y), X)$$

The non-orientability of this last equality suggests that it would be too productive. Finally, whilst it seems most natural to introduce an equivalence, we would not typically need to reason in the other direction and it could be interesting to see how necessary the first clause is.

3.5 Other Logics

As we have seen from the previous example, a common activity is to consider other logics as (possibly incomplete) embeddings into first-order logic. In all cases, an interesting direction of research will be to inspect the kinds of problems produced and consider whether the encoding or ATP could be improved.

Translations via higher-order logic. Many non-classical logics, included quantified multimodal logics, intuitionistic logic, conditional logics, hybrid logic, and free logics can be encoded in higher-order logic [5, 13].

Ontology Languages. In this setting it is more typical to use optimised reasoners for decidable logics. However, some work has looked at translating these logics into first-order logic and employing an ATP. As two examples, Schneider and Sutcliffe [45] give a translation of OWL 2 Full into first-order logic and Horrocks and Voronkov [20] translate the KIF language. In both cases the translations allowed ATPs to solve difficult problems but it was not clear whether they were in any sense optimal.

Propositional Modal Logic. There exists a standard relational encoding of certain propositional modal logics into first-order logic where the accessibility relation is given as a predicate R and relational correspondence properties are captured as additional axioms. To combat certain deficiencies in this translation, a functional translation method was introduced with different variations [21]. One observation here is that many of the relational correspondence properties are not friendly for proof search and it may be interesting to explore specific ATP extensions that could help with such translations.

4 Improving a Translation

When thinking about translations of problems to first-order logic there are things that can be done to improve ATP performance within the translation and there are things that can only be done within the ATP. I discuss both sets of improvements here.

4.1 What Can be Controlled in an Encoding

The following describes some things to watch out for when designing a translation.

Throwing things away. A common mistake in encodings is to throw away information that the theorem prover can use in proof search. An obvious example of this is performing Skolemisation and dropping information about which function symbols are Skolem functions – in some applications, e.g. inductive theorem proving, such Skolem constants play a special role in proof search. Another example is dropping information about which formula is the goal. A more subtle example of this is to perform subformula naming before passing the problem to Vampire. In this case the original structure is lost and it is not possible for Vampire to choose to name fewer subformulas. Recall that an advantage of the strategy scheduling structure of Vampire is that it can try out various different preprocessing steps. In general, if an ATP can perform a preprocessing step then it should be given the option to.

Not adding what is helpful. Quite often there is some information about a problem that may not affect the solvability of the problem but can be useful in improving the performance of the solver. An obvious example is in the translation of the problem of finding first-order models to EPR [4]. In this translation there are a number of symmetries that are known during the translation but that would be expensive to recover after the fact. Failing to add these gives the solver unnecessary work to do. Another example would be failing to exclude (sets of) axioms that are known to have no relation to the current goal – something that may be known when generating the problem but can be difficult to determine by the ATP.

4.2 What Needs ATP Support

Here I discuss some issues with translations that require ATP support to handle.

Exploding axioms. It is quite common to include an axiomitisation of some theory that is included in the problem but only needed a little bit in proof search. It is also common for these axioms to be explosive, in the sense that they generate a lot of unnecessary consequences in the search space. A typical example would be axioms for arithmetic, or even for equality. We have already seen an example with this when encoding the boolean sort.

One ATP-based solution to this that we have explored [39, 7] is to identify such axioms and limit the depth to which these axioms can interact. This is effectively the same as precomputing the set of consequences of the axioms up to a certain size but happens dynamically at proof search so may not require the full set. An alternative ATP-based solution is to capture the rules represented by the axioms as additional inference rules.

Rewriting the wrong way. Sometimes rules may not do what we want them to do. For example, the following rule for set extensionality

$$(\forall x)(\forall y)(((\forall e)(e \in y \leftrightarrow e \in y)) \rightarrow x = y)$$

will be clausified as

$$f(x,y) \notin x \vee f(x,y) \notin y \vee x = y$$

which is correct but will not be treated in the way we want by the ATP. Literal selection prefers larger literals and dislikes (positive) equalities. Therefore, the literal $x = y$ will not be selected. However, given the goal $a \neq b$ for sets a and b we want this to unify with $x = y$ to generate the necessary subgoals. The rewrite rule is oriented the wrong way. This can also happen with implications or equalities where we want to rewrite something small into something big.

One ATP-based solution is to introduce specific rules to handle special cases. For example, in the case of extensionality Vampire includes the inference rule [16]

$$\frac{x \simeq y \vee C \quad \underline{s \not\simeq t} \vee D}{C\{x \mapsto s, y \mapsto t\} \vee D} \;,$$

where $x \simeq y \vee C$ is identified as an extensionality clause.

Finding the goal. As mentioned previously, preprocessing and proof search can benefit from knowing what the goal of a problem is. Ideally the translation preserves this but the ATP can also attempt to guess the goal based on the location of axioms in the input problem and the frequency of symbols occurring in different parts of the problem [37].

Changing the translation. It can be possible to detect cases where an alternative encoding may be preferable and switch to that encoding via a further translation step. For example, some work [44] in the CVC4 SMT solver looked at identifying datatypes encoding natural numbers and translating these to guarded usage of integers, which the solver has better support for.

Supporting multiple encodings. Ultimately we may want to try various encodings via the strategy scheduling approach. However, if the ATP cannot understand the original problem then it cannot natively support these encodings. The solution here is to extend the input language of the ATP such that it supports additional features that allow the different encodings to take place. This is what we have done with our work on programs [12] and datatypes [29].

5 Comparing Encodings

It is generally hard to evaluate the addition of a new feature to a first-order ATP [41]. The same can be said for encodings. Here I give a few thoughts about how to go about it:

1. *Actually do it.* Without experiments it is impossible to draw any real conclusions. A corollary here is *don't rely on properties of the output of the encoding that you assume are good proxies for performance* e.g. the number of resulting clauses. Whilst bigger problems can pose a problem for ATPs, there are many small (< 50 clauses) problems that ATPs find very challenging.

2. *Use portfolio mode.* The encoding may require certain preprocessing or proof search parameters to be switched on (or off) to be effective. Running in a single mode may miss this and the wrong conclusion may be drawn. Furthermore, the encoding may react positively to multiple different strategies and similarly, without portfolio mode this will be missed. A corollary here is to look at the strategies actually used to solve problems and see if there are any common patterns.

3. *Don't use portfolio mode.* The portfolio modes in a solver are tuned to the current options and assumptions about input problems. They are usually slightly over-fitted. Perhaps the parameter option needed for your encoding is not included. The obvious solution is to search the parameter space for the optimal combination of parameters for a particular encoding. This is what we often do in Vampire but it is very expensive.

4. *Think about the resources.* Do the input problems reflect what you care about; often it is easy to come up with pathological bad cases but optimising for these often makes little practical difference. Is the time limit sensible; for portfolio mode allow minutes (I use 5) and for single strategies allow seconds (I use 10 to 30) but use cases vary. The point here is whether the results reflect actual usage for the problem being targeted – one encoding may work better than another for cases that you don't care about.

6 Conclusion

In this paper I aimed to give some thoughts about the activity of translating problems into first-order logic. My main conclusion is that for some encodings we need ATP support, and this is what we are trying to provide in Vampire. If you want help extending Vampire for a particular encoding please contact me.

References

[1] L. Bachmair and H. Ganzinger. Resolution theorem proving. In A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*, volume I, chapter 2, pages 19–99. Elsevier Science, 2001.

[2] Michael Barnett and K. Rustan M. Leino. To goto where no statement has gone before. In *Verified Software: Theories, Tools, Experiments, Third International Conference, VSTTE 2010, Edinburgh, UK, August 16-19, 2010. Proceedings*, pages 157–168, 2010.

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

[4] Peter Baumgartner, Alexander Fuchs, Hans de Nivelle, and Cesare Tinelli. Computing finite models by reduction to function-free clause logic. *Journal of Applied Logic*, 7(1):58 – 74, 2009. Special Issue: Empirically Successful Computerized Reasoning.

[5] Christoph Benzmüller and Lawrence C Paulson. Multimodal and intuitionistic logics in simple type theory. *Logic Journal of IGPL*, 18(6):881–892, 2010.

[6] Ahmed Bhayat and Giles Reger. Higher-order reasoning vampire style. In *25th Automated Reasoning Workshop*, page 19, 2018.

[7] Ahmed Bhayat and Giles Reger. Set of support for higher-order reasoning. In *PAAR 2018*, 2018.

[8] J. C. Blanchette, S. Böhme, A. Popescu, and N. Smallbone. Encoding monomorphic and polymorphic types. In *Proceedings of the 19th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2013)*, 2013.

[9] Jasmin Christian Blanchette, Sascha Böhme, and Lawrence C. Paulson. Extending sledgehammer with SMT solvers. *J. Autom. Reasoning*, 51(1):109–128, 2013.

[10] Franois Bobot, Jean christophe Fillitre, Claude March, and Andrei Paskevich. Why3: Shepherd your herd of provers. In *In Workshop on Intermediate Veri cation Languages*, 2011.

[11] Koen Claessen, Ann Lillieström, and Nicholas Smallbone. Sort it out with monotonicity - translating between many-sorted and unsorted first-order logic. In *Automated Deduction - CADE-23 - 23rd International Conference on Automated Deduction, Wroclaw, Poland, July 31 - August 5, 2011. Proceedings*, pages 207–221, 2011.

[12] Laura Kovcs Evgenii Kotelnikov and Andrei Voronkov. A FOOLish encoding of the next state relations of imperative programs. In *IJCAR 2018*, 2018.

[13] Tobias Gleißner, Alexander Steen, and Christoph Benzmüller. Theorem provers for every normal modal logic. In *LPAR-21, 21st International Conference on Logic for Programming, Artificial Intelligence and Reasoning, Maun, Botswana, May 7-12, 2017*, pages 14–30, 2017.

[14] Sylvia Grewe, Sebastian Erdweg, and Mira Mezini. Using vampire in soundness proofs of type systems. In *Proceedings of the 1st and 2nd Vampire Workshops, Vampire@VSL 2014, Vienna, Austria, July 23, 2014 / Vampire@CADE 2015, Berlin, Germany, August 2, 2015*, pages 33–51, 2015.

[15] Sylvia Grewe, Sebastian Erdweg, André Pacak, Michael Raulf, and Mira Mezini. Exploration of language specifications by compilation to first-order logic. *Sci. Comput. Program.*, 155:146–172, 2018.

[16] Ashutosh Gupta, Laura Kovcs, Bernhard Kragl, and Andrei Voronkov. Extensional crisis and proving identity. In Franck Cassez and Jean-Franois Raskin, editors, *Automated Technology for Verification and Analysis*, volume 8837 of *Lecture Notes in Computer Science*, pages 185–200. Springer International Publishing, 2014.

[17] Krystof Hoder, Zurab Khasidashvili, Konstantin Korovin, and Andrei Voronkov. Preprocessing techniques for first-order clausification. In *Formal Methods in Computer-Aided Design, FMCAD 2012, Cambridge, UK, October 22-25, 2012*, pages 44–51, 2012.

[18] Kryštof Hoder, Giles Reger, Martin Suda, and Andrei Voronkov. Selecting the selection. In Nicola Olivetti and Ashish Tiwari, editors, *Automated Reasoning: 8th International Joint Conference, IJCAR 2016, Coimbra, Portugal, June 27 – July 2, 2016, Proceedings*, pages 313–329, Cham, 2016. Springer International Publishing.

[19] Krystof Hoder and Andrei Voronkov. Sine qua non for large theory reasoning. In *Automated Deduction - CADE-23 - 23rd International Conference on Automated Deduction, Wroclaw, Poland, July 31 - August 5, 2011. Proceedings*, pages 299–314, 2011.

[20] Ian Horrocks and Andrei Voronkov. Reasoning support for expressive ontology languages using a theorem prover. In *Proceedings of the 4th International Conference on Foundations of Information and Knowledge Systems*, FoIKS'06, pages 201–218, Berlin, Heidelberg, 2006. Springer-Verlag.

[21] Ullrich Hustadt and Renate A. Schmidt. MSPASS: Modal reasoning by translation and first-order resolution. In Roy Dyckhoff, editor, *Automated Reasoning with Analytic Tableaux and Related Methods*, pages 67–71, Berlin, Heidelberg, 2000. Springer Berlin Heidelberg.

[22] Jan Jakubuv, Martin Suda, and Josef Urban. Automated invention of strategies and term orderings for vampire. In *GCAI 2017, 3rd Global Conference on Artificial Intelligence, Miami, FL, USA, 18-22 October 2017.*, pages 121–133, 2017.

[23] D. Knuth and P. Bendix. Simple word problems in universal algebras. In J. Leech, editor,

Computational Problems in Abstract Algebra, pages 263–297. Pergamon Press, Oxford, 1970.

[24] Konstantin Korovin. Inst-Gen – A Modular Approach to Instantiation-Based Automated Reasoning. In Andrei Voronkov and Christoph Weidenbach, editors, *Programming Logics: Essays in Memory of Harald Ganzinger*, pages 239–270, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[25] Konstantin Korovin and Andrei Voronkov. Orienting rewrite rules with the Knuth–Bendix order. *Inf. Comput.*, 183(2):165–186, June 2003.

[26] Evgenii Kotelnikov, Laura Kovács, Giles Reger, and Andrei Voronkov. The vampire and the FOOL. In *Proceedings of the 5th ACM SIGPLAN Conference on Certified Programs and Proofs*, CPP 2016, pages 37–48. ACM, 2016.

[27] Evgenii Kotelnikov, Laura Kovács, Martin Suda, and Andrei Voronkov. A clausal normal form translation for FOOL. In *GCAI 2016. 2nd Global Conference on Artificial Intelligence, September 19 - October 2, 2016, Berlin, Germany*, pages 53–71, 2016.

[28] Evgenii Kotelnikov, Laura Kovács, and Andrei Voronkov. A first class boolean sort in first-order theorem proving and TPTP. In *Intelligent Computer Mathematics - International Conference, CICM 2015, Washington, DC, USA, July 13-17, 2015, Proceedings*, pages 71–86, 2015.

[29] Laura Kovács, Simon Robillard, and Andrei Voronkov. Coming to terms with quantified reasoning. In Giuseppe Castagna and Andrew D. Gordon, editors, *Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages, POPL 2017, Paris, France, January 18-20, 2017*, pages 260–270. ACM, 2017.

[30] Laura Kovács and Andrei Voronkov. First-order theorem proving and Vampire. In *CAV 2013*, volume 8044 of *LNCS*, pages 1–35, 2013.

[31] K. Rustan M. Leino and Philipp Rümmer. A polymorphic intermediate verification language: Design and logical encoding. In *Proceedings of the 16th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, TACAS'10, pages 312–327, Berlin, Heidelberg, 2010. Springer-Verlag.

[32] Michel Ludwig and Uwe Waldmann. An extension of the Knuth-Bendix ordering with LPO-like properties. In *Logic for Programming, Artificial Intelligence, and Reasoning, 14th International Conference, LPAR 2007, Yerevan, Armenia, October 15-19, 2007, Proceedings*, pages 348–362, 2007.

[33] Jia Meng and Lawrence C. Paulson. Translating higher-order clauses to first-order clauses. *Journal of Automated Reasoning*, 40(1):35–60, Jan 2008.

[34] R. Nieuwenhuis and A. Rubio. Paramodulation-based theorem proving. In A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*, volume I, chapter 7, pages 371–443. Elsevier Science, 2001.

[35] Lawrence C. Paulson and Jasmin Christian Blanchette. Three years of experience with sledgehammer, a practical link between automatic and interactive theorem provers. In Geoff Sutcliffe, Stephan Schulz, and Eugenia Ternovska, editors, *IWIL 2010. The 8th International Workshop on the Implementation of Logics*, volume 2 of *EPiC Series in Computing*, pages 1–11. EasyChair, 2012.

[36] Michael Rawson and Giles Reger. Dynamic strategy priority: Empower the strong and abandon the weak. In *PAAR 2018*, 2018.

[37] Giles Reger and Martin Riener. What is the point of an smt-lib problem? In *16th International Workshop on Satisfiability Modulo Theories*, 2018.

[38] Giles Reger and Martin Suda. Measuring progress to predict success: Can a good proof strategy be evolved? *AITP 2017*, 2017.

[39] Giles Reger and Martin Suda. Set of support for theory reasoning. In *IWIL Workshop and LPAR Short Presentations*, volume 1 of *Kalpa Publications in Computing*, pages 124–134. EasyChair, 2017.

[40] Giles Reger, Martin Suda, and Andrei Voronkov. Playing with AVATAR. In P. Amy Felty and Aart Middeldorp, editors, *Automated Deduction - CADE-25: 25th International Conference on*

Automated Deduction, Berlin, Germany, August 1-7, 2015, Proceedings, pages 399–415, Cham, 2015. Springer International Publishing.

[41] Giles Reger, Martin Suda, and Andrei Voronkov. The challenges of evaluating a new feature in Vampire. In Laura Kovács and Andrei Voronkov, editors, *Proceedings of the 1st and 2nd Vampire Workshops*, volume 38 of *EPiC Series in Computing*, pages 70–74. EasyChair, 2016.

[42] Giles Reger, Martin Suda, and Andrei Voronkov. Finding finite models in multi-sorted first-order logic. In Nadia Creignou and Daniel Le Berre, editors, *Theory and Applications of Satisfiability Testing – SAT 2016: 19th International Conference, Bordeaux, France, July 5-8, 2016, Proceedings*, pages 323–341. Springer International Publishing, 2016.

[43] Giles Reger, Martin Suda, and Andrei Voronkov. New techniques in clausal form generation. In Christoph Benzmüller, Geoff Sutcliffe, and Raul Rojas, editors, *GCAI 2016. 2nd Global Conference on Artificial Intelligence*, volume 41 of *EPiC Series in Computing*, pages 11–23. EasyChair, 2016.

[44] Andrew Reynolds and Viktor Kuncak. Induction for SMT solvers. In *Verification, Model Checking, and Abstract Interpretation - 16th International Conference, VMCAI 2015, Mumbai, India, January 12-14, 2015. Proceedings*, pages 80–98, 2015.

[45] 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, *Automated Deduction – CADE-23*, pages 461–475, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[46] S. Schulz. E — a brainiac theorem prover. 15(2-3):111–126, 2002.

[47] Alexander Steen and Christoph Benzmüller. The higher-order prover Leo-III. *CoRR*, abs/1802.02732, 2018.

[48] Geoff Sutcliffe. The TPTP problem library and associated infrastructure. *J. Autom. Reasoning*, 43(4):337–362, 2009.

[49] Geoff Sutcliffe and Evgenii Kotelnikov. TFX: The TPTP extended typed first-order form. In *PAAR 2018*, 2018.

[50] Andrei Voronkov. AVATAR: The architecture for first-order theorem provers. In Armin Biere and Roderick Bloem, editors, *Computer Aided Verification*, volume 8559 of *Lecture Notes in Computer Science*, pages 696–710. Springer International Publishing, 2014.

[51] C. Weidenbach. Combining superposition, sorts and splitting. In A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*, volume II, chapter 27, pages 1965–2013. Elsevier Science, 2001.

[52] Lawrence Wos, George A. Robinson, and Daniel F. Carson. Efficiency and completeness of the set of support strategy in theorem proving. *J. ACM*, 12(4):536–541, October 1965.

Pseudo-Propositional Logic

Ahmad-Saher Azizi-Sultan

Taibah University, Medinah Munawwarah, Saudi Arabia
sultansaher@hotmail.com

Abstract

Propositional logic is the main ingredient used to build up SAT solvers which have gradually become powerful tools to solve a variety of important and complicated problems such as planning, scheduling, and verifications. However further uses of these solvers are subject to the resulting complexity of transforming counting constraints into conjunctive normal form (CNF). This transformation leads, generally, to a substantial increase in the number of variables and clauses, due to the limitation of the expressive power of propositional logic. To overcome this drawback, this work extends the alphabet of propositional logic by including the natural numbers as a means of counting and adjusts the underlying language accordingly. The resulting representational formalism, called pseudo-propositional logic, can be viewed as a generalization of propositional logic where counting constraints are naturally formulated, and the generalized inference rules can be as easily applied and implemented as arithmetic.

1 Introduction

During the last few decades SAT solvers have gained considerable advances and become a tool suitable for attacking more and more practical problems arising in different areas such as formal verification [1, 2, 12], planning [9, 11], scheduling [8], etc. Most of these solvers, if not all, are a variety of Davis-Putnam-Logemann-Loveland (DPLL) algorithm [4, 5] which is based on blind branch and backtrack techniques that explore the search space exhaustively until a solution is found. As SAT is one of the canonical NP-complete problems [3], generally any exhaustive search algorithm results in impractical excessive time complexity.

In order to reduce the size of the search tree, modern SAT solvers such as Chaff, BerkMin, and MiniSAT have equipped the DPLL algorithm with pruning techniques known as backjumping, conflict-driven lemma learning, and restarts [6, 7, 10]. Although these techniques were able to reduce the search space, the major drawback of having blind control over the search process remains.

To mine solutions efficiently, there is a need for a tool that could scan over the search field and detect the spots that potentially contain solutions. Unfortunately, the limited expressive power of propositional logic does not allow for such a tool to be built-in. Furthermore, the input of SAT solvers is, usually, a formula in its CNF. However, many applications contain counting constraints and transforming these constraints into CNF generally leads to a substantial increase in the number of variables and clauses. This is again due to the lack of expressive tools in the underlying propositional language.

This work takes the liberty to extend the alphabet of propositional logic by including the natural numbers as a means of counting and adjusts the underlying language accordingly. The resulting representational formalism, which we can conveniently agree to call pseudo-propositional logic, may be viewed as a generalization of propositional logic, where counting constraints are naturally formulated and at the same time the Boolean nature of the propositional variables is kept preserved. This allows for encoding counting constraints as well as SAT instances much more compact than if it is encoded using CNF. Furthermore, the generalized inference rules are as easily applied and implemented as arithmetic. In such a case, equipping backtracking procedures with some combinatorial techniques allows for assigning truth values to a variety of propositional variables simultaneously. This leads to easily detecting the branches of the search tree that possibly contain solutions, or at least prune the useless ones, allowing for possible improvement in terms of calculation complexity.

2 Language

Definition 1. Let $\mathscr{P} = \{p, q, \cdots\}$ be a finite or countably infinite set of propositional symbols and $\mathbb{N}$ be the natural numbers. The alphabet $\mathscr{A}$ underlying the language of pseudo-propositional formulas is defined as $\mathscr{A} = \mathscr{P} \cup \mathbb{N} \cup \{\neg, +, (,)\}$, where $\{\neg, +, (,)\}$ resemble the negation, addition, opening and closing punctuation symbols, respectively.

Definition 2. The language or formulas of pseudo-propositional logic, symbolized by $\mathscr{F}$, is defined recursively as follows:

- If $p \in \mathscr{P}$ and $n \in \mathbb{N}$ then $np \in \mathscr{F}$, called prime formula.
- If $\alpha, \beta \in \mathscr{F}$, then $(\alpha + \beta)$, $\neg\alpha \in \mathscr{F}$.

A prime formula or its negation is called a *literal*. A formula which is a literal or an addition of two or more literals is said to be in *normal form*. A *subformula of a formula* α is a substring occurring in α, which is itself a formula.

Before proceeding further, let us agree upon the following two conventions to ease readability:

- Propositional variables or symbols will be denoted by $p, q, \ldots$, formulas by $\alpha, \beta, \varphi, \ldots$, set of formulas by $F, G, \ldots$, and set of queries, which will be defined in section 4, by $\boldsymbol{F}, \boldsymbol{G}, \ldots$, where these letters may also be indexed.
- As in arithmetical terms, parenthesis are omitted whenever it is possible.

3 Semantics

A proposition can have only one of the *truth values*, true or false. Conveniently to the context of this work, these values are represented by $(1,0)$ for true and $(0,1)$ for false. More formally, this is rephrased by the definition of interpretation.

Definition 3. An interpretation I is a subset of $\mathscr{P}$ represented by the mapping $\phi : \mathscr{P} \to \{(1,0), (0,1)\}$ which is defined as follows:

$$\phi(p) = \begin{cases} (1,0) & \text{if } p \in I, \\ (0,1) & \text{if } p \notin I. \end{cases}$$

Thus, the interpretation I is the subset of $\mathscr{P}$ containing only those propositional symbols that are mapped to $(1,0)$ under ϕ. That is

$$I = \{p \in \mathscr{P} \mid \phi(p) = (1,0)\}. \tag{1}$$

Recursively, the mapping ϕ can be extended to become from the set of formulas $\mathscr{F}$ to $\mathscr{M} = \mathbb{Z}^2$ which is the *meaning set* in the context of pseudo-propositional logic. In order to do so the following two functions are prerequisites:

- Negation $\neg^* : \mathscr{M} \to \mathscr{M}$ where $\neg^*(n,m) = (m,n)$.
- Addition[1] $+ : \mathscr{M} \times \mathscr{M} \to \mathscr{M}$, where $+((n,m),(k,l)) = (n+k, m+l)$.

Yet every interpretation I defines recursively its own mapping $I : \mathscr{F} \to \mathscr{M}$ as follows:

[1] This addition is easily distinguished from the addition of formulas in definition 1.

1) **Recursion base.** Recall that if φ is an atom then $\varphi = n\,p$ for some $n \in \mathbb{N}$ and $p \in \mathscr{P}$. Consequently the recursion base reads
$$I(\varphi) = I(n\,p) = n\,\phi(p).$$

2) **Recursion steps.**

$$I(\alpha) = \begin{cases} \neg^*(I(\beta)) & \text{if } \alpha \text{ is of the form } \neg\beta, \\ I(\beta_1) + I(\beta_2) & \text{if } \alpha \text{ is of the form } \beta_1 + \beta_2. \end{cases}$$

After assigning meanings to formulas, one can investigate how formulas are related to each other according to their meanings.

Definition 4. Two formulas α and β are *equivalent*, in symbols $\alpha \equiv \beta$, iff $I(\alpha) = I(\beta)$ for every interpretation I.

Example 1. *For any $\alpha,\ \beta \in \mathscr{F}$, it is obvious that $\alpha + \beta \equiv \beta + \alpha$ and $\neg\neg\alpha \equiv \alpha$.*

Proposition 1. *For any $\alpha,\ \beta \in \mathscr{F}$, the equivalence $\neg(\alpha+\beta) \equiv \neg\alpha + \neg\beta$ holds.*

Proof. Given an interpretation I suppose that $I(\alpha) = (n,l)$ and $I(\beta) = (m,k)$.

$$\begin{aligned} I(\neg(\alpha+\beta)) &= \neg^* I(\alpha+\beta) = \neg^*(I(\alpha)+I(\beta)) = \neg^*((n,l)+(m,k)) \\ &= \neg^*(n+m, l+k) = (l+k, n+m). \end{aligned}$$

On the other hand,

$$\begin{aligned} I(\neg\alpha+\neg\beta) &= I(\neg\alpha)+I(\neg\beta) = \neg^* I(\alpha) + \neg^* I(\beta)) = \neg^*(n,l) + \neg^*(m,k) \\ &= (l,n)+(k,m) = (l+k, n+m). \end{aligned}$$

Thus $I(\neg(\alpha+\beta)) = I(\neg\alpha+\neg\beta)$ for any given interpretation I. □

Proposition 2. *If $p \in \mathscr{P}$ and $n,m \in \mathbb{N}$ then $(n\,p + m\,p) \equiv (n+m)p$.*

Proof. Let I be an interpretation then,

$$I(np+mp) = I(np)+I(mp) = n\phi(p)+m\phi(p) = (n+m)\phi(p) = I((n+m)p).$$

□

Obviously, $\equiv$ is an equivalence relation. Moreover, it is a *congruence relation* on $\mathscr{F}$, i.e., for all $\alpha_1, \alpha_2, \beta_1, \beta_2 \in \mathscr{F}$,

$$\alpha_1 \equiv \alpha_2, \beta_1 \equiv \beta_2 \Rightarrow \neg\alpha_1 \equiv \neg\alpha_2, \alpha_1+\beta_1 \equiv \alpha_2+\beta_2. \tag{2}$$

For this reason the *replacement theorem* holds. It enables one to substitute a subformula β, of a formula α, by an equivalent one without altering the meaning of α. If we let $\alpha\,[\beta_1/\beta_2]$ denote the formula that is obtained from α by substituting every occurrence of β_1 by β_2, the replacement theorem becomes as follows:

Theorem 1. *If the formulas β_1 and β_2 are equivalent, so are α and $\alpha\,[\beta_1/\beta_2]$.*

Proof by induction on α. Suppose α is a prime formula. Then, for both cases $\alpha = \beta_1$ and $\alpha \neq \beta_1$ we clearly have $\alpha \equiv \alpha\,[\beta_1/\beta_2]$. Now let $\alpha = \alpha_1 + \alpha_2$. If $\alpha = \beta_1$ then trivially $\alpha \equiv \alpha\,[\beta_1/\beta_2]$ holds. Otherwise $\alpha\,[\beta_1/\beta_2] = \alpha_1\,[\beta_1/\beta_2] + \alpha_2\,[\beta_1/\beta_2]$. By the induction hypothesis we have $\alpha_1 \equiv \alpha_1\,[\beta_1/\beta_2]$ and $\alpha_2 \equiv \alpha_2\,[\beta_1/\beta_2]$. According to the congruence property 2 one concludes that

$$\alpha = (\alpha_1 + \alpha_2) \equiv \alpha_1\,[\beta_1/\beta_2] + \alpha_2\,[\beta_1/\beta_2] = \alpha\,[\beta_1/\beta_2]\,.$$

The induction steps for $\neg$ follows analogously. □

Taking into account that one can eliminate all negation signs except those in front of prime formulas by Proposition 1, the replacement theorem transforms every formula into an equivalent one which is a normal form. This is actually interesting from an implementational point of view, as efficiency might be gained by restricting the inference rules to the mentioned normal form.

After assigning meaning to formulas and seeing how they are related, it is time to consider counting constraints which are represented by queries defined in the upcoming section.

4 Queries and Models

Definition 5. For a given formula $\varphi \in \mathscr{F}$ and an $n \in \mathbb{N}$, in pseudo-propositional logic we are interested in finding an answer to the query: is there an interpretation I such that $I(\varphi) = (m,l)$ where $m \geq n$. Every formula φ combined with a natural number n forms a query $\varphi^{(n)}$. The set of all possible queries is denoted by $\mathscr{Q}$. That is $\mathscr{Q} = \{\varphi^{(n)} : \varphi \in \mathscr{F}, n \in \mathbb{N}\}$.

Having defined queries, modelling becomes straightforward. Simply, it defines relations between interpretations and queries.

Definition 6. It is said that an interpretation I is a model for a query $\varphi^{(n)}$, in symbols $I \models \varphi^{(n)}$, iff $I(\varphi) = (\bar{n}, l)$ with $\bar{n} \geq n$. Considering a set of queries $\boldsymbol{Q}$, it is said that I is a model for $\boldsymbol{Q}$, and symbolised by $I \models \boldsymbol{Q}$, iff $I \models \varphi^{(n)}$ for every query $\varphi^{(n)} \in \boldsymbol{Q}$.

It is time now to start reasoning which is as easy as arithmetic. The coming proposition is an ideal example of reasoning in pseudo-propositional logic.

Proposition 3. *Let $\alpha^{(n)}, \varphi^{(m)} \in \mathscr{Q}$. If I is an interpretation such that $I \models \alpha^{(n)}$ and $I \models \varphi^{(m)}$, then $I \models (\alpha+\varphi)^{(n+m)}$.*

Proof. Since $I \models \alpha^{(n)}$ and $I \models \varphi^{(m)}$, this implies that $I(\alpha) = (\bar{n}, l)$, $\bar{n} \geq n$ and $I(\varphi) = (\bar{m}, k)$, $\bar{m} \geq m$. Thus

$$I(\alpha+\varphi) = I(\alpha) + I(\varphi) = (\bar{n}, l) + (\bar{m}, k) = (\bar{n}+\bar{m}, l+k).$$

Since $\bar{n}+\bar{m} \geq n+m$ we conclude that $I \models (\alpha+\varphi)^{(n+m)}$. □

One can easily conceive that satisfiability in pseudo-propositional logic concerns queries rather than formulas.

- It is said that $\varphi^{(n)}$ (resp. $\boldsymbol{Q}$) is *satisfiable* iff there exists an interpretation I such that $I \models \varphi^{(n)}$ (resp. $I \models \boldsymbol{Q}$).
- It is said that $\varphi^{(n)}$ (resp. $\boldsymbol{Q}$) is *unsatisfiable* iff for every interpretation I we have $I \not\models \varphi^{(n)}$ (resp. $I \not\models \boldsymbol{Q}$).

Consequently the equivalence relation is lifted to the level of queries as demonstrated below.

5 Consequence and Equivalence

Definition 7. $\boldsymbol{Q}$ is a logical consequence of $\boldsymbol{F}$, written $\boldsymbol{F} \models \boldsymbol{Q}$, if $I \models \boldsymbol{Q}$ for every interpretation I that is a model for $\boldsymbol{F}$. In short, $I \models \boldsymbol{F} \Rightarrow I \models \boldsymbol{Q}$ for all interpretations I.

Definition 8. If $\boldsymbol{F} \models \boldsymbol{Q}$ and $\boldsymbol{Q} \models \boldsymbol{F}$ then we say $\boldsymbol{F}$ and $\boldsymbol{Q}$ are semantically equivalent. We denote this by $\boldsymbol{F} \equiv \boldsymbol{Q}$.

In this work, $\boldsymbol{F} \models \alpha^{(n)}$ (resp. $\alpha^{(n)} \models \boldsymbol{F}$) will mean $\boldsymbol{F} \models \{\alpha^{(n)}\}$ (resp. $\{\alpha^{(n)}\} \models \boldsymbol{F}$). More generally, we write $\boldsymbol{F} \models \alpha_1^{(n_1)}, \alpha_2^{(n_2)}, \ldots, \alpha_k^{(n_k)}$ (resp. $\alpha_1^{(n_1)}, \alpha_2^{(n_2)}, \ldots, \alpha_k^{(n_k)} \models \boldsymbol{F}$) instead of $\boldsymbol{F} \models \{\alpha_1^{(n_1)}, \alpha_2^{(n_2)}, \ldots, \alpha_k^{(n_k)}\}$ (resp. $\{\alpha_1^{(n_1)}, \alpha_2^{(n_2)}, \ldots, \alpha_k^{(n_k)}\} \models \boldsymbol{F}$), and more briefly we write $\boldsymbol{F}, \alpha^{(n)} \models \varphi^{(m)}$ instead of $\boldsymbol{F} \cup \{\alpha^{(n)}\} \models \varphi^{(m)}$. Analogous notation will be used regarding semantic equivalence.

Note 1. *Proposition 3 can be rewritten as follows:*

$$\alpha^{(n)}, \varphi^{(m)} \models (\alpha + \varphi)^{(n+m)}.$$

Example 2. *Let* $\boldsymbol{Q} = \{\varphi^{(i)} : i = 1, 2, \ldots, n-1\}$. *Moreover suppose that* $I \models \varphi^{(n)}$. *This means that* $I(\varphi) = (\bar{n}, l)$ *where* $\bar{n} \geq n > i$. *Consequently,* $I \models \varphi^{(i)}$ *for all* $i < n$. *Thus* $\varphi^{(n)} \models \boldsymbol{Q}$.

Lemma 1. *Let* φ *be a formula and* $n > 1$ *then* $(\varphi + p + \neg p)^{(n)} \models \varphi^{(n-1)}$.

Proof. Suppose $I \models (\varphi + p + \neg p)^{(n)}$. This means that $I(\varphi + p + \neg p) = I(\varphi) + (1,1) = (\bar{n}, l)$ where $\bar{n} \geq n$ and $l \geq 1$. Since $(\mathbb{Z}^2, +)$ is an abelian group we conclude that $I(\varphi) = (\bar{n} - 1, l - 1)$. That is $I \models \varphi^{(n-1)}$. □

An obvious generalization and a direct consequence of lemma 1 is the resolution theorem which reads:

Theorem 2. *If* $\varphi \in \mathscr{F}$ *and* $n > \bar{m} \geq m$, *then the following consequence holds:*

$$(\varphi + \bar{m}p + \neg(mp))^{(n)} \models (\varphi + (\bar{m} - m)p)^{(n-m)}.$$

To keep things from being complicated before going any further, the following theorem must be proven.

Theorem 3. *If the formulas* β_1 *and* β_2 *are equivalent, so are the queries* $\alpha^{(n)}$ *and* $(\alpha[\beta_1/\beta_2])^{(n)}$ *for every* $n \in \mathbb{N}$.

Proof. Suppose that $I \models \alpha^{(n)}$. This means that $I(\alpha) = (\bar{n}, l)$ where $\bar{n} \geq n$. Since $\beta_1 \equiv \beta_2$, from Theorem 1 it follows that $\alpha \equiv \alpha[\beta_1/\beta_2]$. Thus $I(\alpha) = I(\alpha[\beta_1/\beta_2]) = (\bar{n}, l)$ where $\bar{n} \geq n$. Thus $I \models (\alpha[\beta_1/\beta_2])^{(n)}$. We conclude that $\alpha^{(n)} \models (\alpha[\beta_1/\beta_2])^{(n)}$. Taking into account that $(\alpha[\beta_1/\beta_2])[\beta_2/\beta_1] = \alpha$, the consequence $(\alpha[\beta_1/\beta_2])^{(n)} \models \alpha^{(n)}$ follows analogously. □

If we call a query with a normal form formula a *normal form query*, then theorems 1 and 3 transform any query into an equivalent one which is a normal form. Thus to solve the satisfiability problem in pseudo-proposition logic it suffices to consider only the sets of queries that are normal form.

Finally, to start applying pseudo-propositional logic to solve real-world problems, such as SAT for example, one more theorem is needed.

Theorem 4. *If* $\boldsymbol{F} \models \boldsymbol{Q}$ *and for every interpretation* I *which models* $\boldsymbol{Q}$ *we have* $I \not\models \boldsymbol{F}$, *then* $\boldsymbol{F}$ *is unsatisfiable.*

Proof. Suppose that $\boldsymbol{F}$ is satisfiable. This means that there exists an interpretation I such that $I \models \boldsymbol{F}$. Consequently, $I \models \boldsymbol{Q}$ since $\boldsymbol{F} \models \boldsymbol{Q}$. This contradicts the theorem's hypothesis. □

Informally speaking, before solving a SAT problem for a given set of queries $\boldsymbol{F}$, Theorem 4 tempts one to use proper inference rules to find a simpler set of queries $\boldsymbol{Q}$ such that $\boldsymbol{F} \models \boldsymbol{Q}$. Now it is enough to look for a solution for $\boldsymbol{F}$ among only those solutions that solve $\boldsymbol{Q}$.

Furthermore any algorithm that solves SAT problem in pseudo-propositional logic can be used to solve the SAT problem of propositional logic. Actually pseudo-propositional logic can be viewed as a generalization of propositional logic. This is justified by the fact that every formula or sentence S in propositional logic can be represented equivalently by a set of queries $\boldsymbol{Q}$ in pseudo-propositional logic. To see this let $\mathscr{P} = \{p_1, p_2, p_3, \cdots\}$ be our set of propositional symbols and let the literal l_i be either p_i or its negation. Moreover, suppose that converting the sentence S into its CNF results in the set of clauses $\{C_i : i = 1, 2, \ldots, n\}$ where each clause C_i is the disjunctions of a set of literals $\{l_j : j \in J_i \subset \mathbb{N}\}$. If we denote to S in its CNF by S_{CNF} we conclude that

$$S_{CNF} = \bigwedge_{i=1}^{n} C_i = \bigwedge_{i=1}^{n} \left(\bigvee_{j \in J_i} l_j \right).$$

Clearly each clause C_i which has the form

$$C_i = \bigvee_{j \in J_i} l_j$$

can be equivalently represented in pseudo-propositional logic by the query

$$Q_i = \left(\sum_{j \in J_i} l_j \right)^{(1)}.$$

If we let $\boldsymbol{Q} = \{Q_i : i = 1, 2, \ldots, n\}$ then the sentence S, which is equivalent to S_{CNF}, is satisfiable iff $\boldsymbol{Q}$ is satisfiable. Moreover, an interpretation I is a model for $\boldsymbol{Q}$ iff I is a model for S. A detailed example is presented in the sequel.

6 Application on SAT

This section is not meant to present an algorithm that competes with the current SAT solvers. It just gives an idea of how one can make use of pseudo-propositional logic to solve problems such as SAT. This is actually done in three steps. Intuitively, the first step involves transforming the given SAT instance into the corresponding set of queries $\boldsymbol{F}$ in pseudo-propositional logic. The second step reprocesses the set $\boldsymbol{F}$ using inference rules to generate a proper compact set of queries $\boldsymbol{Q}$ such that $\boldsymbol{F} \models \boldsymbol{Q}$. Finally, apply a backtracking procedure to find a solution for $\boldsymbol{Q}$ and check if it satisfies the original SAT instance. The final step is repeated until a solution is found or the problem is unsatisfiable otherwise.

Example 3. *Consider the following CNF instance:*

$$\begin{aligned}
&x_1 \vee x_2 \vee x_3,\ \neg x_1 \vee \neg x_2,\ \neg x_1 \vee \neg x_3,\ \neg x_2 \vee \neg x_3,\\
&x_1 \vee x_2 \vee x_4,\ \neg x_1 \vee \neg x_2,\ \neg x_1 \vee \neg x_4,\ \neg x_2 \vee \neg x_4,\\
&x_1 \vee x_3 \vee x_4,\ \neg x_1 \vee \neg x_3,\ \neg x_1 \vee \neg x_4,\ \neg x_3 \vee \neg x_4,\\
&x_2 \vee x_3 \vee x_4,\ \neg x_2 \vee \neg x_3,\ \neg x_2 \vee \neg x_4,\ \neg x_3 \vee \neg x_4.
\end{aligned}$$

This can be equivalently represented by a set of queries in pseudo-propositional logic as follows:

$$\begin{aligned}\boldsymbol{F} = \{ \quad & (x_1+x_2+x_3)^{(1)}, (\neg x_1+\neg x_2)^{(1)}, (\neg x_1+\neg x_3)^{(1)}, (\neg x_2+\neg x_3)^{(1)},\\ & (x_1+x_2+x_4)^{(1)}, (\neg x_1+\neg x_2)^{(1)}, (\neg x_1+\neg x_4)^{(1)}, (\neg x_2+\neg x_4)^{(1)},\\ & (x_1+x_3+x_4)^{(1)}, (\neg x_1+\neg x_3)^{(1)}, (\neg x_1+\neg x_4)^{(1)}, (\neg x_3+\neg x_4)^{(1)},\\ & (x_2+x_3+x_4)^{(1)}, (\neg x_2+\neg x_3)^{(1)}, (\neg x_2+\neg x_4)^{(1)}, (\neg x_3+\neg x_4)^{(1)}\}.\end{aligned}$$

Taking into account Proposition 2 while adding each consecutive homogeneous[2] *couple of queries in* $\boldsymbol{F}$ *results in*

$$\begin{aligned}\boldsymbol{F}_1 = \{ \quad & (2x_1+2x_2+x_3+x_4)^{(2)}, (\neg 2x_1+\neg x_2+\neg x_3)^{(2)}, (\neg x_1+\neg 2x_2+\neg x_3)^{(2)},\\ & (\neg x_1+\neg x_2+\neg 2x_4)^{(2)}, (x_1+x_2+2x_3+2x_4)^{(2)}, (\neg 2x_1+\neg x_3+\neg x_4)^{(2)},\\ & (\neg x_2+\neg 2x_3+\neg x_4)^{(2)}, (\neg x_2+\neg x_3+\neg 2x_4)^{(2)}\}.\end{aligned}$$

One can easily conceive that $\boldsymbol{F} \equiv \boldsymbol{F}_1$ *which shows how encoding in pseudo-propositional logic is much more compact than it is in CNF. Although it is not always the case that* $\boldsymbol{F} \equiv \boldsymbol{F}_1$ *but we, at least, know from Proposition 3 that* $\boldsymbol{F} \models \boldsymbol{F}_1$*. If we add again and again each consecutive homogeneous couple of queries in* $\boldsymbol{F}_1$*, we finally get*

$$\boldsymbol{F}_2 = \{(3x_1+3x_2+3x_3+3x_4)^{(4)}, (\neg 6x_1+\neg 6x_2+\neg 6x_3+\neg 6x_4)^{(12)}\},$$

as a logical consequence of $\boldsymbol{F}_1$*. By backtracking and propagation,* $\boldsymbol{F}_2$ *has only the following six interpretations:*

$$\begin{aligned} I_1 = \{x_1,x_2\}, I_2 = \{x_1,x_3\}, I_3 = \{x_1,x_4\},\\ I_4 = \{x_2,x_3\}, I_5 = \{x_2,x_4\}, I_6 = \{x_3,x_4\}.\end{aligned}$$

Since non of these interpretations model $\boldsymbol{F}$*, according to Theorem 4* $\boldsymbol{F}$ *and consequently the original SAT instance are unsatisfiable.*

7 Conclusion and Further Work

This work has introduced pseudo-propositional logic, a generalization of propositional logic with considerable extension of its expressive power. This enables the resulting representational formalism to encode counting constraints as well as SAT instances naturally, and much more compact than the encoding using CNF. The inference rules of the resulting pseudo-propositional logic, besides their Boolean nature, have arithmetical flavour allowing for easy implementation. Moreover, as it was seen in Example 3, applying backtracking on the final entailed set of queries may yield simultaneous multi-variables guesses, eliminating considerable parts of the search tree that have not been yet explored and hence allowing for potential improvement in terms of time complexity. Another promising improvement is subject to a further investigation on how to construct a compact proper final entailed set of queries which maximizes the eliminated part of the search tree and at the same time captures all possible solutions of the original problem.

8 Acknowledgments

I would like to thank Prof. Steffen Hölldobler, Director of the International Center for Computational Logic, Dresden, Germany. I have learned from him how to rationalise logically rather than just thinking mathematically. Without his influence this work would not have been established.

[2] containing literals of the same polarity

References

[1] A. Biere, A. Cimatti, E. M. Clarke, M. Fujita, and Y. Zhu. Symbolic model checking using sat procedures instead of bdds. In *Proceedings of the 36th Annual ACM/IEEE Design Automation Conference*, DAC '99, pages 317–320, New York, NY, USA, 1999. ACM.

[2] Armin Biere, Alessandro Cimatti, Edmund M. Clarke, and Yunshan Zhu. Symbolic model checking without bdds. In *Proceedings of the 5th International Conference on Tools and Algorithms for Construction and Analysis of Systems*, TACAS '99, pages 193–207, London, UK, UK, 1999. Springer-Verlag.

[3] Stephen A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the 3rd Annual ACM Symposium on Theory of Computing, May 3-5, 1971, Shaker Heights, Ohio, USA*, pages 151–158, 1971.

[4] Martin Davis, George Logemann, and Donald W. Loveland. A machine program for theorem-proving. *Commun. ACM*, 5(7):394–397, 1962.

[5] Martin Davis and Hilary Putnam. A computing procedure for quantification theory. *J. ACM*, 7(3):201–215, 1960.

[6] Niklas Eén and Niklas Sörensson. An extensible sat-solver. In *Theory and Applications of Satisfiability Testing, 6th International Conference, SAT 2003. Santa Margherita Ligure, Italy, May 5-8, 2003 Selected Revised Papers*, pages 502–518, 2003.

[7] Eugene Goldberg and Yakov Novikov. Berkmin: A fast and robust sat-solver. *Discrete Applied Mathematics*, 155(12):1549 – 1561, 2007. SAT 2001, the Fourth International Symposium on the Theory and Applications of Satisfiability Testing.

[8] Carla P. Gomes, Bart Selman, Ken McAloon, and Carol Tretkoff. Randomization in backtrack search: Exploiting heavy-tailed profiles for solving hard scheduling problems. In *Proceedings of the Fourth International Conference on Artificial Intelligence Planning Systems, Pittsburgh, Pennsylvania, USA, 1998*, pages 208–213, 1998.

[9] Henry Kautz and Bart Selman. Pushing the envelope: Planning, propositional logic, and stochastic search. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence - Volume 2*, AAAI'96, pages 1194–1201. AAAI Press, 1996.

[10] Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik. Chaff: Engineering an efficient SAT solver. In *Proceedings of the 38th Design Automation Conference, DAC 2001, Las Vegas, NV, USA, June 18-22, 2001*, pages 530–535, 2001.

[11] Stuart J. Russell and Peter Norvig. *Artificial intelligence - a modern approach, 2nd Edition*. Prentice Hall series in artificial intelligence. Prentice Hall, 2003.

[12] 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, 2003.

A Simple Semi-automated Proof Assistant for First-order Modal Logics

Tomer Libal

The American University of Paris, France
tlibal@aup.edu

Abstract

Most theorem provers and proof assistants are written in imperative or functional programming languages. Recently, the claim that higher-order logic programming languages might be better suited for this task was revisited and a new interpreter, as well as new proof assistants based on it, were introduced. In this paper I follow these results and describe a concise implementation of a prototype for a semi-automated proof assistant for first-order modal logics. The aim of this paper is to encourage the development of personal proof assistants and semi-automated provers for a variety of modal logics.

1 Introduction

Proof assistants are sophisticated systems which have helped users to prove a wide range of mathematical theorems [8, 20, 21, 23] and program properties [3, 11, 25]. Nevertheless, these tools normally require knowledge of computational logic, mathematical skills and experience with the chosen tool. In addition, these tools are based on specific theories, such as intuitionistic type theory for Coq [1] or higher-order logic for Isabelle/HOL [35] and HOL Light [24], which might not be easily applicable to other domains, such as to first-order modal logics.

Benzmüller and Wolzenlogel Paleo have shown that by embedding higher-order modal logics in Coq [4], one can interactively search for proofs. A general description of their work with respect also to other proof assistants is described in [6]. Such an approach takes advantage of the full power of a leading proof assistant and is also clearly general and applicable to other domains. Possible downsides are the Coq expertise required, the required knowledge in intuistionistic type theory for extensions as well as the fact that despite being shallow, an embedding is still an indirect way of communicating with the target calculus - modal logic in our case.

Except the above mentioned works, very little progress has been made towards using proof assistants for modal logics. One reason for that is that proof assistants are non-trivial software requiring a high level of programming skills. Therefore, the majority of proof assistants are implemented in functional programming languages which facilitate their creation. Still, programmers of any proof assistant must handle a variety of common but non-trivial tasks such as proof search, unification, substitutions and many others. Therefore, it is not surprising that one of the leading theorem provers for first-order modal logics is MleanCoP, which is written in Prolog [36]. Prolog gives programmers proof search and other operations for free and allow for a more concise and trusted code. Still, the fact that Prolog is based on first-order logic necessarily means that it is not suitable for a shallow embedding of systems whose meta-theory requires higher-order logic. Among such systems are first-order classical and modal logics. Such embeddings would require a programming language which supports higher-order features, such as binding and higher-order unification.

Advocates of higher-order logic programming languages, such as Felty and Miller [18] have argued that these languages are very suited for the creation of proof assistants [17, 30] and proof checkers [29]. Higher-order logic programming languages provide a native support for all of the

required tasks just mentioned and offer, therefore, not only a much easier coding experience but also an increased level of trust in its correctness. More recently, Sacerdoti-Coen, Tassi, Dunchev and Ferruccio have developed an efficient interpreter [16] for the higher-order logic programming language λProlog [31] and used it for the creation of several proof assistants [15, 22, 38]. They showed that using higher-order logic programming greatly reduces the size of the program. While there are many similarities between their work and the current paper, I am interested in utilizing logic programming for the creation of many, simple and personalized proof assistants and not for the implementation of full scale, general and complex ones.

Another complexity arising in the creation of proof assistants is the need to interface between the users and the tool. The calculi at the core of most proof assistants do not support, out of the box, interactive proof search. Focused sequent calculi [2] partially solve this problem by separating proof search into two different modes. One of the modes, which can be executed fully automatically, can be applied eagerly in order to save the user from tasks not requiring her attention. The assistant then switches to the second mode when user interaction is required.

The above discussion identifies two issues. First, when one wants to use a proof assistant for modal logics, she needs to have both a good proficiency with the current ones and the ability to embed her logic in their theories. Second, if she chooses to implement her own, the task is far from being simple. In this paper I want to present a third alternative – implement your own proof assistant by following a precise recipe. As we will see, the advantages of this approach are the simplicity of the process – I exemplify that by the implementation of a proof assistant for first-order modal logic which consists of less than 200 lines of code. There are also several disadvantages, the most important of which is the need to have a proof calculus based on focused sequents. Another disadvantage is the required λProlog skill. It should be noted though, that the vast majority of the code requires only proficiency in Prolog.

The technique demonstrated in this paper, of using a focused calculus together with higher-order logic programming, is based on the work by Miller and his group towards proof certification [29, 13, 12]. Given that proof certification can be considered as a restriction of interactive proof search where the interaction is done between the proof certificate and the program, this paper attempts to generalize the approach to arbitrary interaction including the interaction between a user and a program. At the same time, using higher-order logic programming towards the creation of proof assistants is one of the purposes of the group behind the ELPI interpreter [15, 22, 38]. Their work is focused on the implementation and extension of fully fledged proof assistants.

My main aim though, is the application of this approach to the creation of proof assistants in domains where proof automation is lacking, such as in modal logic and in fields such as law [37]. I propose that by using λProlog and focusing, any user can implement and customize her theorem prover to meet her needs. The proof assistant for first-order modal logics described in this paper is based on the existence of a focused sequent calculus for this logic. I have therefore obtained such a calculus by the combination of two existing ones. The next section focuses on its presentation. The following section then introduces the other technology I use – higher-order logic programming. I then describe the implementation of a proof assistant for first-order modal logic based on these technologies and give examples of usage and extension. I finish with a short conclusion and mention some possible future work.

2 Focused sequent calculus for first-order modal logics

Theorem provers are often implemented using efficient proof calculi, like, e.g., resolution, combined with the additional use of heuristics and optimization techniques. The use of these

techniques together with required operations such as unification and search leads to a lower degree of trust. On the other hand, traditional proof calculi, like the sequent calculus, rely on less meta-theory and enjoy a higher degree of trust but are quite inefficient for proof search. In order to use the sequent calculus as the basis of automated deduction, much more structure within proofs needs to be established. Focused sequent calculi, first introduced by Andreoli [2] for linear logic, combine the higher degree of trust of sequent calculi with a more efficient proof search. They take advantage of the fact that some of the rules are "invertible", i.e., can be applied without requiring backtracking, and that some other rules can "focus" on the same formula for a batch of deduction steps.

In this paper, I will combine two different focused sequent calculi in order to obtain a sound and complete system for first-order modal logic for K with constant domains and rigid designation. This means that each quantified variable denotes the same element in all worlds and in addition, that the domain for quantification in each world is the same. Please refer to [9] for more information. The existence of focused systems for some other modal logics [27] suggests that the approach described in this paper can be extended. The syntax for first-order modal formulas contains atomic predicates $P(t_1, \ldots, t_n)$, the usual first-order connectives and quantifiers as well as the modal operators $\Box$ and $\Diamond$. The first system I will use is the focused first-order sequent calculus (*LKF*) system defined in [26]. I will combine it with the focused sequent calculus for propositional modal logic for K defined in [33]. This calculus is based on labeled sequents.

The basic idea behind labeled proof systems for modal logic is to internalize elements of the corresponding Kripke semantics into the syntax. The *LMF* system defined in [33] is a sound and complete system for a variety of propositional modal logics. Fig. 1 presents the combined system, LMF^1.

Sequents in LMF^1 have the form $\mathcal{G} \vdash \Theta \Downarrow x : B$ or $\mathcal{G} \vdash \Theta \Uparrow \Gamma$, where the relational set (of the sequent) $\mathcal{G}$ is a set of relational atoms, $x : B$ is a labeled formula (see below) and Θ and Γ are multisets of labeled formulas.

Formulas in LMF^1 which are expressed in negation normal form, can have either positive or negative polarity and are constructed from atomic formulas, whose polarity has to be assigned, and from logical connectives whose polarity is pre-assigned. The choice of polarization does not affect the provability of a formula, but it can have a big impact on proof search and on the structure of proofs: one can observe, e.g., that in *LKF* the rule for $\vee^-$ is invertible while the one for $\vee^+$ is not. The connectives $\wedge^-, \vee^-$, $\Box$ and $\forall$ are of negative polarity, while $\wedge^+, \vee^+$, $\Diamond$ and $\exists$ are of positive polarity. A composed formula has the same polarity of its main connective. In order to polarize literals, we are allowed to fix the polarity of atomic formulas in any way we see fit. We may ask that all atomic formulas are positive, that they are all negative, or we can mix polarity assignments. In any case, if A is a positive atomic formula, then it is a positive formula and $\neg A$ is a negative formula: conversely, if A is a negative atomic formula, then it is a negative formula and $\neg A$ is a positive formula. The basic entities of the calculus are labelled formulas – $x : F$ – which attach to each formula F a label x which denotes the world F is true in.

Deductions in *LKF* are constructed by synchronous and asynchronous phases. A synchronous phase, in which sequents have the form $\mathcal{G} \vdash \Theta \Downarrow x : B$, corresponds to the application of synchronous rules to a specific positive formula B under focus (and possibly its immediate positive subformulas). An asynchronous phase, in which sequents have the form $\mathcal{G} \vdash \Theta \Uparrow \Gamma$, consists in the application of invertible rules to negative formulas contained in Γ (and possibly their immediate negative subformulas). Phases can be changed by the application of the *release* rule. In order to simplify the implementation and the representation, I have excluded the `cut` rule from the calculus. The admissibility of this rule in *LKF* means that while proofs might be

Asynchronous introduction rules

$$\frac{\mathcal{G} \vdash \Theta \Uparrow x : A, \Gamma \quad \mathcal{G} \vdash \Theta \Uparrow x : B, \Gamma}{\mathcal{G} \vdash \Theta \Uparrow x : A \wedge^- B, \Gamma} \wedge_K^- \qquad \frac{\mathcal{G} \vdash \Theta \Uparrow x \colon A, x \colon B, \Gamma}{\mathcal{G} \vdash \Theta \Uparrow x \colon A \vee^- B, \Gamma} \vee_K^-$$

$$\frac{\mathcal{G} \cup \{xRy\} \vdash \Theta \Uparrow y \colon B, \Gamma}{\mathcal{G} \vdash \Theta \Uparrow x : \Box B, \Gamma} \Box_K \qquad \frac{\mathcal{G} \vdash \Theta \Uparrow x : [y/z]B, \Gamma}{\mathcal{G} \vdash \Theta \Uparrow x : \forall z.B, \Gamma} \forall_K$$

Synchronous introduction rules

$$\frac{\mathcal{G} \vdash \Theta \Downarrow x : A \quad \mathcal{G} \vdash \Theta \Downarrow x : B}{\mathcal{G} \vdash \Theta \Downarrow x : A \wedge^+ B} \wedge_K^+ \qquad \frac{\mathcal{G} \vdash \Theta \Downarrow x : [t/z]B}{\mathcal{G} \vdash \Theta \Downarrow x : \exists z.B} \exists_K$$

$$\frac{\mathcal{G} \vdash \Theta \Downarrow x : B_i}{\mathcal{G} \vdash \Theta \Downarrow x : B_1 \vee^+ B_2} \vee^+, i \in \{1,2\} \qquad \frac{\mathcal{G} \cup \{xRy\} \vdash \Theta \Downarrow y : B}{\mathcal{G} \cup \{xRy\} \vdash \Theta \Downarrow x : \Diamond B} \Diamond_K$$

Identity rule

$$\frac{}{\mathcal{G} \vdash x : \neg B, \Theta \Downarrow x : B} \; init_K$$

Structural rules

$$\frac{\mathcal{G} \vdash \Theta, x : B \Uparrow \Gamma}{\mathcal{G} \vdash \Theta \Uparrow x : B, \Gamma} \; store_K \qquad \frac{\mathcal{G} \vdash \Theta \Uparrow x : B}{\mathcal{G} \vdash \Theta \Downarrow x : B} \; release_K \qquad \frac{\mathcal{G} \vdash x : B, \Theta \Downarrow x : B}{\mathcal{G} \vdash x : B, \Theta \Uparrow \cdot} \; decide_K$$

In $decide_K$, B is positive; in $release_K$, B is negative; in $store_K$, B is a positive formula or a negative literal; in $init_K$, B is a positive literal. In $\Box_K$ and $\forall_K$, y is different from x and z and does not occur in Θ, Γ, $\mathcal{G}$.

Figure 1: LMF^1: a focused labeled proof system for the first-order modal logic K.

harder to find, completeness is not impaired.

In order to prove the soundness and completeness of LMF^1, we need to define a translation of first-order modal formulas into first-order logic and prove that the translated formula is provable in *LKF* iff the original formula is provable in LMF^1. The translation $ST_x()$ is similar to the one in [9] and extends the standard translation (see, e.g., [7]) with a treatment for quantifiers. Treatment of polarities is omitted from the definition below since it does not affect provability. This translation provides a bridge between first-order modal logic and first-order classical logic:

$$\begin{array}{rclrcl}
ST_x(P(y_1,\ldots,y_n)) & = & P(x,y_1,\ldots,y_n) & ST_x(A \wedge B) & = & ST_x(A) \wedge ST_x(B) \\
ST_x(\neg A) & = & \neg ST_x(A) & ST_x(\Box A) & = & \forall y(R(x,y) \supset ST_y(A)) \\
ST_x(A \vee B) & = & ST_x(A) \vee ST_x(B) & ST_x(\Diamond A) & = & \exists y(R(x,y) \wedge ST_y(A)) \\
ST_x(\forall y P(y)) & = & \forall y ST_x(P(y)) & ST_x(\exists y P(y)) & = & \exists y ST_x(P(y))
\end{array}$$

where x is a free variable denoting the world in which the formula is being evaluated. The first-order language into which modal formulas are translated is usually referred to as *first-order correspondence language* [7]. It consists of a binary predicate symbol R and an $(n+1)$ary predicate symbol P for each $n$ary predicate P in the modal language. When a modal operator is translated, a new fresh variable is introduced. It is easy to show that for any modal formula A, any model $\mathcal{M}$ and any world w, we have that $\mathcal{M}, w \models A$ if and only if $\mathcal{M} \models ST_x(A)[x \leftarrow w]$.

Using the translation, we can state the soundness and completeness proposition. It is intuitively correct given the results for similar calculi but a proof will clearly be added in the future.

$$\frac{\Xi', \mathcal{G} \cup \{xRy\} \vdash \Theta \Downarrow y : B \quad \Diamond(\Xi, \Diamond B, y, \Xi')}{\Xi, \mathcal{G} \cup \{xRy\} \vdash \Theta \Downarrow x : \Diamond B} \Diamond_K$$

Figure 2: Augmenting the $\Diamond_K$ inference rule

Proposition 2.1. *Given a first-order modal formula F, $\vdash \Downarrow x : F$ is provable in LMF^1 for an arbitrary world variable x iff $\vdash \Downarrow ST_x(F)$ is provable in LKF.*

2.1 Driving the search in the focused sequent calculus

The system LMF^1 offers a structure for proof search – we can eagerly follow paths which apply asynchronous inference rules. Full proof search needs also to deal with the synchronous inference rules, for which there is no effective automation. The ProofCert project [29], which offers solutions to proof certification, suggests augmenting the inference rules with additional predicates. These predicates, on the one hand, will serve as points of communication with the implementation of the calculus (the kernel from now on) and will allow for the control and tracking of the search. On the other hand, being added as premises to the inference rules, these predicates do not affect the soundness of the kernel and therefore, do not impair the trust we can place in searching over it. They can, nevertheless, harm completeness. Consider for example an implementation of the $\exists$ control predicate which always returns the same witness. Clearly, the program will fail to find a proof if it requires any other witness. In this example, a correct implementation will prompt the user for the witness or postpone supplying it (more about that in Sec. 4.4). The control predicates communicate with the user or prover using a data structure which is being transferred and manipulated by the predicates. This data structure represents the proof evidence in the proof certifier architecture discussed by Miller in [29].

This approach is very suitable for conducting search using interactive or automated theorem provers as well. We can generalize the role of the data structure discussed above to represent information between the user and the kernel. I will therefore generalize the "proof evidence" data structure in the proof certification architecture of Miller to a *"proof control"* data structure. In this paper, this data structure can serve as a proof evidence but it will also serve for getting commands from the user as well as for generating a proof certificate once a proof was found. I can now follow other works on proof certification [12, 13] and enrich each rule of LMF^1 with proof controls and additional predicates. Figure 2 gives an example of adding the control and additional predicate (in blue) to the $\Diamond_K$ inference rule. Figure 3 lists all the predicates separately from the calculus (due to lack of space). Each sequent now contains additional information in the form of the proof control Ξ. At the same time, each rule is associated with a predicate, such as $\Diamond(\Xi, F, w, \Xi')$. This predicate might prevent the rule from being called or guide it by supplying such information as the witness to be used in the application of the $\exists_K$ or $\Diamond_K$ inference rules. The arguments in the example are the input proof control Ξ, the formula F, the world w to which we should move next and a proof control Ξ' which is given to the upper sequent. I call the resulting calculus LMF^a.

One implementation choice is to use indices in order to refer to formulas in the context. In order to achieve that, the implementations of $store_K$ and $decide_K$ rules contain additional information which is omitted from the definition of the LMF^1 calculus given in Fig. 1.

Asynchronous control predicates

$$\wedge^{-}(\Xi, F, \Xi', \Xi'') \qquad \vee^{-}(\Xi, F, \Xi') \qquad \forall(\Xi, F, \Xi' y) \qquad \Box(\Xi, F, \Xi' w)$$

Synchronous control predicates

$$\wedge^{+}(\Xi, F, \Xi', \Xi'') \qquad \vee^{+}(\Xi, F, \Xi', i) \qquad \exists(\Xi, F, t, \Xi') \qquad \Diamond(\Xi, F, w, \Xi')$$

Identity and structural control predicates

$$\mathit{init}(\Xi, l) \qquad \mathit{release}(\Xi, \Xi') \qquad \mathit{store}(\Xi, C, l, \Xi') \qquad \mathit{decide}(\Xi, l, \Xi')$$

Figure 3: The additional predicates added to the inference rules of LMF^1 in order to obtain LMF^a

3 Higher-order logic programming

The other technology I use in this paper in an attempt to build a simple but trusted proof assistant, is a higher-order logic programming language. λProlog [31] is an extension of Prolog which supports binders [30] and restricted higher-order formulas [32]. Being a logic programming language, it gives us proof search, unification, substitution and other operations which are required in any automated or interactive theorem prover. The extensions allow for the encoding of the meta-theory of predicate calculi, which is impossible in the first-order Prolog language. More concretely, the syntax of λProlog has support for λ-abstractions, written `x\ t` for $\lambda x.t$ and for applications, written `(t x)`. Existential variables can occur anywhere inside a term and are denoted by words starting with a capital letter. The variable `w` occurring in a term `F` can be universally quantified by writing `pi w\ F`. I use the symbols `some, all, box, dia, !-!` and `&-&` to denote the encoded logical connectives "exists", "for all", the modalities "box" and "diamond", a negative disjunction and a negative conjunction. The implementation contains only the negative versions of the disjunction and conjunction rules presented in Sec. 2. As discussed there, this choice does not affect provability. β-normalization and α-equality are built-in. The full syntax of the language can be found in Miller and Nadathur's book "Programming with Higher-Order Logic" [31].

The implementation of λProlog on which I have tested the prover is ELPI [16] which can be installed following instructions on Github[1]. ELPI offers more than just the implementation of λProlog and includes features such as having input/output modes on predicates and support of constraints [15]. These features are not required in the simple proof assistant I describe and are therefore not used in the implementation. Examples of the way these features are used can be found in the implementations of proof assistants for HOL [15] and type theory [22].

4 A proof assistant based on focusing and logic programming

In this section, I will present the architecture and techniques used in order to obtain a minimal, trusted proof assistant for first-order modal logic. I believe that this approach can be applied for creating proof assistants for various other logics, based on the existence of suitable focused

[1] `https://github.com/LPCIC/elpi`

calculi. Some parts of the code are omitted from this paper for brevity. These parts mainly deal with bootstrapping the program and are written using shell scripts. The proof assistant implementation can be found on Github[2] and Zenodo[3].

4.1 The kernel

The first immediate advantage of using a higher-order logic programming language is the simple and direct coding of the calculus. Fig. 4, 5 and 6 show the code of the whole implementation. A comparison to Fig. 1 shows that each inference rule directly maps to a λProlog clause. The conclusion of each rule is denoted by the head of the formula while each premise is denoted by a single conjunct in the body. The components of each head are the `Cert` variable, which is used for the transformation of information between the user and the kernel as well the formula (or formulas, in the case of a negative phase) to prove. The two phases are denoted by the function symbols `unfk` and `foc`.

We can see immediately the way the control predicates work. Before we can apply a rule, we need first to consult with the control predicate, which in turn, may change the `Cert` data structure or even falsify the call. I will refer to the implementation of these predicates in the next section, but we can already demonstrate how they work. Consider, for example, the `diamond` rule (Fig. 2 and Fig. 6). When λProlog tries to satisfy this clause, it attempts to satisfy each of the antecedents. The first of which is a call to the implementation of the `dia_ke` control predicate. The implementation is discussed in the next section but one can see that in case the implementation of this clause fails, λProlog will fail to apply the `diamond` rule and it will backtrack. One can also see that the implementation can substitute for the variable `T` a term. This term will then be used by the rule as the new label for the formula. This simple mechanism allows us to both control the proof search and to supply additional information (based on user input, for example).

The way we store polarized formulas in the implementation of the labeled sequent calculus is by using a term of the form `lform w f` where `w` is the label (world) and `f` is the formula. Atoms are polarized using the constructor `p` for positive atoms and `n` for negative ones. The example in Sec. 4.3 demonstrates the use of these constructors. Five predicates of special interest are the `store`, `forall`, `exists`, `box` and `diamond`. Each emphasizes the need for a higher-order logic programming language in a different way.

The `store` shows the importance of supporting implications in the bodies of predicates. It allows us to dynamically update the λProlog database with new true predicates. We use this feature in order to both denote the context of the sequent, i.e those formulas on which we may decide on later, and the relational set. One can also deal with this problem in the Prolog programming language. Either by using lists for denoting the context or by using the `assert` and `retract` predicates. Both approaches prevent us from having a direct and concise representation of LMF^1. The first due to the requirement to repeatedly manipulate and check the list (not to mention the overhead for searching in the list). The second due to the need to apply the system predicates manually in the correct points in the program. For example, one should manually retract an asserted predicate once we leave the scope of the implication. These manual manipulations can lead to unnecessary complications.

The `forall` predicate has a condition that the variable `y` is a fresh variable. Dealing with fresh variables is a recurring problem in all implementations of theorem provers. Some approaches favor using a specific naming scheme in order to ensure that variables are fresh while others

[2] https://github.com/proofcert/PPAssistant

[3] https://zenodo.org/record/1252457

```
1  % decide
2  check Cert (unfK []) :-
3    decide Cert Indx Cert',
4    inCtxt Indx P,
5    isPos P,
6    check Cert' (foc P).
7  % release
8  check Cert (foc N) :-
9    isNeg N,
10   release Cert Cert',
11   check Cert' (unfK [N]).
12 % store
13 check Cert (unfK [C|Rest]) :-
14   (isPos C ; isNegAtm C),
15   store Cert C Indx Cert',
16   inCtxt Indx C => check Cert' (unfK Rest).
17 % initial
18 check Cert (foc (lform L (p A))) :-
19   initial_ke Cert Indx,
20   inCtxt Indx (lform L (n A)).
```

Figure 4: λProlog implementation of the structural rules

might use an auxiliary set of used variables. Using λProlog we need just to quantify over this variable. λProlog variable capture avoidance mechanism will ensure that this variable is fresh. Another feature of λProlog which is exhibited by this rule is higher-order application. The quantified formula variable `B` is applied to the fresh variable. In general, the application of a variable to a term requires higher-order unification in the proof search, which is known to be undecidable [19]. Miller has shown [28] that the application of a variable to a bound variable require a simpler form of unification, which is not only decidable but exhibits the same properties as the first-order unification used in Prolog.

A more intriguing predicate though, is `exists`. Here we see an application of two free variables, `B` and `T`. Such an application is beyond the scope of the efficient unification algorithm just mentioned. Despite that, implementations of λProlog apply sound techniques of postponing these unification problems [34] which seem to suffice in most cases.

Regarding the modalities, we see a close similarity between `box` and `forall`. The only difference being the addition of the new accessible world to the λProlog database, in a similar way to `store`. The `diamond` rule, which is very similar to the `exists` one, then also requires the existence of the specific relation in the λProlog database in order to proceed.

4.2 Interacting with the user

The previous section discussed the implementation of the calculus. For some problems, all we need to do is to apply the kernel on a given formula. λProlog will succeed only if a proof can be found and will automatically handle all issues related to search, substitution, unification, normalization, etc. which are normally implemented as part of each theorem prover or proof assistant. This gives us a very simple implementation of an automated theorem prover for first-order modal logic. The downside is, of course, that first-order modal theorem proving is undecidable and requires coming up with witnesses for worlds and terms, making automated theorem proving over the sequent calculus less practical than other methods, such as resolution [14] and free-variable tableaux [10]. The main novelty of this paper is that we can overcome this downside by using other features of λProlog, namely the input and output functionality.

Using the control predicates, we can notify the user of interesting rule applications, such as the addition of fresh variables, new worlds or the storing of formulas in the context. We can also

```
1  % orNeg
2  check Cert (unfK [lform L (A !-! B) | Rest]) :-
3    orNeg_kc Cert (lform L (A !-! B))  Cert',
4    check Cert' (unfK [lform L A, lform L B| Rest]).
5  % conjunction
6  check Cert (unfK [lform L (A &-& B) | Rest]) :-
7    andNeg_kc Cert (lform L (A &-& B)) CertA CertB,
8    check CertA (unfK [lform L A | Rest]),
9    check CertB (unfK [lform L B | Rest]).
10 % box modality
11 check Cert (unfK [lform L (box B) | Theta]) :-
12   box_kc Cert (lform L (box B)) Cert',
13   pi w\ rel L w => check (Cert' w) (unfK [lform w B | Theta] ).
14 % forall quantifier
15 check Cert (unfK [lform L (all B) | Theta]) :-
16   all_kc Cert (all B) Cert',
17   pi w\ (check (Cert' w) (unfK [lform L (B w) | Theta] )).
```

Figure 5: λProlog implementation of the asynchronous rules

```
1  % diamond modality
2  check Cert (foc (lform L (dia B))) :-
3    dia_ke Cert (lform L (dia B)) T Cert',
4    rel L T,
5    check Cert' (foc (lform T B)).
6  % exists quantifier
7  check Cert (foc (lform L (some B))) :-
8    some_ke Cert (lform L (some B)) T Cert',
9    check Cert' (foc (lform L (B T))).
```

Figure 6: λProlog implementation of the Synchronous rules

use them in order to prompt the user for input about how to proceed in case we need to decide on a formula from the context or pick up a witness or a world. Fig. 7 shows the implementation of the control predicates which support these basic operations. The predicates are divided into two groups. Those which can be applied fully automatically, which include most predicates, and those which are applied interactively, which include the `decide`, `diamond` and `exists` predicates. I have simplified the implementation to include only negative conjunctions and disjunctions (see Sec. 3). The addition of the positive versions does not fundamentally change the approach presented here. In case we would like to support these two inference rules, we will have to treat them in the interactive group.

The interface for a user interaction with the program is to iteratively add guidance information to the proof control. At the beginning, the control contains no user information and the program stops the moment such information is required. In addition, the program displays to the user information about the current proof state such as about fresh variables which were used or new formulas which were added to the context, together with their indices. When the program stops due to required user information, it prompts a message to the user asking the user to supply this information as can be seen in the implementation of the predicates `decide` (lines 1-3), `diamond` (lines 25-32) and `some` (lines 42-49).

The proof control I use contains 5 elements.

- the proof evidence – this is used in order to display at the end to the user the generated proof
- the list of user commands – this list, initially empty, contains the commands from the user

- an index marking the current inference rule – this index is used to store labeled formulas in a consistent way
- the list of fresh worlds generated so far – this list is used in order to allow the user to pick up a world The user, of course, has no access to the fresh worlds (or to any other part in the trusted kernel) and I use a mechanism discussed below in order to allow her to supply them
- the list of fresh variables generated so far – Similarly to the list of fresh worlds, this list is used in order to allow the user to supply term witnesses which contain fresh variables

Each of the interactive predicates contains two versions, one for prompting the user for input and the other for applying the user input. The first is applied when the user commands list (the second argument in the controls object) is empty. The input in the case of the `decide` predicate is an index of a formula in context (which should be chosen from the ones displayed earlier by the `store` predicate). In the case of the `diamond` or `some` predicates, the input is the term witness.

In the case of the `diamond` and on some cases, also for the `some` inference rule, the implementation needs to substitute fresh variables inside the term supplied by the user. I use λProlog abstraction and β-normalization directly. The user keeps track on the number n and order of both fresh worlds and fresh variables introduced so far and the chosen world or the term witness is then of the form $x_1\backslash \dots x_n\backslash t$ where t may contain any of the bound variables, in the case of `some`, or the actual chosen world, in the case of `diamond`. The `apply_vars` predicate is responsible for applying to the terms the fresh variables in the correct order. It should be noted that this cumbersome mechanism can be easily replaced by a naming mechanism given a more sophisticated user interface.

The indexing mechanism I use is based on trees and assign each unitary child of a parent `I` the index `(u I)` while binary children are assigned the indices `(l I)` and `(r I)` respectively. The index of the theorem is `e`. Note that the indexing system is based on inferences and that indices are assigned to formulas only upon storing them in the context. For example, if our theorem is `A &-& B`, meaning a negative conjunction, then this formula is assigned the index `e`. In the focused sequent calculus, the only inference rule which can be applied right now is the negative conjunction and the left and right derivations keep track of the indices `(l e)` and `(r e)`. Only in case we store the formulas `A` or `B` in the following step will we assign them the indices `(l e)` or `(r e)`

I note here that the implementation is using a relatively basic user feedback. In particular, when conjunctions and branching are involved, it becomes very difficult to follow the different branches and their respective contexts and fresh variables. I do supply a mechanism for handling conjunctions (via the `branch` command), but a more user friendly implementation would need to display this information in a better way, for example, by the use of graphical trees.

4.3 An example

In this section I demonstrate the execution of the prover on the Barcan formula, which is a theorem of modal logic K with constant domains and rigid terms. In order to use the assistant, the user needs to call the prover with the theorem and an empty commands list.

```
$>./run.sh '((some x\ dia (n (q x))) !-! (box (all x\ (p (q x)))))' '[]'
```

Since we start in a negative phase and the theorem is negative, the assistant eagerly does the following ordered steps, the first five of which are asynchronous.

```
1  decide (interact (unary (decideI no_index) leaf) [] _ _ _) _ _ :- !,
2    output std_out "You␣have␣to␣choose␣an␣index␣to␣decide␣on␣from␣the␣context",
3    output std_out "\n", fail.
4  decide_ke (interact (unary (decideI I) L) [I|Com] FI E1 E2) I
5      (interact L Com (u FI) E1 E2).
6  store_kc (interact (unary (storeI I) L) Com I E1 E2) F I (interact L Com (u I) E1 E2) :-
7    output std_out "Adding␣to␣context␣formula␣",
8    term_to_string F S1,
9    output std_out S1,
10   output std_out "␣with␣index␣",
11   term_to_string I S2,
12   output std_out S2,
13   output std_out "\n".
14 release_ke (interact (unary releaseI L) Com FI E1 E2) (interact L Com (u FI) E1 E2).
15 initial_ke (interact (axiom (initialI I)) []  _ _ _) I.
16 orNeg_kc (interact (unary (orNegI FI) L) Com FI E1 E2) F (interact L Com (u FI) E1 E2).
17 andNeg_kc (interact (binary (andNegI FI) L1 L2) [branch LC RC] FI E1 E2) F
18     (interact L1 LC (l FI) E1 E2) (interact L2 RC (r FI) E1 E2).
19 box_kc (interact (unary (boxI FI) L) Com FI E1 E2) F (Eigen\ (interact L) Com (u FI)
20     [eigen FI Eigen| E1] E2) :-
21   output std_out "Using␣world␣variable␣",
22   term_to_string FI S1,
23   output std_out S1,
24   output std_out "\n".
25 dia_ke (interact (unary (diaI no_index) leaf) [] FI _ _) F _ _ :- !,
26   output std_out "You␣have␣to␣choose␣the␣world␣to␣use␣for␣instatiation␣for␣the␣formula:␣",
27   term_to_string F S1,
28   output std_out S1,
29   output std_out "\nAt␣index:␣",
30   term_to_string FI S2,
31   output std_out S2,
32   output std_out "\n", fail.
33 dia_ke (interact (unary (someI FI) L) [W|Com] FI E1 E2) _ W'
34     (interact L Com (u FI) E1 E2) :-
35   apply_vars W E1 W'.
36 all_kc (interact (unary (allI FI) L) Com FI E1 E2) F
37     (Eigen\ (interact L) Com (u FI) E1 [eigen FI Eigen| E2]) :-
38   output std_out "Using␣eigen␣variable␣",
39   term_to_string FI S1,
40   output std_out S1,
41   output std_out "\n".
42 some_ke (interact (unary (someI no_index) leaf) [] FI _ _) F _ _ :- !,
43   output std_out "You␣have␣to␣choose␣the␣term␣to␣use␣for␣instantiation:␣",
44   term_to_string F S1,
45   output std_out S1,
46   output std_out "\nAt␣index:␣",
47   term_to_string FI S2,
48   output std_out S2,
49   output std_out "\n", fail.
50 some_ke (interact (unary (someI FI) L) [T|Com] FI E1 E2) _ T'
51     (interact L Com (u FI) E1 E2) :-
52   apply_vars T E2 T'.
53 apply_vars T [] T.
54 apply_vars T [eigen _ X|L] T' :-
55   apply_vars (T X) L T'.
```

Figure 7: λProlog implementation of basic interaction with the user

```
Adding to context formula lform z (some x0 \ dia (n (q x0))) with index u e
Using world variable u (u e)
Using eigen variable u (u (u e))
Adding to context formula lform x0 (p (q x1)) with index u (u (u (u e)))
Adding to context formula lform x0 (n (q x1)) with index u (u (u (u (u (u (u (u e)))))))
You have to choose an index of a formula to decide on from the context
At index: u (u (u (u (u (u (u (u (u e))))))))
```

Figure 8: A screenshot of an intermediary step of proving the Barcan formula

1. Applies the $\vee^-_K$ inference rule
2. Adds to the context the positive formula `lform z some x\ dia (n (q x))` with index `u e` (`z` denotes the starting world)
3. Applies the $\Box_K$ in order to produce a fresh world
4. Applies the $\forall_K$ in order to produce a fresh variable
5. Adds to the context the positive formula `lform x0 (p (q x1))` with index `u (u (u (u e)))`
6. Prompts the user to input an index to decide on

Using the provided simple user interface, the program now terminates and the user is expected to run it again, this time setting the list provided in the second argument to contain an interactive command. The first interactive command is therefore, to choose the index `u e`. We are now entering the synchronous phase and are asked also to supply the witness for the $\exists_K$ rule. In this case, the witness is just the (only) fresh variable introduced earlier and we command the assistant to choose `(x\x)`. Still being in a synchronous phase, we are now asked to supply the world to satisfy the $\Diamond_K$ rule. We choose the first (and only) previously introduced world using `(x\x)`. The assistant now observes that we have the negative atom `lform x0 (n (q x1))` with index `u (u (u (u (u (u (u (u e))))))))`. We are now asked again to pick up an index of a formula to decide on. We observe that the context contains the positive and negative versions of the same atom (in the same world) and we decide on the positive version with index `u (u (u (u e)))`. The $init_K$ rule is automatically applied and the assistant responds with the formal proof we have obtained.

The last execution is therefore,

```
$>./run.sh '((some x\ dia (n (q x))) !-! (box (all x\ (p (q x)))))'
  '[(u e),(x\x),(x\x),u (u (u (u e)))]'
```

Fig. 8 shows a screenshot of the interaction after supplying the input `'[(u e),(x\x),(x\x)]'`

4.4 Creating and using tactics

Supporting interactive proof search still falls short from the needs of most users. Optimally, a proof assistant would require the help of the user only for the most complex problems and will be able to deal with simpler ones by itself. In the previous example, we had to search for the index to decide on. But, there are finitely many options only. Can't we let the prover try all options by itself? In order to support a tactics language, I extend the program with an additional tactics file. This file will contain additional implementations for the control predicates. The λProlog interpreter will choose the right implementation according to whether the predicate is called with a tactic command or with a command to decide on an index of a formula or a witness

```
1  decide_ke (interact (unary (decideI I) L) [auto|Com] FI E1 E2) I
2      (interact L Com (u FI) E1 E2).
3  dia_ke (interact (unary (diaI FI) L) [world|Com] FI E1 E2) _ T
4      (interact L Com (u FI) E1 E2) :-
5    apply_vars T E1 T'.
6  some_ke (interact (unary (someI FI) L) [var|Com] FI E1 E2) _ _
7      (interact L Com (u FI) E1 E2) :- !.
```

Figure 9: λProlog implementation of some tactics

term. Fig. 9 presents the additional predicates we need to add in order to support several basic tactics.

The `auto` tactic, which is supplied when asked to decide on a formula, attempts to choose one according to the order they are stored in the λProlog database. Similarly, the `world` tactic attempts to choose a world according to the order we have stored them in the proof control. Coming up with a witness is more complex. Unlike with deciding on a formula or selecting a world, we are now facing a possibly infinite number of options. Luckily, λProlog can again help us with the task. We can use the language metavariables and λProlog will postpone the choice until it can unify this variable with an appropriate term. The `var` tactic therefore replaces the chosen term with such a metavariable.

Using these tactics, the commands required in order to prove the theorem from the example is

```
$>./run.sh '((some x\ dia (n (q x))) !-! (box (all x\ (p (q x)))))'
  '[auto,var,world,auto]'
```

It should be noted though, that in the general case the simple tactics presented may not be as easy to use. For example, when deciding using the `auto` command, the system will decide on the last formula stored in the context and only if it fails later to find a proof, will it backtrack and pick another. This scheme will therefore get very confusing if we are also prompt to input information on the wrong branch. One can think of more advanced tactics which present to the user all possible paths and not just one as in the current implementation.

5 Conclusion and further work

The aim of this paper was to investigate the applicability of a minimal proof assistant based on focusing and λProlog for interactive proof search in first-order modal logic. We have considered also the amount of work required in order to design proof assistants for arbitrary focused systems.

The main target audience of this approach are users who are in need of a proof assistant for logics which do not enjoy an abundant number of tools. The next step is to try to apply this approach to concrete domains where interactive tools are scarce, such as in deontic logic ([5] contains some interesting recent developments). There are several other possible extensions to this work. An important extension is the creation of a generic graphical user interface, which can parse and display λProlog proofs and proof information. Another is the creation of a library of basic tactics which can be applied to a variety of logics.

References

[1] The Coq proof assistant. https://coq.inria.fr/.

[2] Jean-Marc Andreoli. Logic programming with focusing proofs in linear logic. *J. Log. Comput.*, 2(3):297–347, 1992.

[3] Gilles Barthe, Benjamin Grégoire, and Santiago Zanella Béguelin. Formal certification of code-based cryptographic proofs. *ACM SIGPLAN Notices*, 44(1):90–101, 2009.

[4] Christoph Benzmüller and Bruno Woltzenlogel Paleo. Interacting with modal logics in the coq proof assistant. In *International Computer Science Symposium in Russia*, pages 398–411. Springer, 2015.

[5] Christoph Benzmüller, Xavier Parent, and Leendert van der Torre. A deontic logic reasoning infrastructure. In Russel Miller, Dirk Nowotka, and Florin Manea, editors, *14th Conference on Computability in Europe, CiE 2018, Kiel, Germany, July 30-August, 2018, Proceedings*. Springer, 2018. To appear.

[6] Christoph Benzmüller and Bruno Woltzenlogel Paleo. Higher-order modal logics: Automation and applications. In Adrian Paschke and Wolfgang Faber, editors, *Reasoning Web 2015*, number 9203 in LNCS, pages 32–74, Berlin, Germany, 2015. Springer. (Invited paper).

[7] Patrick Blackburn and Johan Van Benthem. Modal logic: a semantic perspective. In *Studies in Logic and Practical Reasoning*, volume 3, pages 1–84. Elsevier, 2007.

[8] Sylvie Boldo, François Clément, Jean-Christophe Filliâtre, Micaela Mayero, Guillaume Melquiond, and Pierre Weis. Formal proof of a wave equation resolution scheme: the method error. In *International Conference on Interactive Theorem Proving*, pages 147–162. Springer, 2010.

[9] Torben Braüner and Silvio Ghilardi. 9 first-order modal logic. In *Studies in Logic and Practical Reasoning*, volume 3, pages 549–620. Elsevier, 2007.

[10] Serenella Cerrito and Marta Cialdea Mayer. Free-variable tableaux for constant-domain quantified modal logics with rigid and non-rigid designation. In *International Joint Conference on Automated Reasoning*, pages 137–151. Springer, 2001.

[11] Boutheina Chetali and Quang-Huy Nguyen. About the world-first smart card certificate with eal7 formal assurances. *Slides 9th ICCC, Jeju, Korea (September 2008), www. commoncriteriaportal. org/iccc/9iccc/pdf B*, 2404, 2008.

[12] Zakaria Chihani, Tomer Libal, and Giselle Reis. The proof certifier checkers. In *International Conference on Automated Reasoning with Analytic Tableaux and Related Methods*, pages 201–210. Springer, 2015.

[13] Zakaria Chihani, Dale Miller, and Fabien Renaud. A semantic framework for proof evidence. *Journal of Automated Reasoning*, 59(3):287–330, 2017.

[14] Marta Cialdea. Resolution for some first-order modal systems. *Theoretical Computer Science*, 85(2):213–229, 1991.

[15] Cvetan Dunchev, Claudio Sacerdoti Coen, and Enrico Tassi. Implementing hol in an higher order logic programming language. In *Proceedings of the Eleventh Workshop on Logical Frameworks and Meta-Languages: Theory and Practice*, page 4. ACM, 2016.

[16] Cvetan Dunchev, Ferruccio Guidi, Claudio Sacerdoti Coen, and Enrico Tassi. Elpi: Fast, embeddable, \lambda prolog interpreter. In *Logic for Programming, Artificial Intelligence, and Reasoning*, pages 460–468. Springer, 2015.

[17] Amy Felty. Implementing tactics and tacticals in a higher-order logic programming language. *Journal of Automated Reasoning*, 11(1):43–81, 1993.

[18] Amy Felty and Dale Miller. Specifying theorem provers in a higher-order logic programming language. In *International Conference on Automated Deduction*, pages 61–80. Springer, 1988.

[19] Warren D Goldfarb. The undecidability of the second-order unification problem. *Theoretical Computer Science*, 13(2):225–230, 1981.

[20] Georges Gonthier. Formal proof–the four-color theorem. *Notices of the AMS*, 55(11):1382–1393, 2008.

[21] Georges Gonthier, Andrea Asperti, Jeremy Avigad, Yves Bertot, Cyril Cohen, François Garillot,

Stéphane Le Roux, Assia Mahboubi, Russell OConnor, Sidi Ould Biha, et al. A machine-checked proof of the odd order theorem. In *International Conference on Interactive Theorem Proving*, pages 163–179. Springer, 2013.

[22] Ferruccio Guidi, Claudio Sacerdoti Coen, and Enrico Tassi. Implementing type theory in higher order constraint logic programming. 2017.

[23] Thomas Hales, Mark Adams, Gertrud Bauer, Tat Dat Dang, John Harrison, Hoang Le Truong, Cezary Kaliszyk, Victor Magron, Sean McLaughlin, Tat Thang Nguyen, et al. A formal proof of the kepler conjecture. In *Forum of Mathematics, Pi*, volume 5. Cambridge University Press, 2017.

[24] John Harrison. The HOL light theorem prover. `https://github.com/jrh13/hol-light/`.

[25] Xavier Leroy. Formal verification of a realistic compiler. *Communications of the ACM*, 52(7):107–115, 2009.

[26] Chuck Liang and Dale Miller. Focusing and polarization in linear, intuitionistic, and classical logics. *Theor. Comput. Sci.*, 410(46):4747–4768, 2009.

[27] Sonia Marin, Dale Miller, and Marco Volpe. A focused framework for emulating modal proof systems. In *Advances in Modal Logic 11, proceedings of the 11th conference on "Advances in Modal Logic," held in Budapest, Hungary, August 30 - September 2, 2016*, pages 469–488, 2016.

[28] Dale Miller. A logic programming language with lambda-abstraction, function variables, and simple unification. *Journal of logic and computation*, 1(4):497–536, 1991.

[29] Dale Miller. A proposal for broad spectrum proof certificates. In *International Conference on Certified Programs and Proofs*, pages 54–69. Springer, 2011.

[30] Dale Miller. Mechanized Metatheory Revisited: An Extended Abstract . In *Post-proceedings of TYPES 2016* , Novi Sad, Serbia, 2017.

[31] Dale Miller and Gopalan Nadathur. *Programming with higher-order logic*. Cambridge University Press, 2012.

[32] Dale Miller, Gopalan Nadathur, Frank Pfenning, and Andre Scedrov. Uniform proofs as a foundation for logic programming. *Annals of Pure and Applied logic*, 51(1-2):125–157, 1991.

[33] Dale Miller and Marco Volpe. Focused labeled proof systems for modal logic. In *Logic for Programming, Artificial Intelligence, and Reasoning*, pages 266–280. Springer, 2015.

[34] Gopalan Nadathur. A treatment of higher-order features in logic programming. *Theory and Practice of Logic Programming*, 5(3):305–354, 2005.

[35] Tobias Nipkow, Lawrence C Paulson, and Markus Wenzel. *Isabelle/HOL: a proof assistant for higher-order logic*, volume 2283. Springer Science & Business Media, 2002.

[36] Jens Otten. Mleancop: A connection prover for first-order modal logic. In *International Joint Conference on Automated Reasoning*, pages 269–276. Springer, 2014.

[37] Livio Robaldo and Xin Sun. Reified input/output logic: Combining input/output logic and reification to represent norms coming from existing legislation. *Journal of Logic and Computation*, 27(8):2471–2503, 2017.

[38] Enrico Tassi. Elpi: an extension language for coq metaprogramming coq in the elpi λprolog dialect. 2017.

Labelled Connection-based Proof Search for Multiplicative Intuitionistic Linear Logic

Didier Galmiche and Daniel Méry

LORIA - Université de Lorraine
Campus Scientifique, BP 239
Vandœuvre-lès-Nancy, France

Abstract

We propose a connection-based characterization for multiplicative intuitionistic linear logic (MILL) which is based on labels and constraints that capture Urquhart's possible world semantics of the logic. We first briefly recall the purely syntactic sequent calculus for MILL, which we call LMILL. Then, in the spirit of our previous results on the Logic of Bunched Implications (BI), we present a connection-based characterization of MILL provability. We show its soundness and completeness without the need for any notion of multiplicity. From the characterization, we finally propose a labelled sequent calculus for MILL.

1 Introduction

In previous works we have developed connection-based characterizations of validity in non-classical logics like the Logic of Bunched Implications (BI) [2] and Bi-intuitionistic logic [3]. They are based on specific concepts like *labels* and *constraints* in order to capture the model semantics and then the semantic interactions between connectives in such logics. It is an alternative approach to the standard view of connection calculi for non-classical logics that are based on the notion of *prefixes*. This notion allows one to capture the non-permutabilities of the sequent calculi rules and has been developed and improved in the context of intuitionistic logic [9, 10] but also of modal logics [8]. There exist connection-based characterizations and related connection methods for multiplicative (commutative) linear logic (MLL) [1, 6] but, as far as we know, not for multiplicative intuitionistic linear logic (MILL). The connection-based characterization proposed for fragments of Linear Logic [4] like MLL or MELL is based on particular prefixes and substitutions dedicated to these logics [6, 5]. In order to extend or adapt it to MILL, it would be necessary to define and consider what could be called intuitionistic and linear prefixes, which could be difficult to deal with.

Our approach consists in specializing our above-mentioned results for BI to the Multiplicative Intuitionistic Linear Logic (MILL) and then define and illustrate a connection-based characterization of provability for MILL that deals with specific labels and constraints. Then we generate semantic structures from MILL's Urquhart's semantics [12] and develop a characterization of provability from labels and constraints that capture this semantics. It could be seen as a generalization of the prefixes more appropriate to connection-based proof search in resource logics like BI logic or Linear Logic. Since BI is conservative over MILL [7], a connection-based characterization for MILL can be obtained, by restriction of the previous characterization for BI, to the multiplicative connectives. In this case, we have to take into account the notion of multiplicity and also global conditions on the paths. The characterization for MILL proposed here does not deal with multiplicity and only considers local conditions on the paths. In addition we define a labelled sequent calculus for MILL, called GMILL and prove its soundness and also its completeness by translation of MILL proofs to GMILL proofs.

$$\frac{}{\mathrm{A} \vdash \mathrm{A}}\ ax \qquad \frac{\Gamma \vdash \mathrm{C}}{\Gamma, 1 \vdash \mathrm{C}}\ 1_L \qquad \frac{}{\vdash 1}\ 1_R$$

$$\frac{\Gamma, \mathrm{A}, \mathrm{B}, \Delta \vdash \mathrm{C}}{\Gamma, \mathrm{A} * \mathrm{B}, \Delta \vdash \mathrm{C}}\ *_L \qquad \frac{\Gamma \vdash \mathrm{A} \quad \Delta \vdash \mathrm{B}}{\Gamma, \Delta \vdash \mathrm{A} * \mathrm{B}}\ *_R$$

$$\frac{\Gamma \vdash \mathrm{A} \quad \Delta, \mathrm{B} \vdash \mathrm{C}}{\Gamma, \mathrm{A} \mathrel{-\!\!*} \mathrm{B}, \Delta \vdash \mathrm{C}}\ {-\!\!*}_L \qquad \frac{\Gamma, \mathrm{A} \vdash \mathrm{B}}{\Gamma \vdash \mathrm{A} \mathrel{-\!\!*} \mathrm{B}}\ {-\!\!*}_R$$

Figure 1: Sequent Calculus for MILL

2 Multiplicative Intuitionistic Linear Logic

The propositional language of MILL consists of a denumerable set $L = \mathrm{P}, \mathrm{Q}, \ldots$ of propositional letters, the multiplicative unit 1 and the multiplicative connectives $*$ and $-\!\!*$. $\mathcal{P}(L)$, the collection of MILL propositions over L, is given by the following inductive definition:

$$\mathrm{A} ::= \mathrm{P} \mid 1 \mid \mathrm{A} * \mathrm{A} \mid \mathrm{A} \mathrel{-\!\!*} \mathrm{A}.$$

Let us remark that since the forthcoming connection-based characterization of MILL-provability is inspired by our previous work on BI [2], we do not use the more widespread symbols $\otimes$ and $\multimap$ to denote multiplicative conjunction and implication and rather stick with the star and magic-wand notations of these connectives.

Judgements of MILL are sequents of the form $\Gamma \vdash \mathrm{A}$, where A is a proposition and Γ, called the context, is a (possibly empty) multiset of formulas.

The standard sequent calculus for MILL, which we call LMILL[1], is given in Figure 1. One difficulty with such a calculus lies in the fact that the rules for left-implication and right-conjunction both require context splitting from conclusion to premises. Making relevant choices when context-splitting is required is crucial for the efficiency of backward proof-search.

The semantics we use for MILL models is a possible worlds semantics à la Kripke, mainly inspired from the operational semantics of Urquhart [12]. Let us recall it briefly.

Definition 1 (MILL-frame). *A* MILL-frame *is a partially ordered commutative monoid* $\mathcal{M} = \langle M, \cdot, e, \sqsubseteq \rangle$, *in which M is a set of worlds and $\sqsubseteq$ is compatible with $\cdot$,* i.e.*:*

$$\forall m \forall n \in M.\ \textit{if } m \sqsubseteq n \textit{ and } m' \sqsubseteq n' \textit{ then } m \cdot m' \sqsubseteq n \cdot n'.$$

Definition 2 (MILL-interpretation). *A* MILL-intepretation *is a function* $[\![-]\!] : L \to \mathcal{P}(M)$ *that satisfies Kripke monotonicity,* i.e.*:*

$$\forall m, n \in M.\ \textit{if } m \sqsubseteq n \textit{ and } m \in [\![\mathrm{P}]\!] \textit{ then } n \in [\![\mathrm{P}]\!].$$

Definition 3 (MILL-model). *Let $\mathcal{P}(L)$ be the collection of MILL propositions over a language L of propositional letters, a* MILL-model *is a structure* $\mathcal{R} = \langle M, \cdot, e, \sqsubseteq, [\![-]\!], \models \rangle$, *in which* $\langle M, \cdot, e, \sqsubseteq \rangle$ *is a MILL-frame,* $[\![-]\!]$ *is a MILL-interpretation, and $\models$ is a forcing relation on* $M \times \mathcal{P}(L)$ *satisfying the following conditions:*

- $m \models \mathrm{P}$ *iff* $m \in [\![\mathrm{P}]\!]$

[1] In the spirit of LJ and LK for intuitionistic and classic logic, although LMILL has no labels.

- $m \models 1$ *iff* $e \sqsubseteq m$
- $m \models \mathrm{A} * \mathrm{B}$ *iff there exist* $n_1, n_2 \in M$ *such that* $n_1 \cdot n_2 \sqsubseteq m$, $n_1 \models \mathrm{A}$ *and* $n_2 \models \mathrm{B}$
- $m \models \mathrm{A} \multimap\!\!* \, \mathrm{B}$ *iff, for all* $n_1, n_2 \in M$, *if* $n_1 \models \mathrm{A}$ *and* $m \cdot n_1 \sqsubseteq n_2$ *then* $n_2 \models \mathrm{B}$.

Definition 4 (MILL-validity)**.** *Let* $\mathcal{R}$ *be MILL-model. A formula* A *is* valid in $\mathcal{R}$*, written* $\mathcal{R} \models \mathrm{A}$*, iff* $e \models \mathrm{A}$*.* A *is* valid*, written,* $\models \mathrm{A}$*, iff* $\mathcal{R} \models \mathrm{A}$ *for all models* $\mathcal{R}$*. A finite set of formulas* $\{\mathrm{A}_1, \ldots, \mathrm{A}_n\}$ entails *a formula* B*, written* $\mathrm{A}_1, \ldots, \mathrm{A}_n \models \mathrm{B}$*, iff* $\models (\mathrm{A}_1 * \ldots * \mathrm{A}_n) \multimap\!\!* \, \mathrm{B}$.

Theorem 1 (Soundness and Completeness)**.** *For all formulas* A*,* $\models \mathrm{A}$ *iff* $\vdash \mathrm{A}$.

3 Labelled Connections for MILL

The connection-based characterization of MILL-provability we define in this paper is based on *labels* and *constraints* that capture the semantic properties of MILL-frames instead of capturing the syntactic properties (such as permutabilities, context-splitting or linearity) of the purely syntactic sequent calculus LMILL as the standard prefix-based approach does for IL or MLL [5, 6, 13]. We already successfully used a similar approach for BI [2], and since BI is conservative w.r.t. MILL [7], the characterization in [2] also applies to MILL. However, although the characterization for MILL presented in this paper can indeed be seen as a refinement of the one given for BI, its improvements are two-fold: firstly, it does not need any notion of multiplicity and, secondly, it is local in that the conditions for characterizing provability are not stated w.r.t. the global set of atomic paths but w.r.t. each atomic path individually. The locality of the characterization is a key step towards implementing a memory-space efficient depth-first path-reduction strategy in a future connection-based prover for MILL. Indeed global conditions would require us to keep the whole set of atomic paths in memory while checking provability conditions. Since the number of atomic paths can grow exponentially large with the size of a formula, local conditions are preferable if one wants to make a more efficient use of the memory space (linear in the size of the initial formula).

3.1 Labels and Constraints

Given an alphabet C (for example $a, b, c, \cdots$), C_0, the set of *atomic labels over* C, is defined as the set C extended with the unit symbol ϵ. We then define $\mathcal{L}C$, the set of *labels over* C, as the smallest set containing C_0, and closed under composition ($x, y \in \mathcal{L}C$ implies $xy \in \mathcal{L}C$). Labels are considered up to associativity, commutativity and identity w.r.t. ϵ. Therefore, $aabcc$, $cbaca$ and $cbcaa\epsilon$ are simply regarded as equivalent.

A *label constraint* is an expression $x \leq y$ where x and y are labels. A constraint of the form $x \leq x$ is called an *axiom* and we write $x = y$ to express that $x \leq y$ and $y \leq x$. We use the following inference rules for reasoning on constraints:

$$\frac{}{x \leq x}\, R \qquad\qquad \frac{x \leq z \quad z \leq y}{x \leq y}\, T \qquad\qquad \frac{x \leq y \quad x' \leq y'}{xx' \leq yy'}\, F$$

The R and T rules formalize the reflexivity and transitivity of $\leq$ while the F rule corresponds to the functoriality (also called compatibility) of label-composition w.r.t. $\leq$. In this formal system, given a constraint k and a set of constraints H, we write $H \mathrel{|\!\!\approx} k$ if there is a deduction of k from H. The notation $H \mathrel{|\!\!\approx} K$, where K is a non-empty set of constraints, means that for all $k \in K$, $H \mathrel{|\!\!\approx} k$.

3.2 Labelled Indexed Formula Tree

Here we recall the standard notions coming from previous matrix-characterizations of provability [6, 13]. A *decomposition tree* of a formula A is its representation as a syntactic tree with nodes called *positions*. A position u exactly identifies a subformula of A denoted $f(u)$. An *atomic position* is a position for an atomic formula. If u is a non-atomic position the principal connective of $f(u)$ is denoted $c(u)$. Moreover such a position corresponds to an internal node and we denote $[u]_i$ with $i \in \{1,2\}$ the position of the i-th child of the node corresponding to u and then $[u]_\star = \{ v \mid (\exists i \in \mathbb{N})(v = [u]_i) \}$. If u is not a root position we say that u is of rank $r(u) = i$ if u is the i-th child of its father position denoted by $[u]_0$.

The decomposition tree induces a partial order $\ll$ on the positions such that the root is the least element and if $u \ll v$ then u dominates v in the tree or in the formula (from now on we do not distinguish a formula A from its decomposition tree). Then we denote $[u]\!\uparrow$ the set $\{ v \mid v \in \mathrm{A} \text{ and } v \ll u \}$ of upward positions of u and $[u]\!\downarrow$ the set $\{ v \mid v \in \mathrm{A} \text{ and } u \ll v \}$ of its downward positions. The notations $[\cdot]\!\uparrow$ and $[\cdot]\!\downarrow$ are easily generalized to a set s of positions by $[s]\!\uparrow = \{ u \mid u \in [v]\!\uparrow \text{ and } v \in s \}$ and $[s]\!\downarrow = \{ u \mid u \in [v]\!\downarrow \text{ and } v \in s \}$.

For each position, we assign a polarity $pol(u)$ but also a principal type $ptyp(u)$ and a secondary type $styp(u)$. Therefore, we have different principal types depending on the connective and the associated polarity. We define two principal types named π_α, π_β. Given a set of positions p, $P_\alpha(p) = \{ u \mid u \in p \text{ and } ptyp(u) = \pi_\alpha \}$ is the set of positions of type π_α and $P_\beta(p) = \{ u \mid u \in p \text{ and } ptyp(u) = \pi_\beta \}$ is the set of positions of type π_β.
Moreover we consider the following sets of secondary type positions: for i in $\{1,2\}$, $S_{\alpha i}(p) = \{ u \mid u \in p \text{ and } styp(u) = \pi_{\alpha i} \}$, $S_{\beta_i}(p) = \{ u \mid u \in p \text{ and } styp(u) = \pi_{\beta_i} \}$ and $S_\alpha(p) = S_{\alpha 1}(p) \cup S_{\alpha 2}(p)$, $S_\beta(p) = S_{\beta_1}(p) \cup S_{\beta_2}(p)$.

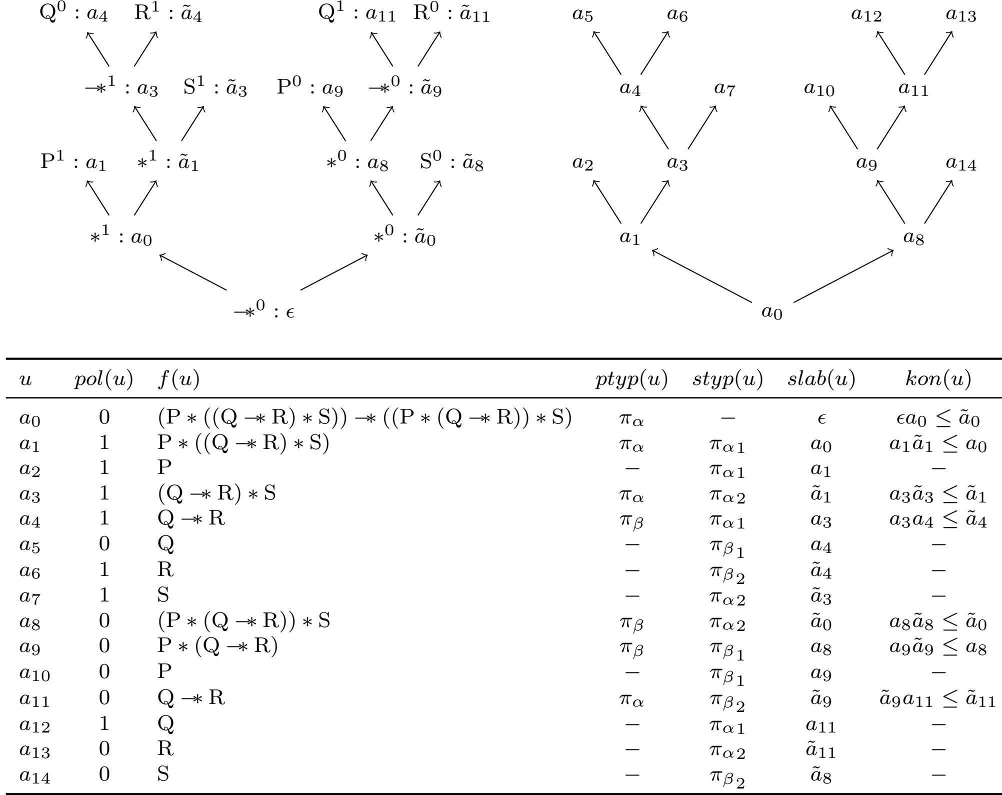
Depending on the principal type, we associate a label $slab(u)$ and sometimes a constraint $kon(u)$ to a position u. Such a label is either a position or a position with a tilde in order to identify the formula that introduced the label. We define constraints in order to capture context-splitting. The *labelled signed formula* $lsf(u)$ of a position u is a triple $(slab(u), f(u), pol(u))$ and is denoted $f(u)^{pol(u)} : slab(u)$. The construction of an indexed formula tree is obtained by inductively applying the rules described in Figure 2.

$lsf(u)$	$ptyp(u)$	$kon(u)$	$lsf(u_1)$	$lsf(u_2)$
$(\mathrm{A} \multimap \mathrm{B})^0 : x$	π_α	$xu \leq \tilde{u}$	$\mathrm{A}^1 : u$	$\mathrm{B}^0 : \tilde{u}$
$(\mathrm{A} * \mathrm{B})^1 : x$	π_α	$u\tilde{u} \leq x$	$\mathrm{A}^1 : u$	$\mathrm{B}^1 : \tilde{u}$
$(\mathrm{A} \multimap \mathrm{B})^1 : x$	π_β	$xu \leq \tilde{u}$	$\mathrm{A}^0 : u$	$\mathrm{B}^1 : \tilde{u}$
$(\mathrm{A} * \mathrm{B})^0 : x$	π_β	$u\tilde{u} \leq x$	$\mathrm{A}^0 : u$	$\mathrm{B}^0 : \tilde{u}$

Figure 2: Signed formulae for MILL

For a given formula A the root position a_0 has a polarity $pol(a_0) = 0$, a label $slab(a_0) = \epsilon$ and the signed formula $(\mathrm{A})^0 : \epsilon$ where ϵ is the identity of label composition. u_1 and u_2 are respectively the first and second subpositions. The principal type of a position u depends on its principal connective and polarity.

The constraints associated to π_α-positions are called *assertions* while those associated to π_β-positions are called *requirements*. The atomic labels introduced by positions of principal type π_α (resp. π_β) are called constants (resp. variables). Given a set of positions p, the associated sets of assertions and requirements are $\mathcal{K}_\alpha(p) = \{ kon(u) \mid u \in P_\alpha(p) \}$ and $\mathcal{K}_\beta(p) =$



u	$pol(u)$	$f(u)$	$ptyp(u)$	$styp(u)$	$slab(u)$	$kon(u)$
a_0	0	$(\mathrm{P} * ((\mathrm{Q} -\!* \mathrm{R}) * \mathrm{S})) -\!* ((\mathrm{P} * (\mathrm{Q} -\!* \mathrm{R})) * \mathrm{S})$	π_α	$-$	ϵ	$\epsilon a_0 \leq \tilde{a}_0$
a_1	1	$\mathrm{P} * ((\mathrm{Q} -\!* \mathrm{R}) * \mathrm{S})$	π_α	$\pi_{\alpha 1}$	a_0	$a_1\tilde{a}_1 \leq a_0$
a_2	1	P	$-$	$\pi_{\alpha 1}$	a_1	$-$
a_3	1	$(\mathrm{Q} -\!* \mathrm{R}) * \mathrm{S}$	π_α	$\pi_{\alpha 2}$	$\tilde{a}_1$	$a_3\tilde{a}_3 \leq \tilde{a}_1$
a_4	1	$\mathrm{Q} -\!* \mathrm{R}$	π_β	$\pi_{\alpha 1}$	a_3	$a_3 a_4 \leq \tilde{a}_4$
a_5	0	Q	$-$	$\pi_{\beta 1}$	a_4	$-$
a_6	1	R	$-$	$\pi_{\beta 2}$	$\tilde{a}_4$	$-$
a_7	1	S	$-$	$\pi_{\alpha 2}$	$\tilde{a}_3$	$-$
a_8	0	$(\mathrm{P} * (\mathrm{Q} -\!* \mathrm{R})) * \mathrm{S}$	π_β	$\pi_{\alpha 2}$	$\tilde{a}_0$	$a_8\tilde{a}_8 \leq \tilde{a}_0$
a_9	0	$\mathrm{P} * (\mathrm{Q} -\!* \mathrm{R})$	π_β	$\pi_{\beta 1}$	a_8	$a_9\tilde{a}_9 \leq a_8$
a_{10}	0	P	$-$	$\pi_{\beta 1}$	a_9	$-$
a_{11}	0	$\mathrm{Q} -\!* \mathrm{R}$	π_α	$\pi_{\beta 2}$	$\tilde{a}_9$	$\tilde{a}_9 a_{11} \leq \tilde{a}_{11}$
a_{12}	1	Q	$-$	$\pi_{\alpha 1}$	a_{11}	$-$
a_{13}	0	R	$-$	$\pi_{\alpha 2}$	$\tilde{a}_{11}$	$-$
a_{14}	0	S	$-$	$\pi_{\beta 2}$	$\tilde{a}_8$	$-$

Figure 3: Indexed formula tree of $(\mathrm{P} * ((\mathrm{Q} -\!* \mathrm{R}) * \mathrm{S})) -\!* ((\mathrm{P} * (\mathrm{Q} -\!* \mathrm{R})) * \mathrm{S})$

$\{ kon(u) \mid u \in P_\beta(p) \}$ respectively. The associated sets of constants and variables are then defined as $\Sigma_\alpha(p) = \{ x \mid x \text{ occurs in } \mathcal{K}_\alpha(p) \}$ and $\Sigma_\beta(p) = \{ x \mid x \text{ occurs in } \mathcal{K}_\beta(p) \}$. We set $\Sigma_{\alpha\beta}(p) = \Sigma_\alpha(p) \cup \Sigma_\beta(p)$ and define $\mathcal{L}_\alpha(p)$, $\mathcal{L}_\beta(p)$ and $\mathcal{L}_{\alpha\beta}(p)$ as the sets of atomic and compound labels generated by $\Sigma_\alpha(p)$, $\Sigma_\beta(p)$ and $\Sigma_{\alpha\beta}(p)$ respectively. For readability, we omit the set of positions p in notations whenever p is the set $\mathcal{P}os$ of all positions.

3.2.1 An Example (part 1)

Let us consider the formula $\mathrm{A} \equiv (\mathrm{P} * ((\mathrm{Q} -\!* \mathrm{R}) * \mathrm{S})) -\!* ((\mathrm{P} * (\mathrm{Q} -\!* \mathrm{R})) * \mathrm{S})$ that is represented as a syntax tree each node of which being identified with a position (see the tree at the righthand side of Figure 3). Moreover, we can associate an indexed formula tree (with labelled signed formulae as nodes), inductively built from $(\mathrm{A})^0 : \epsilon$ and the rules of Figure 2. This tree is the one at the lefthand side of Figure 3. In parallel, we have the generation of constraints for the positions of principal type π_α and π_β (see $kon(u)$ in the table of Figure 3). Then, in this case we can deduce that $P_\alpha = \{ a_0, a_1, a_3, a_{11} \}$, $P_\beta = \{ a_4, a_8, a_9 \}$, $S_{\alpha 1} = \{ a_1, a_2, a_4, a_{12} \}$, $S_{\beta 1} = \{ a_5, a_9, a_{10} \}$, $S_{\alpha 2} = \{ a_3, a_7, a_8, a_{13} \}$, $S_{\beta 2} = \{ a_6, a_{11}, a_{14} \}$. In addition the assertions and requirements are

$$\mathcal{K}_\alpha = \{ \epsilon a_0 \leq \tilde{a}_0, a_1\tilde{a}_1 \leq a_0, a_3\tilde{a}_3 \leq \tilde{a}_1, \tilde{a}_9 a_{11} \leq \tilde{a}_{11} \}$$
$$\mathcal{K}_\beta = \{ a_8\tilde{a}_8 \leq \tilde{a}_0, a_9\tilde{a}_9 \leq a_8, a_3 a_4 \leq \tilde{a}_4 \}$$

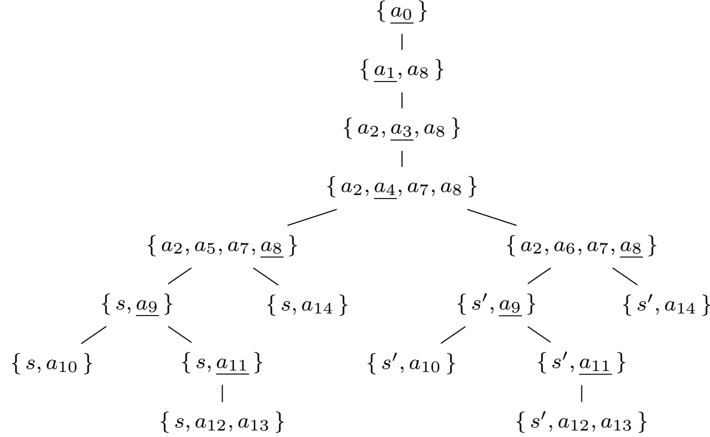


Figure 4: Path reduction with $s = a_2, a_5, a_7$ and $s' = a_2, a_6, a_7$

The constants and variables are

$$\Sigma_\alpha = \{ a_0, \tilde{a}_0, a_1, \tilde{a}_1, a_3, \tilde{a}_3, a_{11}, \tilde{a}_{11} \} \qquad \Sigma_\beta = \{ a_4, \tilde{a}_4, a_8, \tilde{a}_8, a_9, \tilde{a}_9 \}.$$

3.3 Paths and Connections

In this section, we adapt the standard notions of path, connection and spanning set of connections in the context of labels and constraints.

Definition 5 (Paths)**.** *Let* A *be an indexed formula. The set of* paths *in* A *is inductively defined as follows:*

1. $\{ a_0 \}$ *is a path, where* a_0 *is the root position.*
2. *If* s *is a path such that* $u \in s$ *then*
   - *if* $ptyp(u) \in \{ \pi_\alpha \}$ *then* $s \backslash \{ u \} \cup \{ [u]_1, [u]_2 \}$ *is a path,*
   - *if* $ptyp(u) \in \{ \pi_\beta \}$ *then* $s \backslash \{ u \} \cup \{ [u]_1 \}$ *and* $s \backslash \{ u \} \cup \{ [u]_2 \}$ *are paths*[2]*.*

We say that a path s' in A is obtained from a path s by *reduction* on the indexed position u if it results from s using the second clause of Definition 5. An *atomic path* is a path that only contains atomic indexed positions. Consequently an atomic path is non-reducible and is always a leaf of a path reduction tree. A *configuration* of A is a finite set of paths in A.

Definition 6 (Reduction)**.** *A* reduction *of an indexed formula* A *is a finite sequence* $(\mathcal{S}_i)_{1 \leq i \leq n}$ *of configurations in* A *such that* $\mathcal{S}_{i+1}$ *is obtained from* $\mathcal{S}_i$ *by reduction of a position* u *in a path* s *of* $\mathcal{S}_i$ *following Definition 5. We say that* $\mathcal{S}_{i+1}$ *is obtained by reduction of* $\mathcal{S}_i$ *of* u *in* s*. A reduction* $(\mathcal{S}_i)_{1 \leq i \leq n}$ *is said* atomic *if all the paths of* $\mathcal{S}_n$ *are atomic.*

Definition 7 (Connection)**.** *Let* A *be an indexed formula, a* connection c *in* A *is:*

1. *a pair* $\langle u, v \rangle$ *of atomic positions such that* $f(u) = f(v)$*,* $pol(u) = 1$ *and* $pol(v) = 0$*, or*

[2] Let us remark that the branching indicates non-determinism.

2. a pair $\langle a_0, v \rangle$ *such that* $f(v) = 1$ *and* $pol(v) = 0$.

The first case corresponds to an atomic axiom rule (ax) in LMILL, while the second one corresponds to the rule 1_R*. Let us also note that the first position of a connection is the one with the positive polarity.*

We denote $\mathcal{C}on$ the set of connections in A. The constraint $kon(c)$ associated to a connection $c = \langle u, v \rangle$ is defined as $kon(c) = slab(u) \leq slab(v)$. For a 1-connection $c = \langle a_0, v \rangle$ we have $kon(\langle a_0, v \rangle) = \epsilon \leq slab(v)$ since $slab(a_0) = \epsilon$. In order to distinguish these constraints from the assertions and requirements they are called *connection constraints*. Moreover, the notions of upward and downward positions are extended to connections as follows: $c\uparrow = \{ u, v \}\uparrow$ and $c\downarrow = \{ u, v \}\downarrow$.

Definition 8 (MILL-cover)**.** *Let* A *be an indexed formula, a connection* $c = \langle u, v \rangle$ *in* A covers *a path* s *in* A*, denoted* $c \succ s$*, if* $v \in s$ *and* $(u \neq a_0 \Rightarrow u \in s)$*. Let* S *be a set of paths in* A*, a* cover *of* S *is a set* $C = \{ (s, \langle u, v \rangle) \mid s \in S \text{ and } \langle u, v \rangle \in \mathcal{C}on \text{ and } \langle u, v \rangle \succ s \}$ *such that*

$$(s, \langle u, v \rangle) \in C \text{ and } (s, \langle u', v' \rangle) \in C \Rightarrow u = u' \text{ and } v = v'.$$

A cover of A *is a cover of the set of atomic paths in* A*.*

3.3.1 An Example (part 2)

The reduction of the initial path $\{ a_0 \}$ results in six atomic paths as depicted in Figure 4. At each step, we indicate the position which is reduced with an underscore. For conciseness, we write s and s' as shortcuts for a_2, a_5, a_7 and a_2, a_6, a_7. The set $C =$

$$\{ (s_1, \langle a_2, a_{10} \rangle), (s_2, \langle a_{12}, a_5 \rangle), (s_3, \langle a_7, a_{14} \rangle), (s_4, \langle a_2, a_{10} \rangle), (s_5, \langle a_6, a_{13} \rangle), (s_6, \langle a_7, a_{14} \rangle) \}$$

covers all atomics paths. Indeed, $\langle a_2, a_{10} \rangle$ covers paths s_1 and s_4, $\langle a_{12}, a_5 \rangle$ covers the path s_2, $\langle a_7, a_{14} \rangle$ covers the path s_3 and s_6, $\langle a_6, a_{13} \rangle$ covers the path s_5. We observe that connections $\langle a_2, a_{10} \rangle$ and $\langle a_7, a_{14} \rangle$ cover two atomic paths at the same time.

3.4 Characterizing MILL-Provability

In this section we define a connection-based characterization of MILL-provability which relies on the notions of substitution and certification.

Definition 9 (Substitution)**.** *Let* A *be an indexed formula. A* substitution for A *is an application* $\sigma : \Sigma_\beta \to \mathcal{L}_\alpha$*, that can be extended to labels and constraints as follows:*

- $x\sigma = x$ *if* x *is a constant or if* $x = \epsilon$*,*
- $(x \circ y)\sigma = x\sigma \circ y\sigma$*,*
- $(x \leq y)\sigma = x\sigma \leq y\sigma$*.*

Moreover a substitution σ for an indexed formula A induces an *instantiation relation* on indexed positions, denoted $\prec$, such that

$$(\forall u, v \in \mathcal{P}os)(u \prec v \text{ iff } v \in P_\beta \text{ and } (u \subseteq v\sigma \text{ or } \tilde{u} \subseteq v\sigma).$$

Definition 10 (Certification)**.** *Let* A *be an indexed formula. A* certification for A *is an application* $\gamma : P_\beta \to \wp(P_\alpha)$ *that associates a set of* π_α*-positions with any* π_β*-position in* A*.*

A certification γ for an indexed formula A induces a *deduction relation* on indexed positions, denoted $\sqsubset$, such that

$$(\forall u, v \in \mathcal{P}os)(u \sqsubset v \text{ iff } v \in P_\beta \text{ and } u \in v\gamma).$$

An expression $u \sqsubset v$ means that v is deduced from u (in A). The relations of domination, instantiation and deduction induce a *reduction relation* $\lhd = (\ll \cup \prec \cup \sqsubset)^+$ where $(\cdot)^+$ represents the transitive closure. An expression $u \lhd v$ means that u must be reduced before v (in A). Now we can express the provability conditions in terms of connections.

Definition 11 (Complementarity)**.** *Let s be a path in an indexed formula* A *and σ be a substitution, a connection c in* A *is* complementary in s under σ *if $c \succ s$ and $\mathcal{K}_\alpha(s\!\uparrow)\sigma \models\!\!\approx kon(c)\sigma$. A path s is* complementary under σ *if there exists a connection that is complementary in s under σ. A cover C of a set of paths in* A *is* complementary under σ *if $(\forall (s, c) \in C)$ c is complementary in s under σ.*

Definition 12 (Provability)**.** *A formula* A *is* provable *if there exist a cover C of the set of atomic paths of* A*, a substitution σ and a certification γ for* A *such that:*

1. *the reduction relation $\lhd$ is irreflexive,* $(C1)$
2. $\forall (s, \langle u, v \rangle) \in C, \forall w \in P_\beta(s\!\uparrow), w\gamma \subseteq P_\alpha(s\!\uparrow),$ $(C2)$
3. $\forall (s, \langle u, v \rangle) \in C, \forall w \in P_\beta(s\!\uparrow), k(w\gamma)\sigma \models\!\!\approx k(w)\sigma,$ $(C3)$
4. $\forall (s, \langle u, v \rangle) \in C, \forall x \in \Sigma_\beta(s\!\uparrow), x\sigma \in \mathcal{L}_\alpha(s\!\uparrow),$ $(C4)$
5. $\forall (s, \langle u, v \rangle) \in C, \langle u, v \rangle$ *is complementary in s under σ.* $(C5)$

The first condition induces the acyclicity of the graph associated to $\lhd$ and then the existence of a reduction (decomposition) order of the formula A that respects the precedence constraints between π_α and π_β positions. The second and third conditions ensure that, in an atomic path s, every requirement introduced by a position of principal type π_β must be introduced before the two positions of the connection that makes the path s complementary and should be certified by assertions corresponding to positions of principal type π_α that can be reduced before this requirement in a reduction from the initial path $\{ a_0 \}$ to s. In a similar way the fourth condition means that each variable, introduced before reaching the connection that makes an atomic path complementary, is instantiated by a label composed from constants that can be reduced before this variable in a reduction from the initial path $\{ a_0 \}$ to s.

3.4.1 An Example (part 3)

The reduction path process from $\{ a_0 \}$ provides the following atomic paths:
$s_1 = \{ a_2, a_5, a_7, a_{10} \}$, $s_2 = \{ a_2, a_5, a_7, a_{12}, a_{13} \}$, $s_3 = \{ a_2, a_5, a_7, a_{14} \}$, $s_4 = \{ a_2, a_6, a_7, a_{10} \}$, $s_5 = \{ a_2, a_6, a_7, a_{12}, a_{13} \}$ and $s_6 = \{ a_2, a_6, a_7, a_{14} \}$.
From the following cover $C =$

$$\{ (s_1, \langle a_2, a_{10} \rangle), (s_2, \langle a_{12}, a_5 \rangle), (s_3, \langle a_7, a_{14} \rangle), (s_4, \langle a_2, a_{10} \rangle), (s_5, \langle a_6, a_{13} \rangle), (s_6, \langle a_7, a_{14} \rangle) \}$$

we generate the set of constraints:

$$\mathcal{K}_C = \{ (s_1, a_1 \le a_9), (s_2, a_{11} \le a_4), (s_3, \tilde{a}_3 \le \tilde{a}_8), (s_4, \tilde{a}_1 \le \tilde{a}_9), (s_5, \tilde{a}_4 \le \tilde{a}_{11})(s_6, \tilde{a}_3 \le \tilde{a}_8) \}.$$

Then we consider the substitution:

$$a_9\sigma = a_1, \quad a_4\sigma = a_{11}, \quad \tilde{a}_8\sigma = \tilde{a}_3, \quad \tilde{a}_4\sigma = \tilde{a}_{11}, \quad a_8\sigma = X, \quad \tilde{a}_9\sigma = Y$$

Then we solve the following requirements:

1) $\epsilon a_0 \leq \tilde{a}_0,\ a_1\tilde{a}_1 \leq a_0,\ a_3\tilde{a}_3 \leq \tilde{a}_1,\ Y a_{11} \leq \tilde{a}_{11} \mid\!\approx a_3 a_{11} \leq \tilde{a}_{11}$

2) $\epsilon a_0 \leq \tilde{a}_0,\ a_1\tilde{a}_1 \leq a_0,\ a_3\tilde{a}_3 \leq \tilde{a}_1,\ Y a_{11} \leq \tilde{a}_{11} \mid\!\approx X\tilde{a}_3 \leq \tilde{a}_0$

3) $\epsilon a_0 \leq \tilde{a}_0,\ a_1\tilde{a}_1 \leq a_0,\ a_3\tilde{a}_3 \leq \tilde{a}_1,\ Y a_{11} \leq \tilde{a}_{11} \mid\!\approx a_1 Y \leq X$

From 1) we directly deduce $Y = a_3$ and also $a_4\gamma = \{ a_{11} \}$. The requirement in 1) is the one of the position a_4 and in order to verify it we use the assertion $a_3 a_{11} \leq \tilde{a}_{11}$ of position a_{11}. From 3) we deduce a trivial solution for X that is $X = a_1 a_3$ and also that $a_9\gamma = \emptyset$. Requirement 2) is verified because we have:

$$\dfrac{\dfrac{\dfrac{a_1 \leq a_1 \quad a_3\tilde{a}_3 \leq \tilde{a}_1}{a_1 a_3\tilde{a}_3 \leq a_1\tilde{a}_1}\,F \qquad a_1\tilde{a}_1 \leq a_0}{a_1 a_3 \tilde{a}_3 \leq a_0}\,T \qquad \epsilon a_0 \leq \tilde{a}_0}{a_1 a_3 \tilde{a}_3 \leq \tilde{a}_0}\,T$$

and then we deduce $a_8\gamma = \{ a_0, a_1, a_3 \}$ since $\epsilon a_0 \leq \tilde{a}_0$, $a_1\tilde{a}_1 \leq a_0$, $a_3\tilde{a}_3 \leq \tilde{a}_1$ are the respective assertions of a_0, a_1, a_3.

In order to verify the conditions $(C2)$ to $(C5)$, let us consider the following table:

$(s, \langle u, v \rangle)$	$P_\beta(\{ u, v \}\uparrow)$	$P_\alpha(s\uparrow)$	$\Sigma_\beta(\{ u, v \}\uparrow)$	$\Sigma_\alpha(s\uparrow)$
$(s_1, \langle a_2, a_{10} \rangle)$	a_8, a_9	a_0, a_1, a_3	$a_8, \tilde{a}_8, a_9, \tilde{a}_9$	$a_0, \tilde{a}_0, a_1, \tilde{a}_1, a_3, \tilde{a}_3$
$(s_2, \langle a_{12}, a_5 \rangle)$	a_4, a_8, a_9	a_0, a_1, a_3, a_{11}	$a_4, \tilde{a}_4, a_8, \tilde{a}_8, a_9, \tilde{a}_9$	$\tilde{a}_0, a_1, \tilde{a}_1, a_3, \tilde{a}_3, a_{11}, \tilde{a}_{11}$
$(s_3, \langle a_7, a_{14} \rangle)$	a_8, a_9	a_0, a_1, a_3	$a_8, \tilde{a}_8$	$a_0, \tilde{a}_0, a_1, \tilde{a}_1, a_3, \tilde{a}_3$
$(s_4, \langle a_2, a_{10} \rangle)$	a_8, a_9	a_0, a_1, a_3	$a_8, \tilde{a}_8, a_9, \tilde{a}_9$	$a_0, \tilde{a}_0, a_1, \tilde{a}_1, a_3, \tilde{a}_3$
$(s_5, \langle a_6, a_{13} \rangle)$	a_4, a_8, a_9	a_0, a_1, a_3, a_{11}	$a_4, \tilde{a}_4, a_8, \tilde{a}_8, a_9, \tilde{a}_9$	$\tilde{a}_0, a_1, \tilde{a}_1, a_3, \tilde{a}_3, a_{11}, \tilde{a}_{11}$
$(s_6, \langle a_7, a_{14} \rangle)$	a_8, a_9	a_0, a_1, a_3	$a_8, \tilde{a}_8$	$a_0, \tilde{a}_0, a_1, \tilde{a}_1, a_3, \tilde{a}_3$

We have $a_9\gamma = \emptyset \subseteq P_\alpha(s\uparrow)$ for all paths $s \in \{ s_1, s_2, s_4, s_5, s_6 \}$ and $a_8\gamma = \{ a_0, a_1, a_3 \} \subseteq P_\alpha(s\uparrow)$ for all paths $s \in \{ s_1 \ldots s_6 \}$. Moreover, for all paths $s \in \{ s_2, s_5 \}$, $a_4\gamma = \{ a_{11} \} \subseteq P_\alpha(s\uparrow)$. Then the condition $(C2)$ is verified. In addition, for all paths $s \in \{ s_1, s_2, s_4, s_5 \}$ we have $a_9\sigma = a_1 \in \mathcal{L}_\alpha(s\uparrow)$ and $\tilde{a}_9\sigma = a_3 \in \mathcal{L}_\alpha(s\uparrow)$ for all paths $s \in \{ s_1, s_2, s_3, s_4, s_5, s_6 \}$ we have $a_8\sigma = a_1 a_3 \in \mathcal{L}_\alpha(s\uparrow)$ and $\tilde{a}_8\sigma = \tilde{a}_3 \in \mathcal{L}_\alpha(s\uparrow)$, and for all paths $s \in \{ s_2, s_5 \}$ we have $a_4\sigma = a_{11} \in \mathcal{L}_\alpha(s\uparrow)$ and $\tilde{a}_4\sigma = \tilde{a}_{11} \in \mathcal{L}_\alpha(s\uparrow)$.

The last thing to do is to compute the reduction relation $\lhd$ that is obtained by the transitive closure of the domination relation $\ll$, the instantiation relation $\sqsubset$ and the deduction relation $\sqsubset$. The instantiation relation induced by σ is

$$a_1 \sqsubset a_9, a_3 \sqsubset a_9, a_1 \sqsubset a_8, a_{11} \sqsubset a_4$$

and the deduction relation induced by γ is

$$a_0 \sqsubset a_8, a_1 \sqsubset a_8, a_3 \sqsubset a_8, a_{11} \sqsubset a_4.$$

The reduction relation $\lhd$ is represented in Figure 5. As the graph is acyclic, A is valid in MILL.

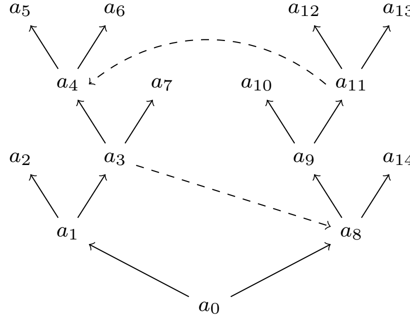


Figure 5: Reduction order for (P ∗ ((Q −∗ R) ∗ S) −∗ ((P ∗ (Q −∗ R)) ∗ S)

4 Properties of the Characterization

In this section we prove the soundness and completeness of characterization given in Definition 12. The completeness is proved by showing that any formula provable in the LMILL sequent calculus is also provable according to the connection-based characterization (CMILL-provable).

Definition 13 (Complete reduction)**.** *Let* A *be an indexed formula and* C *be a cover of* A*, a reduction* $(\mathcal{S}_i)_{1\leq i\leq n}$ *in* A *is* complete *for* C *if* C *is a cover of* $\mathcal{S}_n$*.*

Definition 14 (Proper reduction)**.** *Let* A *be an indexed formula and* σ *be a substitution for* A*, a reduction* $(\mathcal{S}_i)_{1\leq i\leq n}$ *is* σ-proper *iff*

1. $\forall S \in (\mathcal{S}_i)_{1\leq i\leq n}, \forall s \in S, \mathcal{K}_\alpha(s\uparrow)\sigma \mid\!\approx \mathcal{K}_\beta(s\uparrow)\sigma$ *and*
2. $\forall S \in (\mathcal{S}_i)_{0\leq i\leq n}, \forall s \in S, \forall x \in \Sigma_\beta(s\uparrow), x\sigma \in \mathcal{L}_\alpha(s\uparrow)$.

Definition 15 (Realization)**.** *Let* A *be an indexed formula and* s *be a path in* A*. An* interpretation *of* s *in a resource model* $\mathcal{R} = \langle (M, \sqsubseteq, \cdot, e), \models, [\![-]\!] \rangle$ *is a function* $\|-\| : \Sigma_\alpha(s\uparrow) \to M$ *that can be extended to labels* $\mathcal{L}_\alpha(s\uparrow)$ *with* $\|\epsilon\| = e$ *and* $\|xy\| = \|x\| \cdot \|y\|$*.*
Given a substitution σ *for* A*, we denote* $\|-\|_\sigma$ *the composed function* $\|-\| \circ \sigma$ *from the set* $\mathcal{L}_{\alpha\beta}(s\uparrow)$ *of labels of* s *to the set of worlds* M *of* $\mathcal{R}$*.*
A realization *of* s *is a couple* $(\|-\|, \sigma)$ *such that:*

1. *For all assertions* $x \leq y \in \mathcal{K}_\alpha(s\uparrow)$*,* $\|x\|_\sigma \sqsubseteq \|y\|_\sigma$*.*
2. *For all positions* $u \in s$ *such that* $lsf(u) = \mathrm{A}^1 : x$*,* $\|x\|_\sigma \models \mathrm{A}$*.*
3. *For all positions* $u \in s$ *such that* $lsf(u) = \mathrm{A}^0 : x$*,* $\|x\|_\sigma \not\models \mathrm{A}$*.*

A path is realizable *if there exists a realization of* s *in a model* $\mathcal{R}$*. A configuration is* realizable *if at least one of its paths is realizable.*

Lemma 1. *Let* s *be a path in an indexed formula* A*,* $(\|-\|, \sigma)$ *be a realization of* s *in a model* $\mathcal{R} = (M, e, \cdot, \sqsubseteq, \models)$ *and* $K \subseteq \mathcal{K}_\alpha(s\uparrow)$ *a subset of assertions associated to* s*. If* $K\sigma \mid\!\approx (x \leq y)\sigma$ *then* $\|x\|_\sigma \sqsubseteq \|y\|_\sigma$*.*

Proof. By definition of a realization, for any assertion $x' \leq y'$ of K we have $\|x'\|_\sigma \sqsubseteq \|y'\|_\sigma$. By hypothesis, under σ, the constraint $x \leq y$ is deduced (in the K-deduction system) from

assertions of K by rules expressing reflexivity and transitivity of $\leq$ and also compatibility of label composition with $\leq$. Moreover, by definition of a preorder, $\sqsubseteq$ is reflexive and transitive and by definition of a MILL-model, world composition is compatible with $\sqsubseteq$ in $\mathcal{R}$. Consequently, the rules of the K-deduction system transfer the notion of realizability from premisses to conclusion. □

Lemma 2. *Let* A *be an indexed formula,* σ *be a substitution for* A *and* $(\mathcal{S}_i)_{1\leq i\leq n}$ *be a* σ*-proper reduction for* A*, if* $\mathcal{S}_i$ *is* σ*-realizable then* $\mathcal{S}_{i+1}$ *is* σ*-realizable.*

Proof. As the configuration $\mathcal{S}_i$ is realizable under σ, it contains a path s that is realizable under σ in a model $(M, e, \cdot, \sqsubseteq, \models)$ for a interpretation $\|-\|$. Let us suppose that $\mathcal{S}_{i+1}$ is obtained by a reduction of a position u in a path of $\mathcal{S}$. If $u \notin s$ then $\mathcal{S}_{i+1}$ remains realizable under σ because it always contains the path s. Otherwise, we proceed by case analysis depending on the principal connective of $lsf(u) = \mathrm{A}^{\mathrm{pol}} : x$ and show that $\mathcal{S}_{i+1}$ remains realizable under σ:

- Case $(\mathrm{B} * \mathrm{C})^1 : x$
  The path s is reduced into s' by replacing the position u by its children positions $[u]_1$ and $[u]_2$ such that $lsf([u]_1) = \mathrm{B}^1 : u$ and $lsf([u]_2) = \mathrm{C}^1 : \tilde{u}$. Then $\Sigma_\alpha(s'\!\uparrow) = \Sigma_\alpha(s\!\uparrow) \cup \{ u, \tilde{u} \}$ and $\mathcal{K}_\alpha(s'\!\uparrow) = \mathcal{K}_\alpha(s\!\uparrow) \cup \{ u\tilde{u} \leq x \}$. By hypothesis, $\|-\|_\sigma$ is a realization of s and then $\|x\|_\sigma \models \mathrm{B} * \mathrm{C}$. Thus, by definition of $\models$, there exist two worlds $m, n \in M$ such that $m \cdot n \sqsubseteq \|x\|_\sigma$, $m \models \mathrm{B}$ and $n \models \mathrm{C}$. We then extend $\|-\|$ to u and $\tilde{u}$ by defining $\|u\| = m$ and $\|\tilde{u}\| = n$ to obtain $\|u\|_\sigma \models \mathrm{B}$, $\|\tilde{u}\|_\sigma \models \mathrm{C}$ and $\|u\tilde{u}\|_\sigma = \|u\|_\sigma \cdot \|\tilde{u}\|_\sigma \sqsubseteq \|x\|_\sigma$. Consequently s' is realizable under σ.

- Case $(\mathrm{B} * \mathrm{C})^0 : x$
  The path is reduced into two paths s' and s'' such that $\Sigma_\beta(s'\!\uparrow) = \Sigma_\beta(s\!\uparrow) \cup \{ u \}$, $\Sigma_\beta(s''\!\uparrow) = \Sigma_\beta(s\!\uparrow) \cup \{ \tilde{u} \}$ and $\mathcal{K}_\beta(s'\!\uparrow) = \mathcal{K}_\beta(s''\!\uparrow) = \mathcal{K}_\beta(s\!\uparrow) \cup \{ u\tilde{u} \leq x \}$. By hypothesis the reduction $(\mathcal{S}_i)_{1\leq i\leq n}$ is σ-proper and we have $\mathcal{K}_\alpha(s'\!\uparrow)\sigma \mathrel{|\!\approx} \mathcal{K}_\beta(s'\!\uparrow)\sigma$ and $\mathcal{K}_\alpha(s''\!\uparrow)\sigma \mathrel{|\!\approx} \mathcal{K}_\beta(s''\!\uparrow)\sigma$. In particular $\mathcal{K}_\alpha(s\!\uparrow) = \mathcal{K}_\alpha(s'\!\uparrow) = \mathcal{K}_\alpha(s''\!\uparrow)$, $\mathcal{K}_\beta(s'\!\uparrow) = \mathcal{K}_\beta(s''\!\uparrow)$ and $u\tilde{u} \leq x \in \mathcal{K}_\beta(s'\!\uparrow)$ entail $\mathcal{K}_\alpha(s\!\uparrow)\sigma \mathrel{|\!\approx} (u\tilde{u} \leq x)\sigma$. As $\|-\|_\sigma$ is a realization of s, we have $\|x\|_\sigma \not\models \mathrm{B} * \mathrm{C}$ and from Lemma 1 we deduce $\|u\|_\sigma \cdot \|\tilde{u}\|_\sigma \sqsubseteq \|x\|_\sigma$. Then, by definition of $\models$, for all worlds $m, n \in M$ such that $m \cdot n \sqsubseteq \|x\|_\sigma$, we have either $m \not\models \mathrm{B}$ or $n \not\models \mathrm{C}$. In particular, we have either $\|u\|_\sigma \not\models \mathrm{B}$ or $\|\tilde{u}\|_\sigma \not\models \mathrm{C}$. Consequently, either s' is realizable under σ or s'' is realizable under σ.

- The other cases are similar.

□

Lemma 3. *Let* A *be an indexed formula and* σ *be a substitution for* A*. If a path* s *is complementary under* σ *then it is not realizable under* σ*.*

Proof. Let us suppose that s contains a connection $\langle\, u, v \,\rangle$ such that $f(u) = f(v)$, $pol(u) = 1$, $pol(v) = 0$ and $\mathcal{K}_\alpha(s\!\uparrow)\sigma \mathrel{|\!\approx} slab(u)\sigma \leq slab(v)\sigma$. If s is realizable under σ for an interpretation $\|-\|$ in a model $\mathcal{R}$ then $\|slab(u)\|_\sigma \models f(u)$, $\|slab(v)\|_\sigma \not\models f(u)$ and $\|slab(u)\|_\sigma \sqsubseteq \|slab(v)\|_\sigma$, which is contradictory because by Kripke monotonicity $\|slab(u)\|_\sigma \sqsubseteq \|slab(v)\|_\sigma$ and $\|slab(u)\|_\sigma \models f(u)$ entail $\|slab(v)\|_\sigma \models f(u)$. The case of a 1-connection is similar. □

Theorem 2 (Soundness)**.** *If a formula* A *is CMILL-provable then it is valid.*

Proof. As A is provable, there is a cover C of the set of atomic paths of A, a substitution σ and a certification γ for A satisfying the conditions of Definition 12.
Let us suppose that A is not valid. Then, there exists a model $\mathcal{R} = (M, e, \cdot, \sqsubseteq, \models)$ such that $e \not\models$ A. The initial configuration $\mathcal{S}_1 = \{ \{ a_0 \} \}$ is then trivially realizable under σ by considering the interpretation $\|-\|$ the domain of which is empty. It is easy to show that conditions $(C1)$ to $(C5)$ entail the existence of a reduction $(\mathcal{S}_i)_{1 \leq i \leq n}$ from $\mathcal{S}_1$ that is complete for C, σ-proper and such that all paths of $\mathcal{S}_n$ contain at least a connection of C. As $\mathcal{S}_1$ is realizable under σ, Lemma 2 entails that the configuration $\mathcal{S}_n$ is also realizable under σ. But then, by Lemma 3, we deduce that $\mathcal{S}_n$ cannot be complementary, which is a contradiction. Therefore, A is valid. □

Let us now consider the question of completeness of this characterization.

Theorem 3 (Completeness)**.** *If a formula* A *is valid then* A *is CMILL-provable.*

Proof. From the sound and completeness of MILL-models, it is sufficient to prove that if A is LMILL-provable then A is CMILL-provable (provable by the connection characterization). The proof is by induction on a LMILL-proof of A, knowing that a sequent $\Gamma \vdash$ A is provable in LMILL if and only if the formula $\Phi_\Gamma \multimap$ A is provable in LMILL, where Φ_Γ is the formula obtained by replacing each comma in the context Γ with multiplicative conjunction $*$.

- Case ax: the axiom A $\vdash$ A corresponds to the formula A $\multimap$ A which is trivially CMILL-provable.

- Case $\multimap_R$: By induction hypothesis we suppose that the sequent Γ , A $\vdash$ B is provable and we show that the sequent $\Gamma \vdash$ A $\multimap$ B is also provable. If Γ, A $\vdash$ B is CMILL-provable an atomic reduction $\mathcal{R}_1 = (\mathcal{S}_i)_{1 \leq i \leq n}$ of $((\Phi_\Gamma * \mathrm{A}) \multimap \mathrm{B})$, a cover C of $\mathcal{S}_n$, a substitution σ and a certification γ for A that satisfy the conditions of Definition 12.

  From the atomic reduction $\mathcal{R}_1$ for $((\Phi_\Gamma * \mathrm{A}) \multimap \mathrm{B})$ we can build an atomic reduction $\mathcal{R}_2$ for $(\Phi_\Gamma \multimap (\mathrm{A} \multimap \mathrm{B}))$. On the left-hand side of the next figure, we describe the first steps of the reduction $\mathcal{R}_1$ and, on the right-hand side, we describe the first steps of the corresponding reduction $\mathcal{R}_2$. We represent here a path s as a set of signed formulae and not as a set of positions, namely we have $lsf(s) = \{ lsf(u) \mid u \in s \}$.

$\{ ((\Phi_\Gamma * \mathrm{A}) \multimap \mathrm{B})^0 : \epsilon \}$	$\{ (\Phi_\Gamma \multimap (\mathrm{A} \multimap \mathrm{B}))^0 : \epsilon \}$
$\mid$	$\mid$
$\{ (\Phi_\Gamma * \mathrm{A})^1 : a_0, \mathrm{B}^0 : \tilde{a}_0 \}$	$\{ {\Phi_\Gamma}^1 : a_0, (\mathrm{A} \multimap \mathrm{B})^0 : \tilde{a}_0 \}$
$\mid$	$\mid$
$\{ {\Phi_\Gamma}^1 : a_1, \mathrm{A}^1 : \tilde{a}_1, \mathrm{B}^0 : \tilde{a}_0 \}$	$\{ {\Phi_\Gamma}^1 : a_0, \mathrm{A}^1 : a_i, \mathrm{B}^0 : \tilde{a}_i \}$
$\mid$	$\mid$
$\vdots$	$\vdots$
$\mathcal{R}_1$	$\mathcal{R}_2$

  We observe that the first reduction steps in $\mathcal{R}_1$ and $\mathcal{R}_2$ lead to paths containing the same signed formulae, modulo a renaming of a_1 into a_0, of $\tilde{a}_1$ into a_i and of $\tilde{a}_0$ into $\tilde{a}_i$. Consequently, modulo the renaming, we can reduce $\mathcal{R}_2$ by applying exactly the same reduction steps than for $\mathcal{R}_1$. Then, after the two first steps previously described, $\mathcal{R}_1$ and $\mathcal{R}_2$ introduce the same signed formulae and then the same constraints (assertions and requirements) between labels, modulo renaming.
  Moreover, the assertions $\{ a_0\tilde{a}_0 \leq, a_1\tilde{a}_1 \leq \tilde{a}_0 \}$ introduced in the two first steps of $\mathcal{R}_1$ entail relations between labels $\tilde{a}_0, a_1, \tilde{a}_1$ of $\mathcal{R}_1$ weaker than the ones between labels $a_0, a_i, \tilde{a}_i$ in

$$\frac{}{\Gamma, x \le y, \mathrm{A} : x \vdash \mathrm{A} : y, \Delta}\ id \qquad \frac{}{\Gamma, \epsilon \le x \vdash x : 1, \Delta}\ 1_R \qquad \frac{\Gamma, \epsilon \le x \vdash \Delta}{\Gamma, 1 : x \vdash \Delta}\ 1_L$$

$$\frac{\Gamma, ab \le x, \mathrm{A} : a, \mathrm{B} : b \vdash \Delta}{\Gamma, \mathrm{A} * \mathrm{B} : x \vdash \Delta}\ *_L \qquad \frac{yz \le x, \Gamma \vdash \mathrm{A} : y, \Delta \qquad yz \le x, \Gamma \vdash \mathrm{B} : z, \Delta}{\Gamma, yz \le x \vdash \mathrm{A} * \mathrm{B} : x, \Delta}\ *_R$$

$$\frac{\Gamma, xy \le z \vdash \mathrm{A} : y, \Delta \qquad \Gamma, xy \le z, \mathrm{B} : z \vdash \Delta}{\Gamma, xy \le z, \mathrm{A} \multimap \mathrm{B} : x \vdash \Delta}\ \multimap_L \qquad \frac{\Gamma, \mathrm{A} : a, xa \le b \vdash \mathrm{B} : b, \Delta}{\Gamma \vdash \mathrm{A} \multimap \mathrm{B} : x, \Delta}\ \multimap_R$$

$$\frac{\Gamma, x \le x \vdash \Delta}{\Gamma \vdash \Delta}\ R \qquad \frac{\Gamma, x \le z, x \le y, y \le z \vdash \Delta}{\Gamma, x \le y, y \le z \vdash \Delta}\ T \qquad \frac{\Gamma, xx' \le yy', x \le x', y \le y' \vdash \Delta}{\Gamma, x \le y, x' \le y' \vdash \Delta}\ F$$

Side conditions:
– In $*_L$ and $\multimap_R$, the constants a and b do not occur in the conclusion.
– In R the label x must already occur in the conclusion.

Figure 6: Labelled Sequent Calculus GMILL

$\mathcal{R}_2$ by assertions $\{\, a_0\tilde{a}_0 \le, \tilde{a}_0 a_i \le \tilde{a}_i \,\}$ introduced in the two first steps of $\mathcal{R}_2$. In fact, $\mathcal{R}_1$ imposes $slab(\Phi_\Gamma)slab(\mathrm{A}) \le slab(\mathrm{B})$ while $\mathcal{R}_2$ imposes $slab(\Phi_\Gamma)slab(\mathrm{A}) = slab(\mathrm{B})$. Consequently, as $\mathcal{R}_1$ leads to a set of atomic paths satisfying the conditions of Definition 12, it is the same for $\mathcal{R}_2$ by induction hypothesis.

- Case $*_L$: immediate by the translation Φ, because $\Phi_{\Gamma,\mathrm{A},\mathrm{B}} = \Phi_{\Gamma,\mathrm{A}*\mathrm{B}}$.
- The other cases are similar.

□

5 A Labelled Sequent Calculus for MILL

From the previous characterization we can derive a sound and complete labelled sequent calculus GMILL[3] for MILL. Soundness and completeness are easy consequences of Theorem 2 and Theorem 3.

Labels and constraints for GMILL are defined similarly as in Section 3.1 except that GMILL does not make use of variables. The sequent calculus GMILL deals with sequents of the form $\Gamma \vdash \Delta$ where Γ and Δ are multisets containing labelled formulas, Γ being allowed to also contain constraints. Labelled formulas are pairs (A, x), written $\mathrm{A} : x$, where A is a formula and x is a label. Label constraints occurring in Γ are called assertions. We denote Γ_r the restriction of Γ to its constraints. GMILL does not have explicit requirements. Instead, the rules $*_R$ and $\multimap_L$ are required to have a specific constraint (which in the connection-based characterization

[3] The G in GMILL is reminiscent of the fact that labels and label-constraints can be viewed as a graphical structure we usually call a *resource-graph* in related works.

$$\dfrac{}{\ldots, a_1 \leq a_1, \ldots, \mathrm{P} : a_1, \ldots \vdash \ldots, \mathrm{P} : a_1}$$
$$\Pi_3$$

$$\dfrac{}{\ldots, a_3 \leq a_3, \ldots, \mathrm{Q} \multimap \mathrm{R} : a_3, \ldots \vdash \ldots, \mathrm{Q} \multimap \mathrm{R} : a_3}$$
$$\Pi_4$$

$$\dfrac{\dfrac{\dfrac{\Pi_3 \quad \Pi_4}{a_1a_3 \leq a_1a_3, a_3 \leq a_3, \ldots, a_1 \leq a_1, \ldots, \mathrm{P} : a_1, \mathrm{Q} \multimap \mathrm{R} : a_3, \ldots \vdash \ldots, \mathrm{P} * (\mathrm{Q} \multimap \mathrm{R}) : a_1a_3}\, *_R}{a_3 \leq a_3, \ldots, a_1 \leq a_1, \ldots, \mathrm{P} : a_1, \mathrm{Q} \multimap \mathrm{R} : a_3, \ldots \vdash \ldots, \mathrm{P} * (\mathrm{Q} \multimap \mathrm{R}) : a_1a_3}\, F}{a_1a_3\tilde{a}_3 \leq \tilde{a}_0, \ldots, \mathrm{P} : a_1, \mathrm{Q} \multimap \mathrm{R} : a_3, \ldots \vdash \ldots, \mathrm{P} * (\mathrm{Q} \multimap \mathrm{R}) : a_1a_3}\, R$$
$$\Pi_1$$

$$\dfrac{\dfrac{}{\tilde{a}_3 \leq \tilde{a}_3, \ldots, \mathrm{S} : \tilde{a}_3 \vdash \ldots, \mathrm{S} : \tilde{a}_3}}{a_1a_3\tilde{a}_3 \leq \tilde{a}_0, a_1a_3\tilde{a}_3 \leq a_1\tilde{a}_1, a_1 \leq a_1, a_3\tilde{a}_3 \leq \tilde{a}_1, a_1\tilde{a}_1 \leq \tilde{a}_0, \epsilon a_0 \leq \tilde{a}_0, \ldots, \mathrm{S} : \tilde{a}_3 \vdash \ldots, \mathrm{S} : \tilde{a}_3}\, R$$
$$\Pi_2$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\Pi_1 \quad \Pi_2}{a_1a_3\tilde{a}_3 \leq \tilde{a}_0, \ldots, \mathrm{P} : a_1, \mathrm{Q} \multimap \mathrm{R} : a_3, \mathrm{S} : \tilde{a}_3 \vdash ((\mathrm{P} * (\mathrm{Q} \multimap \mathrm{R})) * \mathrm{S}) : \tilde{a}_0}\, *_R}{a_1a_3\tilde{a}_3 \leq a_1\tilde{a}_1, \ldots, a_1\tilde{a}_1 \leq \tilde{a}_0, \ldots, \mathrm{P} : a_1, \mathrm{Q} \multimap \mathrm{R} : a_3, \mathrm{S} : \tilde{a}_3 \vdash ((\mathrm{P} * (\mathrm{Q} \multimap \mathrm{R})) * \mathrm{S}) : \tilde{a}_0}\, T}{a_1 \leq a_1, a_3\tilde{a}_3 \leq \tilde{a}_1, \ldots, \mathrm{P} : a_1, \mathrm{Q} \multimap \mathrm{R} : a_3, \mathrm{S} : \tilde{a}_3 \vdash ((\mathrm{P} * (\mathrm{Q} \multimap \mathrm{R})) * \mathrm{S}) : \tilde{a}_0}\, F}{a_3\tilde{a}_3 \leq \tilde{a}_1, a_1\tilde{a}_1 \leq \tilde{a}_0, \epsilon a_0 \leq \tilde{a}_0, \mathrm{P} : a_1, \mathrm{Q} \multimap \mathrm{R} : a_3, \mathrm{S} : \tilde{a}_3 \vdash ((\mathrm{P} * (\mathrm{Q} \multimap \mathrm{R})) * \mathrm{S}) : \tilde{a}_0}\, R}{a_1\tilde{a}_1 \leq \tilde{a}_0, \epsilon a_0 \leq \tilde{a}_0, \mathrm{P} : a_1, ((\mathrm{Q} \multimap \mathrm{R}) * \mathrm{S}) : \tilde{a}_1 \vdash ((\mathrm{P} * (\mathrm{Q} \multimap \mathrm{R})) * \mathrm{S}) : \tilde{a}_0}}{\dfrac{\epsilon a_0 \leq \tilde{a}_0, (\mathrm{P} * ((\mathrm{Q} \multimap \mathrm{R}) * \mathrm{S}) : a_0 \vdash ((\mathrm{P} * (\mathrm{Q} \multimap \mathrm{R})) * \mathrm{S}) : \tilde{a}_0}{\vdash (\mathrm{P} * ((\mathrm{Q} \multimap \mathrm{R}) * \mathrm{S})) \multimap ((\mathrm{P} * (\mathrm{Q} \multimap \mathrm{R})) * \mathrm{S}) : \epsilon}}$$

Figure 7: GMILL-proof of $(\mathrm{P} * ((\mathrm{Q} \multimap \mathrm{R}) * \mathrm{S})) \multimap ((\mathrm{P} * (\mathrm{Q} \multimap \mathrm{R})) * \mathrm{S})$

corresponds to a requirement) occuring in Γ_{r} for the rules to be applicable. The rules of the GMILL calculus are given in Figure 6.

From the proof of Theorem 3, one can derive a translation of LMILL-proofs into GMILL-proofs so that for any LMILL-proof the corresponding GMILL-proof applies the same rules in the same order. Therefore, since LMILL does not allow contraction, GMILL has no need for it too. Moreover, conditions $(C1)$ to $(C5)$ given in Definition 12 imply that whenever $*_R$ and $\multimap_L$ need to be applied in GMILL, Γ_{r} contains enough assertions to make the rule applicable. Therefore we have the following results.

Theorem 4. *If a formula* A *has a proof in LMILL then it is has a proof in GMILL that follows the same rule application strategy.*

Theorem 5. *A formula* A *is provable in LMILL iff it is provable in GMILL.*

Figure 5 illustrates how GMILL works by giving an example of a derivation in the GMILL

calculus for the formula $(\mathrm{P} * ((\mathrm{Q} \mathrel{-\!\!*} \mathrm{R}) * \mathrm{S})) \mathrel{-\!\!*} ((\mathrm{P} * (\mathrm{Q} \mathrel{-\!\!*} \mathrm{R})) * \mathrm{S})$, which is exactly the one prescribed by the reduction ordering we computed in the running example of Section 3.4.

6 Future Work

From this connection-based characterization of validity in MILL we will consider different perspectives. First we aim at defining a connection method for MILL from such a characterization that mainly corresponds to the definition and implementation of an algorithm for solving our constraints. A complementary question consist in studying how our results can be adapted or refined to deal with other fragments of Intuitionistic Linear Logic and mainly first-order ones including quantifications.

The question of reconstruction of proofs in the MILL sequent calculus from our connection calculus for MILL with labels and constraints has also to be explored w.r.t. existing techniques [11]. Moreover, taking into account the relationships in MLL between some connection-based characterizations and proof-nets [1], we aim at studying similar relationships for MILL and propose new proof methods based on proof-net construction.

References

[1] D. Galmiche. Connection Methods in Linear Logic and Proof nets Construction. *Theoretical Computer Science*, 232(1-2):231–272, 2000.

[2] D. Galmiche and D. Méry. Connection-based proof search in propositional BI logic. In *18th Int. Conference on Automated Deduction, CADE-18, LNAI 2392*, pages 111–128, 2002. Copenhagen, Danemark.

[3] D. Galmiche and D. Méry. A Connection-based Characterization of Bi-intuitionistic Validity. In *23rd Int. Conference on Automated Deduction, CADE-23, LNAI 6803*, pages 253–267, Wroclaw Poland, July 2011.

[4] J.Y. Girard. Linear logic. *Theoretical Computer Science*, 50(1):1–102, 1987.

[5] C. Kreitz and H. Mantel. A matrix characterization for multiplicative exponential linear logic. *Journal of Automated Reasoning*, 32(2):121–166, 2004.

[6] C. Kreitz and J. Otten. Connection-based theorem proving in classical and non-classical logics. *Journal of Universal Computer Science*, 5(3):88–112, 1999.

[7] P.W. O'Hearn and D. Pym. The Logic of Bunched Implications. *Bulletin of Symbolic Logic*, 5(2):215–244, 1999.

[8] J. Otten. Mleancop: A connection prover for first-order modal logic. In *Int. Joint Conference on Automated Reasoning, IJCAR 2014, LNAI 8562*, pages 269–276, Vienna, Austria, 2014.

[9] J. Otten and C. Kreitz. A connection based proof method for intuitionistic logic. In *4th Workshop on Theorem Proving with Analytic Tableaux and Related Methods, LNAI 918*, pages 122–137, St Goar am Rhein, Germany, 1995. Springer Verlag.

[10] J. Otten and C. Kreitz. T-string-unification: unifying prefixes in non classical proof methods. In *5th Int. Workshop TABLEAUX'96, LNAI 1071*, pages 244–260, Terrasini, Palermo, Italy, 1996. Springer Verlag.

[11] S. Schmitt and C. Kreitz. Converting non-classical matrix proofs into sequent-style systems. In *13th Int. Conference on Automated Deduction, LNAI 1104*, pages 418–432, New Brunswick, NJ, USA, 1996.

[12] A. Urquhart. Semantics for Relevant Logic. *Journal of Symbolic Logic*, 37:159–169, 1972.

[13] L.A. Wallen. *Automated Proof search in Non-Classical Logics.* MIT Press, 1990.

Labelled Calculi for Quantified Modal Logics with Non-rigid and Non-denoting Terms

Eugenio Orlandelli and Giovanna Corsi

University of Bologna, Bologna, Italy
{eugenio.orlandelli,giovanna.corsi}@unibo.it

Abstract

We introduce labelled sequent calculi for quantified modal logics with non-rigid and and non-denoting terms. We prove that these calculi have the good structural properties of **G3**-style calculi. In particular, all rules are height-preserving invertible, weakening and contraction are height-preserving admissible and cut is admissible. Finally, we show that each calculus gives a proof-theoretic characterization of validity in the corresponding class of models.

1 Introduction

The proof-theoretic study of propositional modal logics has been a very active field of research in the last few decades thanks to the introduction of generalizations of Gentzen-style sequent calculi such as display calculi [1], hypersequents [1], and labelled sequent calculi [10]. Nevertheless, the proof-theoretic study of quantified modal logics (QMLs) has remained rather underdeveloped. One notable exception is [10, Chap. 12.1], where labelled sequent calculi for QMLs are introduced. More specifically, the labelled calculi for the propositional modal logics in the cube of normal modalities – i.e. the minimal normal modal logic **K** and its extensions with axioms $D, T, 4, 5$ – are extended with quantifiers based on varying, increasing, decreasing and constant domains. One interesting aspect of QMLs that has not been considered in [10] is the study of logics based on a language containing the identity predicate and non-rigid as well as non-denoting terms, see [6, 7]. We introduce labelled calculi for these logics and we study their structural properties.

As it is convincingly argued in [6, 7], the need for non-rigid and non-denoting terms originates from problems already touched upon in the classical works of Frege [8] and Russell [12]. First, as Frege noticed, even if 'Hesperus' and 'Phosphorus' are both names of Venus and the ancient knew that objects are self-identical, the Babylonians did not know that Hesperus is Phosphorus. Despite this, in quantified epistemic logics based on rigid terms, we can prove that the Babylonians knew it. Moreover, Russell showed that the sentence 'The present king of France is not bald' is ambiguous since it might either mean that the sentence 'the present king of France is bald' is false, or that the present king of France is such that he is non-bald. Given that the expression 'The present king of France' does not actually denote anyone, the first reading is, in fact, true and the second false. If we exclude non-denoting terms, we cannot account for these two readings of that sentence (unless we explain away the term expressing the definite description 'the present king of France').

As the two examples above show, it is interesting to consider QMLs with non-rigid and/or non-denoting terms, but their addition to the language of QMLs is not trivial [6, 7]. The problem, roughly, is that if t is a non-rigid (or non-denoting) term, the formal sentence $\Diamond Pt$ becomes ambiguous. When it is evaluated in a possible world w of some model, it might either mean that the object denoted by t in w satisfies the unary predicate P is some world v that is accessible from w, or that there is a world u that is accessible from w and such that the

formal sentence Pt is true therein. For rigid terms these two readings are equivalent. For non-rigid terms neither reading entails the other, and, therefore, we need some scoping mechanism to disambiguate the formula $\Diamond Pt$. The solution proposed in [6, 7] is that of extending the language with the operator of predicate abstraction λ. The two readings of $\Diamond Pt$ can thus be expressed, respectively, by the (semantically independent) formal sentences:

$$\lambda x.\Diamond Px.t \qquad \text{and} \qquad \Diamond(\lambda x.Px.t) \ . \tag{1}$$

We will extend the labelled sequent calculi for QMLs presented in [10] to handle non-rigid and non-denoting terms based on the predicate abstraction operator λ. We will study the structural properties of these extensions, and we will show that, as for the calculi in [10], all the rules of inference are height-preserving invertible, the structural rules of weakening and contraction are height-preserving admissible, and the rule of cut is admissible. Finally, we will prove that each calculus considered characterizes validity in the appropriate class of models.

The paper is organized as follows. Section 2 sketches the labelled calculi for QMLs presented in [10]. In particular, the language and semantics of QMLs without non-rigid and non-denoting terms are introduced in Section 2.1. Labelled calculi for these logics and their main meta-theoretical properties are outlined in Section 2.2. In Section 3, we extend the labelled approach to QMLs with identity and non-rigid and non-denoting terms. We once again start by outlining the syntax and the semantics (Section 3.1). Then, we introduce labelled calculi for these logics (Section 3.2), and we prove: (i) that they have the good structural properties that are distinctive of **G3**-style calculi (Section 3.2.1) and (ii) that they are sound and complete with respect to the appropriate classes of quantified modal models (Section 3.2.2). We conclude in Section 4.

2 Quantified Modal Logics without Individual Constants

In this section, we present QMLs based on a varying domain semantics defined over a signature not containing individual constants nor the identity symbol, and we present labelled calculi for these logics. Apart from some minor adjustment, the semantics is as in [7, Chap. 4.7], and the calculi are as in [10, Chap. 12.1]. This section is needed to make the paper self-contained and it might be skipped by readers already familiar with QMLs and labelled calculi.

2.1 Syntax and Semantics

Let $\mathcal{S}$ be a signature containing, for every $n \in \mathbb{N}$, an at most denumerable set of n-ary predicate letters $P^n_1, P^n_2, \dots$; let VAR be an infinite set of variables $x_1, x_2, \dots$; and let the primitive logical symbols be $\neg, \wedge, \forall, \Box$. The language $\mathscr{L}$ is given by the grammar:

$$A ::= P^n y_1, \dots, y_n \mid \neg A \mid A \wedge A \mid \forall y A \mid \Box A \tag{$\mathcal{L}$}$$

where $P_n \in \mathcal{S}$ and $y, y_1, \dots, y_n \in VAR$. In this paper we use the following metavariables:

- P, Q, R for predicate letters;
- x, y, z for variables;
- p, q, r for atomic formulas;
- A, B, C for formulas.

We follow the standard conventions for parentheses. The formulas $\bot, \top, A \vee B, A \supset B, \exists y A$, and $\diamond A$ are defined as expected. The notions of *free* and *bound occurrences* of a variable in a formula are the usual ones. Given a formula A, we use $A[y/x]$ to denote the formula obtained by replacing each free occurrence of x in A with an occurrence of y, provided that y is free for x in A – i.e., if no one of the new occurrences of y would be bound by a quantifier. The *weight* of a formula is the number of nodes of its generation tree.

A *model* (over the signature $\mathcal{S}$) is a tuple:

$$\mathscr{M} =< \mathcal{W}, \mathcal{R}, \mathcal{D}, \mathcal{V} > \tag{2}$$

where

- $\mathcal{W} \neq \emptyset$ is a nonempty set of (possible) *worlds* (to be denoted by $w, v, u \dots$);
- $\mathcal{R} \subseteq \mathcal{W} \times \mathcal{W}$ is a binary *accessibility relation* between worlds;
- $\mathcal{D} : \mathcal{W} \longrightarrow 2^D$ is a function mapping each world to a possibly empty set of objects D_w (its *domain*), where $D = \bigcup_{w \in \mathcal{W}} D_w$ is nonempty and disjoint from $\mathcal{W}$;
- $\mathcal{V} : \mathcal{S} \times \mathcal{W} \longrightarrow 2^{D^n}$ is an *interpretation* function mapping, at each world w, each n-ary predicate $P(\in \mathcal{S})$ to a subset of D^n.

Given a model $\mathscr{M} =< \mathcal{W}, \mathcal{R}, \mathcal{D}, \mathcal{V} >$, an *assignment* (over $\mathscr{M}$) is a function $\sigma : VAR \longrightarrow D$ mapping each variable x to an element of the union of the domains of the model. Moreover, for $a \in D$, $\sigma^{x \rhd a}$ denotes the assignment behaving like σ save for x that is mapped to the object a.

Definition 1 (Satisfaction)**.** Given a model $\mathscr{M}$, an assignment σ over it, and a world w of that model, we define the notion of *satisfaction* of an $\mathscr{L}$-formula A as follows:

$\sigma \models^{\mathscr{M}}_w Px_1, \dots x_n$	iff	$< \sigma(x_1), \dots, \sigma(x_n) > \in \mathcal{V}(P, w)$
$\sigma \models^{\mathscr{M}}_w \neg B$	iff	$\sigma \not\models^{\mathscr{M}}_w B$
$\sigma \models^{\mathscr{M}}_w B \wedge C$	iff	$\sigma \models^{\mathscr{M}}_w B$ and $\sigma \models^{\mathscr{M}}_w C$
$\sigma \models^{\mathscr{M}}_w \forall x B$	iff	for all $a \in D_w$, $\sigma^{x \rhd a} \models^{\mathscr{M}}_w B$
$\sigma \models^{\mathscr{M}}_w \Box B$	iff	for all $v \in \mathcal{W}$, $w\mathcal{R}v$ implies $\sigma \models^{\mathscr{M}}_v B$

The notions of *truth in a world w of a model* ($\models^{\mathscr{M}}_w A$), *truth in a model* ($\models^{\mathscr{M}} A$), and *validity in a class of models* ($\mathcal{C} \models A$) are defined as usual.

As it is well known, some notable formulas, such as the Barcan Formulas, are valid in classes of models defined by properties of the accessibility relation and/or of the domains. By an $\mathscr{L}$-*logic* **Q.L** we mean the set of all $\mathscr{L}$-formulas that are valid in a class of models defined by some combination of the properties given in Table 1. We use standard names for $\mathscr{L}$-logics – e.g., **Q.K** stands for the set of $\mathscr{L}$-formulas valid in the class of all models, and **Q.S4⊕BF** stands for the set of $\mathscr{L}$-formulas valid in the class of all reflexive and transitive models with decreasing domains. We say that $\mathscr{M}$ *is a model for* **Q.L** whenever $\models^{\mathscr{M}} A$ for all $A \in$ **Q.L**.

2.2 Labelled Sequent Calculi

Labelled sequent calculi for $\mathscr{L}$-logics have been introduced in [10, Chap. 12.1]. These calculi are based on an extension of the modal language in order to internalize the semantics of QMLs as follows. First of all, we introduce a set LAB of fresh variables, called *labels*. Labels will be denoted by $w, v, u, \dots$ and will be used to represent worlds. Then, we extend the set of formulas

Table 1: Modal axioms and corresponding semantic properties

$T := \Box A \supset A$	reflexivity:=$\forall w \in \mathcal{W}(w\mathcal{R}w)$
$D := \Box A \supset \Diamond A$	seriality:=$\forall w \in \mathcal{W}\exists u \in \mathcal{W}(w\mathcal{R}u)$
$4 := \Box A \supset \Box\Box A$	transitivity:=$\forall w, v, u \in \mathcal{W}(w\mathcal{R}v \wedge v\mathcal{R}u \supset w\mathcal{R}u)$
$5 := \Diamond A \supset \Box\Diamond A$	Euclideaness:=$\forall w, v, u \in \mathcal{W}(w\mathcal{R}v \wedge w\mathcal{R}u \supset v\mathcal{R}u)$
$NE := \forall xA \supset \exists xA$	nonempty domains:=$\forall w \in \mathcal{W}\exists a \in D(a \in D_w)$
$UI := \forall xA \supset A[y/x]$	constant domains:=$\forall w \in \mathcal{W}\forall a \in D(a \in D_w)$
$CBF := \Box\forall xA \supset \forall x\Box A$	increasing domains:=$\forall w, v \in \mathcal{W}\forall a \in D(a \in D_w \wedge w\mathcal{R}v \supset a \in D_v)$
$BF := \forall x\Box A \supset \Box\forall xA$	decreasing domains:=$\forall w, v \in \mathcal{W}\forall a \in D(a \in D_v \wedge w\mathcal{R}v \supset a \in D_w)$

by adding atomic formulas of shape $x \in w$ – expressing that (the object assigned to) x is in the domain of quantification of (the world represented by) w – and of shape $w\mathscr{R}v$ – expressing that v is accessible from w. Lastly, we replace each $\mathscr{L}$-formula A with the labelled formulas $w : A$ – expressing that A holds at w. A *labelled sequent* is an expression:

$$\Omega; \Gamma \Rightarrow \Delta$$

where Ω is a multiset of atomic formulas of shape $x \in w$ or $w\mathscr{R}v$, and Γ and Δ are multisets of labelled formulas. Given a formula E of this extended language, $E[w/v]$ is the formula obtained by substituting each occurrence of v in E with an occurrence of w. Substitution of variables is extended to formulas of the extended language as expected, and both kinds of substitution are extended to sequents by applying them componentwise.

The rules of the calculus **G3Q.K**, for the minimal $\mathscr{L}$-logic **Q.K**, are given in Table 2. For each logic **Q.L** extending **Q.K**, the calculus **G3Q.L** is obtained by extending **G3Q.K** with the non-logical rules of Table 3 that express proof-theoretically the semantic properties that define **Q.L** (cf. Table 1). Whenever a calculus contains rule $Eucl$, it contains also all its contracted instances $Eucl^c$. Observe that CBF (BF) is not derivable in calculi where rule $Incr$ ($Decr$) is not primitive nor admissible (given Proposition 2.8, this can be checked semantically).

A **G3Q.L**-*derivation* of a sequent $\Omega; \Gamma \Rightarrow \Delta$ is a tree of sequents, whose leaves are initial sequents, whose root is $\Omega; \Gamma \Rightarrow \Delta$, and which grows according to the rules of **G3Q.L**. As usual, we consider only derivations of *pure sequents* – i.e., sequents where no variable has both free and bound occurrences. The *height* of a **G3Q.L**-derivation is the number of nodes of its longest branch. We say that $\Omega; \Gamma \Rightarrow \Delta$ is **G3Q.L**-derivable (with height n), and we write **G3Q.L** $\vdash^{(n)} \Omega; \Gamma \Rightarrow \Delta$, if there is a **G3Q.L**-derivation (of height at most n) of $\Omega; \Gamma \Rightarrow \Delta$. A rule is said to be *(height-preserving) admissible* in **G3Q.L**, if, whenever its premisses are **G3Q.L**-derivable (with height at most n), also its conclusion is **G3Q.L**-derivable (with height at most n). In each rule depicted in Tables 2 and 3, Ω, Γ and Δ are called *contexts*, the formulas occurring in the conclusion are called *principal*, and the formulas occurring in the premisses only are called *active*.

The following proposition presents the main meta-theoretical properties of **G3Q.L**. The proofs can be found in [10, Chap. 12.1].

Proposition 2 (Properties of **G3Q.L**).

1. Sequents of shape $\Omega; w : A, \Gamma \Rightarrow \Delta, w : A$ (with A non-atomic) are **G3Q.L**-derivable.

Table 2: Rules of **G3Q.K**

initial sequents:	$\Omega; w : p, \Gamma \Rightarrow \Delta, w : p$, with p atomic
logical rules:	
$\dfrac{\Omega; \Gamma \Rightarrow \Delta, w : A}{\Omega; w : \neg A, \Gamma \Rightarrow \Delta}\ L\neg$	$\dfrac{\Omega; w : A, \Gamma \Rightarrow \Delta}{\Omega; \Gamma \Rightarrow \Delta, w : \neg A}\ R\neg$
$\dfrac{\Omega; w : A, w : B, \Gamma \Rightarrow \Delta}{\Omega; w : A \wedge B, \Gamma \Rightarrow \Delta}\ L\wedge$	$\dfrac{\Omega; \Gamma \Rightarrow \Delta, w : A \quad \Omega; \Gamma \Rightarrow \Delta, w : B}{\Omega; \Gamma \Rightarrow \Delta, w : A \wedge B}\ R\wedge$
$\dfrac{y \in w, \Omega; w : A[y/x], w : \forall x A, \Gamma \Rightarrow \Delta}{y \in w, \Omega; w : \forall x A, \Gamma \Rightarrow \Delta}\ L\forall$	$\dfrac{z \in w, \Omega; \Gamma \Rightarrow \Delta, w : A[z/x]}{\Omega; \Gamma \Rightarrow \Delta, w : \forall x A}\ R\forall, z \text{ fresh}$
$\dfrac{w\mathscr{R}v, \Omega; v : A, w : \Box A, \Gamma \Rightarrow \Delta}{w\mathscr{R}v, \Omega; w : \Box A, \Gamma \Rightarrow \Delta}\ L\Box$	$\dfrac{w\mathscr{R}u, \Omega; \Gamma \Rightarrow \Delta, u : A}{\Omega; \Gamma \Rightarrow \Delta, w : \Box A}\ R\Box, u \text{ fresh}$

Table 3: Non-logical rules

$\dfrac{w\mathscr{R}w, \Omega; \Gamma \Rightarrow \Delta}{\Omega; \Gamma \Rightarrow \Delta}\ Ref_{\mathcal{W}}$	$\dfrac{v\mathscr{R}u, w\mathscr{R}v, w\mathscr{R}u, \Omega; \Gamma \Rightarrow \Delta}{w\mathscr{R}v, w\mathscr{R}u, \Omega; \Gamma \Rightarrow \Delta,}\ Eucl$	$\dfrac{v\mathscr{R}v, w\mathscr{R}v, \Omega; \Gamma \Rightarrow \Delta}{w\mathscr{R}v, \Omega; \Gamma \Rightarrow \Delta,}\ Eucl^c$
$\dfrac{w\mathscr{R}u, \Omega; \Gamma \Rightarrow \Delta}{\Omega; \Gamma \Rightarrow \Delta,}\ Ser, u \text{ fresh}$	$\dfrac{w\mathscr{R}u, w\mathscr{R}v, v\mathscr{R}u, \Omega; \Gamma \Rightarrow \Delta}{w\mathscr{R}v, v\mathscr{R}u, \Omega; \Gamma \Rightarrow \Delta}\ Trans$	
$\dfrac{z \in w, \Omega; \Gamma \Rightarrow \Delta}{\Omega; \Gamma \Rightarrow \Delta}\ NonEm, z \text{ fresh}$	$\dfrac{x \in v, x \in w, w\mathscr{R}v, \Omega; \Gamma \Rightarrow \Delta}{x \in w, w\mathscr{R}v, \Omega; \Gamma \Rightarrow \Delta}\ Incr$	
$\dfrac{x \in w, \Omega; \Gamma \Rightarrow \Delta}{\Omega; \Gamma \Rightarrow \Delta}\ Cons$	$\dfrac{x \in w, x \in v, w\mathscr{R}v, \Omega; \Gamma \Rightarrow \Delta}{x \in v, w\mathscr{R}v, \Omega; \Gamma \Rightarrow \Delta}\ Decr$	

2. The rule of α-conversion is height-preserving admissible: if **G3Q.L** $\vdash^n \Omega; \Gamma \Rightarrow \Delta$, then **G3Q.L** $\vdash^n \Omega; \Gamma' \Rightarrow \Delta'$, where Γ' (Δ') is obtained from Γ (Δ) by renaming bound variables.

3. The following rules of substitution are height-preserving admissible in **G3Q.L**:

$$\frac{\Omega; \Gamma \Rightarrow \Delta}{\Omega[y/x]; \Gamma[y/x] \Rightarrow \Delta[y/x]}\ [y/x] \qquad \frac{\Omega; \Gamma \Rightarrow \Delta}{\Omega[w/v]; \Gamma[w/v] \Rightarrow \Delta[w/v]}\ [w/v]$$

where y is free for x in each formula occurring in Γ, Δ for rule $[y/x]$.

4. The following rules of weakening are height-preserving admissible in **G3Q.L**:

$$\frac{\Omega; \Gamma \Rightarrow \Delta}{\Omega', \Omega; \Gamma \Rightarrow \Delta}\ LW_{\Omega} \qquad \frac{\Omega; \Gamma \Rightarrow \Delta}{\Omega; \Gamma', \Gamma \Rightarrow \Delta}\ LW \qquad \frac{\Omega; \Gamma \Rightarrow \Delta}{\Omega; \Gamma \Rightarrow \Delta, \Delta'}\ RW$$

5. Each rule of **G3Q.L** is height-preserving invertible.

6. The following rules of contraction are height-preserving admissible in **G3Q.L**:

$$\frac{\Omega', \Omega', \Omega; \Gamma \Rightarrow \Delta}{\Omega', \Omega; \Gamma \Rightarrow \Delta}\ LC_{\Omega} \qquad \frac{\Omega; \Gamma', \Gamma', \Gamma \Rightarrow \Delta}{\Omega; \Gamma', \Gamma \Rightarrow \Delta}\ LC \qquad \frac{\Omega; \Gamma \Rightarrow \Delta, \Delta', \Delta'}{\Omega; \Gamma \Rightarrow \Delta, \Delta'}\ RC$$

7. The following rule of Cut is admissible in **G3Q.L**:

$$\frac{\Omega; \Gamma \Rightarrow \Delta, w : A \quad \Omega'; w : A, \Gamma' \Rightarrow \Delta'}{\Omega, \Omega'; \Gamma', \Gamma \Rightarrow \Delta, \Delta'} \; Cut$$

8. **G3Q.L** is sound and complete with respect to the class of all models for **Q.L**.

3 Quantified Modal Logics with Individual Constants

Now we move to QMLs based on a language containing also non-rigid and non-denoting individual constants and the identity predicate. Given that constants will have a world-dependent interpretation, we will introduce the operator λ as a scoping mechanism. This allows us to distinguish between the formula $\lambda x.\Box Px.c$ (that is interpreted by first determining the object o denoted by c in w, and then by moving to worlds accessible from w to see whether o satisfies P therein) and the formula $\Box(\lambda x.Px.c)$ (interpreted by first moving to each world v accessible from w, and then by determining the object denoted by c in each v and checking if it satisfies P). The semantics is analogous to the varying domain semantics considered in [7, Chap. 11] and in [6]. Lastly, we introduce labelled sequent calculi for logics based on this semantics and we study their meta-theoretical properties.

3.1 Syntax and Semantics

The language of $\mathcal{L}^\lambda$ is obtained by extending the signature $\mathcal{S}$ with an at most denumerable set CON of individual constants $c_1, c_2, \ldots$; functions of higher arity are omitted for simplicity. Let us call $\mathcal{S}^\lambda$ the extended signature. Moreover, we extend the set of logical symbols with the operator λ of predicate abstraction and with the logical predicate $=$. The term-forming operator $\imath$ [7, Chap. 12] is omitted for simplicity. The set TER of *terms* is the union of VAR and CON. The set of $\mathcal{L}^\lambda$-formulas is generated by the grammar:

$$A ::= P^n y_1, \ldots, y_n \mid y_1 = y_2 \mid \neg A \mid A \wedge A \mid \forall y A \mid \Box A \mid \lambda y.A.t \qquad (\mathcal{L}^\lambda)$$

where $y, y_1, \ldots, y_n \in VAR$, $P^n \in \mathcal{S}^\lambda$, and $t \in TER$. We will use for $\mathcal{L}^\lambda$-formulas the same definitions introduced in Sect 2.1 for $\mathcal{L}$-formulas; the only novelty being that, in $\lambda x.A.y$, the occurrences of x are bound by λ and the displayed instance of y is free. Note that individual constants cannot occur in atomic $\mathcal{L}^\lambda$-formulas: they can only be applied to a formula *via* λ.

A *model* (over the signature $\mathcal{S}^\lambda$) is a tuple:

$$\mathcal{M} = < \mathcal{W}, \mathcal{R}, \mathcal{D}, \mathcal{V} > \qquad (3)$$

where $\mathcal{W}$, $\mathcal{R}$ and $\mathcal{D}$ are defined as for $\mathscr{M}$, see 2, and $\mathcal{V}$ is defined as:

- $\mathcal{V} : \mathcal{S}^\lambda \times \mathcal{W} \longrightarrow 2^{D^n}$ is a partial *interpretation* function such that:
  1. for each predicate P^n and each $w \in \mathcal{W}$, $\mathcal{V}(P^n, w) \subseteq D^n$;
  2. for each individual constants c and for some, possibly not all, $w \in \mathcal{W}$, $\mathcal{V}(c, w) \in D$.

Assignments are defined as before as (total) functions $\sigma : VAR \longrightarrow D$. With an abuse of notation, we use $\sigma_w(t)$ for the object denoted by the term t in $\mathcal{M}$ under σ – i.e., $\sigma_w(t)$ stands for $\sigma(t)$ if t is a variable and for $\mathcal{V}(t, w)$ if t is an individual constant. Notice that, if t is a constant, the fact that $\mathcal{V}(t, w) = o$ does not entail that $\mathcal{V}(t, v) = o$ nor that $\mathcal{V}(t, v)$ is defined.

Table 4: Additional modal axioms and corresponding properties

$RG := \lambda x.\Box A.t \supset \Box(\lambda x.A.t)$	rigidity:$=\forall w, v \in \mathcal{W}, \forall t \in CON(w\mathcal{R}v \& \sigma_w(t) \in D \supset \sigma_v(t) = \sigma_w(t))$
$TT := \lambda x.x = x.t$	totality:$=\forall w \in \mathcal{W} \forall t \in CON(\sigma_w(t) \in D)$

Definition 3 (Satisfaction)**.** Satisfaction of an $\mathcal{L}^\lambda$-formula A at a world w of a model $\mathcal{M}$ under an assignment σ is defined as in Definition 1 with the addition of the following clauses:

$$\sigma \models^{\mathcal{M}}_w x = y \quad \text{iff} \quad \sigma(x) = \sigma(y)$$

$$\sigma \models^{\mathcal{M}}_w \lambda x.B.t \quad \text{iff} \quad \sigma_w(t) \text{ is defined and } \sigma^{x \triangleright \sigma_w(t)} \models^{\mathcal{M}}_w B$$

The notions of truth and of validity are defined as in Sect. 2.1. An $\mathcal{L}^\lambda$-logic $\mathbf{Q\lambda.L}$ is defined as the set of all formulas valid in some class of models $\mathcal{M}$ that is obtained by some combination of the semantic properties in Tables 1 and 4.

We end the presentation of the semantics by emphasizing the effects of non-denoting and non-rigid constants. First, since c may not denote at a world w of a model $\mathcal{M}$, it might happen that c satisfies no predicate at w, not even self-identity. Second, since constants might denote different objects at different worlds, we cannot use the information that the constants h and p denote the same object in w (say, h and p stand, respectively, for the names 'Hesperus' and 'Phosphorus', and both denote Venus in w) to conclude that they denote the same object in worlds accessible from w. The sentence $\lambda x.\lambda y.x = y.p.h \supset \Box(\lambda x.\lambda y.x = y.p.h)$ is not valid in models with non-rigid designators. This might seem incompatible with the fact that an unrestricted rule of replacement holds for identity. But identity atoms express a relation between variables and not between constants, which may only be applied to an identity *via* the operator λ. Thus, the rule of replacement allows only to substitute variables that denote the same object. The operator λ might be looked at as a device to block the permutation of substitutions and modalities for non-rigid constants, as is witness by axiom RG in Table 4. Finally, note that for $y \in VAR$, the formulas $\lambda x.A.y$ and $A[y/x]$ are semantically equivalent. This holds because variables are rigid and always denoting terms.

3.2 Labelled Sequent Calculi

In order to introduce labelled sequent calculi for QMLs with non-rigid and non-denoting individual constants, we extend the language of labelled calculi with ternary atomic expressions of shape $x \overset{w}{\approx} t$, which will be used to express that the variable x picks the object denoted in w by the term t. From now on, a sequent $\Omega; \Gamma \Rightarrow \Delta$ is an expression where Ω is a multiset of atomic formulas of shape $x \overset{w}{\approx} t$, $x \in w$ or $w\mathscr{R}v$; and Γ and Δ are multisets of labelled $\mathcal{L}^\lambda$-formulas.

The rules of the calculus $\mathbf{G3Q\lambda.K}$ are the rules of $\mathbf{G3Q.K}$, see Table 2, plus the rules for identity and the rules for λ of Table 5. Observe that the rules for identity contain the labelled version of the universal rules first introduced in [9]. When $w : y = x$ holds, by *Repl* we can replace x with y in any atomic formula that, so to say, talks about w. The universal rule $RigVar$ implies that if x and y denote the same object in some world, they do so in each world. Thus, variables behave as rigid designators and labels could be omitted from identities. We choose to keep them in order to have a more uniform notation.

The satisfaction clause for $\lambda x.A.t$ in a world w is similar to that for $\exists x A$, the only difference being that A has to be satisfied not by some arbitrary object of D_w, but by the one and only

Table 5: Additional rules for **G3λ.L**

rules for identity:

$$\frac{\Omega; w : x = x, \Gamma \Rightarrow \Delta}{\Omega; \Gamma \Rightarrow \Delta}\ {\scriptstyle Ref_{=}} \qquad \frac{\Omega; v : y = z, w : y = z, \Gamma \Rightarrow \Delta}{\Omega; w : y = z, \Gamma \Rightarrow \Delta}\ {\scriptstyle RigVar}$$

$$\frac{\Omega, E[z/x], E[y/x], w : y = z, \Gamma \Rightarrow \Delta}{\Omega, E[y/x], w : y = z, \Gamma \Rightarrow \Delta,}\ {\scriptstyle Repl} \qquad E \text{ is either } x_i \overset{w}{\approx} t \text{ or } x_i \in w \text{ or } w : p$$

rules for λ:

$$\frac{z \overset{w}{\approx} t, \Omega; w : A[z/x], \Gamma \Rightarrow \Delta}{\Omega; w : \lambda x.A.t, \Gamma \Rightarrow \Delta}\ {\scriptstyle L\lambda,\, z \text{ fresh}} \qquad \frac{y \overset{w}{\approx} t, \Omega; \Gamma \Rightarrow \Delta, w : \lambda x.A.t, w : A[y/x]}{y \overset{w}{\approx} t, \Omega; \Gamma \Rightarrow \Delta, w : \lambda x.A.t}\ {\scriptstyle R\lambda}$$

$$\frac{x \overset{w}{\approx} x, \Omega; \Gamma \Rightarrow \Delta}{\Omega; \Gamma \Rightarrow \Delta}\ {\scriptstyle DenVar} \qquad \frac{y \overset{w}{\approx} x, \Omega; w : x = y, \Gamma \Rightarrow \Delta}{y \overset{w}{\approx} x, \Omega; \Gamma \Rightarrow \Delta}\ {\scriptstyle DenId}$$

non-logical rules:

$$\frac{z \overset{w}{\approx} t, \Omega; \Gamma \Rightarrow \Delta}{\Omega; \Gamma \Rightarrow \Delta}\ {\scriptstyle Tot,\, z \text{ fresh}} \qquad \frac{x \overset{v}{\approx} t, x \overset{w}{\approx} t, w\mathscr{R}v, \Omega; \Gamma \Rightarrow \Delta}{x \overset{w}{\approx} t, w\mathscr{R}v, \Omega; \Gamma \Rightarrow \Delta}\ {\scriptstyle Rig}$$

object of D that is denoted by t in that world of that model. Therefore the rules for λ are like the ones for $\exists$, see [10], save that they are restricted by atomic formulas of shape $y \overset{w}{\approx} t$ instead of $y \in w$. The universal rule $DenId$ ensures that if y in w picks the object denoted by x, then x and y denote the same object. Finally, the universal rule $DenVar$ ensures that variables denote at every world.

The calculus **G3Qλ.L** is obtained by extending **G3Qλ.K** with the non-logical rules from Tables 3 and 5 that express the semantic properties defining **Qλ.L**.

Example 4 (Derivation of axioms in Table 4)**.** We show here that RG is derivable in calculi containing rule Rig (where rule ${\scriptstyle R\supset}$ is admissible) and that TT is derivable in calculi containing rule Tot. These derivations also show that these formulas are not derivable in calculi without these rules when t is an individual constant. If t is a variable, TT is always derivable thanks to $RigVar$, and RG is always derivable thanks to $DenVar$, $DenId$, $RigVar$, and $Repl$.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{y \overset{v}{\approx} t, w\mathscr{R}v, y \overset{w}{\approx} t; v : A[y/x], w : \Box A[y/x] \Rightarrow v : \lambda x.A.t, v : A[y/x]}\ {\scriptstyle Lem.5}}{y \overset{v}{\approx} t, w\mathscr{R}v, y \overset{w}{\approx} t; v : A[y/x], w : \Box A[y/x] \Rightarrow v : \lambda x.A.t}\ {\scriptstyle R\lambda}}{w\mathscr{R}v, y \overset{w}{\approx} t; v : A[y/x], w : \Box A[y/x] \Rightarrow v : \lambda x.A.t}\ {\scriptstyle Rig}}{w\mathscr{R}v, y \overset{w}{\approx} t; w : \Box A[y/x] \Rightarrow v : \lambda x.A.t}\ {\scriptstyle L\Box}}{y \overset{w}{\approx} t; w : \Box A[y/x] \Rightarrow w : \Box(\lambda x.A.t)}\ {\scriptstyle R\Box}}{w : \lambda x.\Box A.t \Rightarrow w : \Box(\lambda x.A.t)}\ {\scriptstyle L\lambda}}{\Rightarrow w : \lambda x.\Box A.t \supset \Box(\lambda x.A.t)}\ {\scriptstyle R\supset}$$

$$\dfrac{\dfrac{\dfrac{\overline{y \overset{w}{\approx} t; w : y = y \Rightarrow w : \lambda x.x = x.t, w : y = y}}{y \overset{w}{\approx} t \Rightarrow w : \lambda x.x = x.t, w : y = y}\ {\scriptstyle Ref_=}}{y \overset{w}{\approx} t \Rightarrow w : \lambda x.x = x.t}\ {\scriptstyle R\lambda}}{\Rightarrow w : \lambda x.x = x.t}\ {\scriptstyle Tot}$$

3.2.1 Structural Properties

Lemma 5 (Initial sequents)**.** *Sequents of shape $\Omega; w : A, \Gamma \Rightarrow \Delta, w : A$ (with A arbitrary $\mathcal{L}^\lambda$-formula) are* **G3Qλ.L**-*derivable.*

Proof. By induction on the weight of A; for the inductive steps it is enough to apply, root first, the rules for the principal operator of A and then the inductive hypothesis (IH). □

Lemma 6 (α-conversion)**.** **G3Qλ.L** $\vdash^n \Omega; \Gamma \Rightarrow \Delta$ *entails* **G3Qλ.L** $\vdash^n \Omega; \Gamma' \Rightarrow \Delta'$*, where Γ' (Δ') is obtained from Γ (Δ) by renaming some bound variable (without capturing variables).*

Proof. The proof is by induction on the height of the **G3Qλ.L**-derivation $\mathcal{D}$ of $\Omega; \Gamma \Rightarrow \Delta$. To illustrate, suppose we know that **G3Qλ.L** $\vdash^n \Omega; w : \lambda x.A.t, \Gamma \Rightarrow \Delta$, and we want to show that **G3Qλ.L** $\vdash^n \Omega; w : \lambda y.A[y/x].t, \Gamma \Rightarrow \Delta$. If $w : \lambda x.A.t$ is not principal in the last step of $\mathcal{D}$, the proof is straightforward. Else, we transform

$$\dfrac{z \overset{w}{\approx} t, \Omega; w : A[z/x], \Gamma \Rightarrow \Delta}{\Omega; w : \lambda x.A.t, \Gamma \Rightarrow \Delta}\ {\scriptstyle L\lambda} \quad \text{into} \quad \dfrac{\dfrac{z \overset{w}{\approx} t, \Omega; w : A[z/x], \Gamma \Rightarrow \Delta}{z \overset{w}{\approx} t, \Omega; w : (A[y/x])[z/y], \Gamma \Rightarrow \Delta)}\ \star}{\Omega; w : \lambda y.A[y/x].t, \Gamma \Rightarrow \Delta}\ {\scriptstyle L\lambda}$$

where the step $\star$ is height-preserving admissible since, having assumed that the renaming cannot capture variables, $w : (A[y/x])[z/y]$ is just a cumbersome notation for $w : A[z/x]$. □

Lemma 7 (Substitutions)**.** *The following rules are height-preserving admissible in* **G3Qλ.L***:*

$$\dfrac{\Omega; \Gamma \Rightarrow \Delta}{\Omega[y/x]; \Gamma[y/x] \Rightarrow \Delta[y/x]}\ {\scriptstyle [y/x]} \qquad \dfrac{\Omega; \Gamma \Rightarrow \Delta}{\Omega[w/v]; \Gamma[w/v] \Rightarrow \Delta[w/v]}\ {\scriptstyle [w/v]}$$

where y is free for x in each formula occurring in Γ, Δ for rule $[y/x]$.

Proof. Both proofs are by induction on the height of the derivation $\mathcal{D}$ of the premiss $\Omega; \Gamma \Rightarrow \Delta$. The base cases and the inductive steps where the last rule is not a rule from Table 5 are proved in [10, Lemma 12.4].

Of the new cases, the only nontrivial one is the one for rule $[y/x]$ where the last step is by $L\lambda$ and the substitution $[y/x]$ clashes with its variable condition. E.g., the last step of $\mathcal{D}$ is

$$\dfrac{y \overset{w}{\approx} t, \Omega; w : A[y/x], \Gamma' \Rightarrow \Delta}{\Omega; w : \lambda x.A.t, \Gamma' \Rightarrow \Delta}\ {\scriptstyle L\lambda}$$

with x occurring free in Ω, Γ', Δ. We apply IH twice to the premiss of the last step of $\mathcal{D}$, the first time to replace y with z, for some fresh variable z, and the second time to replace x with y. We finish by applying rule $L\lambda$. We have thus transformed $\mathcal{D}$ into $\mathcal{D}[y/x]$:

$$\cfrac{\cfrac{\cfrac{y \overset{w}{\approx} t, \Omega; w : A[y/x], \Gamma' \Rightarrow \Delta}{z \overset{w}{\approx} t, \Omega; w : A[z/x], \Gamma' \Rightarrow \Delta}\ IH}{z \overset{w}{\approx} (t[y/x]), \Omega[y/x]; w : A[z/x], \Gamma'[y/x] \Rightarrow \Delta[y/x]}\ IH}{\Omega[y/x]; w : \lambda x.A.(t[y/x]), \Gamma'[y/x] \Rightarrow \Delta[y/x]}\ L\lambda$$

which has the same height as $\mathcal{D}$ because the steps by IH are height-preserving admissible. □

Theorem 8 (Weakening)**.** *The following rules are height-preserving admissible in* **G3Q**λ**.L***:*

$$\cfrac{\Omega; \Gamma \Rightarrow \Delta}{\Omega', \Omega; \Gamma \Rightarrow \Delta}\ LW_\Omega \qquad \cfrac{\Omega; \Gamma \Rightarrow \Delta}{\Omega; \Gamma', \Gamma \Rightarrow \Delta}\ LW \qquad \cfrac{\Omega; \Gamma \Rightarrow \Delta}{\Omega; \Gamma \Rightarrow \Delta, \Delta'}\ RW$$

Proof. The proofs are by induction on the height of the derivation $\mathcal{D}$ of the premiss $\Omega; \Gamma \Rightarrow \Delta$. The base cases and the inductive cases where the last step of $\mathcal{D}$ is not by a rule from Table 5 are proved in [10, Thm. 12.5]. The proofs of the inductive cases when the last step of $\mathcal{D}$ is by $L\lambda$ or by $R\lambda$ are analogous to the ones in [10, Thm. 12.5] with last step of $\mathcal{D}$ by rules $L\exists$ and $R\exists$, respectively. The remaining cases are similar to the other ones with last step by a geometric rule and can, therefore, be omitted. □

Lemma 9 (Invertibility)**.** *Each rule of* **G3Q**λ**.L** *is height-preserving invertible.*

Proof. We prove only the case of $L\lambda$ ($R\lambda$ is 'Kleene'-invertible thanks to the repetition of the principal formulas in the premiss). The proof is by induction on the height of the derivation $\mathcal{D}$ of $\Omega; w : \lambda x.A.t, \Gamma \Rightarrow \Delta$. If the height of $\mathcal{D}$ is 0 or if $w : \lambda x.A.t$ is principal in the last step of $\mathcal{D}$, the lemma holds trivially. In the other inductive cases we obtain a derivation of $y \overset{w}{\approx} t, \Omega; w; A[y/x], \Gamma \Rightarrow \Delta$ (where y is any variable) by first applying to the premiss(es) of the last step of $\mathcal{D}$ an height-preserving admissible instance of substitution to avoid problems with variable conditions on the last step of $\mathcal{D}$, if this is needed. Then, we apply IH to the sequent(s) just obtained and we finish by applying the last rule applied in $\mathcal{D}$. □

Theorem 10 (Contraction)**.** *The following rules are height-preserving admissible in* **G3Q**λ**.L***:*

$$\cfrac{\Omega', \Omega', \Omega; \Gamma \Rightarrow \Delta}{\Omega', \Omega; \Gamma \Rightarrow \Delta}\ LC_\Omega \qquad \cfrac{\Omega; \Gamma', \Gamma', \Gamma \Rightarrow \Delta}{\Omega; \Gamma', \Gamma \Rightarrow \Delta}\ LC \qquad \cfrac{\Omega; \Gamma \Rightarrow \Delta, \Delta', \Delta'}{\Omega; \Gamma \Rightarrow \Delta, \Delta'}\ RC$$

Proof. The proof is handled by a simultaneous induction on the height of the derivations of the premisses of LC_w, LC and RC. Without loss of generality, we assume the multiset we are contracting is made of only one formula E.

The base cases hold, and the inductive cases depend on whether zero, one, or two instances of E are principal in the last step R of the derivation $\mathcal{D}$ of the premiss. If zero instances are principal in R, we apply IH to the premiss(es) of R and then R, and we are done.

If one instance is principal and R is by one of $L\neg, R\neg, L\wedge, R\wedge, R\forall, R\Box$ and $L\lambda$, we proceed by first applying invertibility to that rule, then we apply IH as many times as needed, and we conclude by applying an instance of that rule. Else, R is by a rule with repetition of the principal formula(s) in the premiss and we don't even need invertibility.

If two instances are principal, R is one of *Euclid*, *Trans*, and *Repl*. The case of *Euclid* is taken care by the presence of its contracted instances $Euclid^c$. For *Trans*, we have three occurences of $w\mathscr{R}w$ in the premiss of this rule instances: two principal and one active. We apply IH twice and we are done. For *Repl*, the active formula of the last rule instance must be of shape $w : x = x$, and, after having applied IH, we can get rid of it by applying $Ref_=$. □

Theorem 11 (Cut)**.** *The following rule of Cut is admissible in* **G3Q.L***:*

$$\dfrac{\Omega; \Gamma \Rightarrow \Delta, w : A \quad \Omega'; w : A, \Gamma' \Rightarrow \Delta'}{\Omega, \Omega'; \Gamma', \Gamma \Rightarrow \Delta, \Delta'}\; {\scriptstyle Cut}$$

Proof. The proof, which extends that of [10, Thm. 12.9], considers an uppermost instance of Cut which is handled by a principal induction on the weight of the cut-formula $w : A$ with a sub-induction on the sum of the heights of the derivations $\mathcal{D}_1$ and $\mathcal{D}_2$ of the two premisses of cut (*cut-height*, for shortness). The proof can be organized in four exhaustive cases: in **case 1** one of the two premisses is an initial sequent. In **case 2** the cut formula is not principal in the left premiss only and in **case 3** it is not principal in the right premiss. Finally, in **case 4**, the cut formula is principal in both premisses.

In **case 1**, the conclusion of Cut is an initial sequent and, therefore, we can dispense with that instance of Cut.

In **case 2**, we transform the derivation by: (i) applying, if the last rule applied in $\mathcal{D}_1$ has a variable condition, an height-preserving admissible substitution to rename its *eigenvariable* with a fresh one; then, (ii) we apply one or two instances of Cut on each premiss of $\mathcal{D}_1$ with the conclusion of $\mathcal{D}_2$. These instances of Cut are admissible by IH because they have a lesser cut-height. We finish (iii) by applying an instance of the last rule applied in $\mathcal{D}_1$ and, if needed, some instances of contraction.

Case 3 is similar to case 2. To illustrate, suppose the last step of $\mathcal{D}_2$ is by $L\lambda$, we transform

$$\dfrac{\Omega; \Gamma \Rightarrow \Delta, w : A \quad \dfrac{y \stackrel{v}{\approx} t, \Omega'; w : A, v : B[y/x], \Gamma' \Rightarrow \Delta'}{\Omega'; w : A, v : \lambda x.B.t, \Gamma' \Rightarrow \Delta'}\; {\scriptstyle L\lambda}}{\Omega, \Omega'; v : \lambda x.B.t, \Gamma', \Gamma \Rightarrow \Delta, \Delta'}\; {\scriptstyle Cut}$$

into

$$\dfrac{\dfrac{\Omega; \Gamma \Rightarrow \Delta, w : A \quad \dfrac{y \stackrel{v}{\approx} t, \Omega'; w : A, v : B[y/x], \Gamma' \Rightarrow \Delta'}{z \stackrel{v}{\approx} t, \Omega'; w : A, v : B[z/x], \Gamma' \Rightarrow \Delta'}\; {\scriptstyle [z/y]}}{z \stackrel{v}{\approx} t, \Omega, \Omega'; v : B[z/x], \Gamma, \Gamma' \Rightarrow \Delta, \Delta'}\; {\scriptstyle Cut}}{\Omega, \Omega'; v : \lambda x.B.t, \Gamma', \Gamma \Rightarrow \Delta, \Delta'}\; {\scriptstyle L\lambda}$$

In **case 4**, we have subcases according to the principal operator of $w : A$. We consider only the case where $w : A$ is of shape $w : \lambda x.B.t$ (for $w : A$ of shape either $w : B \wedge C$ or $w : \forall x B$ or $w : \Box B$, see [10, Thm. 12.9]; when it is of shape $w : \neg B$ there is no problem). We transform

$$\dfrac{\dfrac{z \stackrel{v}{\approx} t, \Omega; \Gamma \Rightarrow \Delta, v : \lambda x.B.t, v : B[z/x]}{z \stackrel{v}{\approx} t, \Omega; \Gamma \Rightarrow \Delta, v : \lambda x.B.t}\; {\scriptstyle R\lambda} \quad \dfrac{y \stackrel{v}{\approx} t, \Omega'; v : B[y/x], \Gamma' \Rightarrow \Delta'}{\Omega'; v : \lambda x.B.t, \Gamma' \Rightarrow \Delta'}\; {\scriptstyle L\lambda}}{z \stackrel{v}{\approx} t, \Omega, \Omega'; \Gamma', \Gamma \Rightarrow \Delta, \Delta'}\; {\scriptstyle cut}$$

into

$$\dfrac{\dfrac{\dfrac{z \stackrel{v}{\approx} t, \Omega; \Gamma \Rightarrow \Delta, v : B[z/x], v : \lambda x.B.t \quad \Omega'; v : \lambda x.B.t, \Gamma' \Rightarrow \Delta'}{z \stackrel{v}{\approx} t, \Omega, \Omega'; \Gamma \Rightarrow \Delta, \Delta', v : B[z/x]}\; {\scriptstyle Cut_1} \quad \dfrac{y \stackrel{v}{\approx} t, \Omega'; v : B[y/x], \Gamma' \Rightarrow \Delta'}{z \stackrel{v}{\approx} t, \Omega'; v : B[z/x], \Gamma' \Rightarrow \Delta'}\; {\scriptstyle [z/y]}}{z \stackrel{v}{\approx} t, z \stackrel{v}{\approx} t, \Omega, \Omega', \Omega', ; \Gamma', \Gamma \Rightarrow \Delta, \Delta', \Delta'}\; {\scriptstyle Cut_2}}{z \stackrel{v}{\approx} t, \Omega, \Omega'; \Gamma', \Gamma \Rightarrow \Delta, \Delta}\; {\scriptstyle LC_\Omega + LC + RC}$$

where Cut_1 is admissible because it has a lesser cut-height, and Cut_2 is admissible because its cut-formula has a lower weight. □

3.2.2 Soundness and Completeness

Definition 12. Given a model $\mathcal{M} =< \mathcal{W}, \mathcal{R}, \mathcal{D}, \mathcal{V} >$, let $f : LAB \cup VAR \longrightarrow \mathcal{W} \cup D$ be a function mapping labels to worlds of the model and mapping variables to objects of the union of the domains of the model. We say that:

$\mathcal{M}$ satisfies $w : A$ under f iff $f \models^{\mathcal{M}}_{f(w)} A$

$\mathcal{M}$ satisfies $x \in w$ under f iff $f(x) \in D_{f(w)}$

$\mathcal{M}$ satisfies $w\mathscr{R}v$ under f iff $f(w)\mathcal{R}f(v)$

$\mathcal{M}$ satisfies $x \overset{w}{\approx} t$ under f iff $\begin{cases} V(t, f(w)) = f(x) & \text{if } t \text{ is an individual constant;} \\ f(t) = f(x) & \text{if } t \text{ is a variable;} \end{cases}$

Given a sequent $\Omega; \Gamma \Rightarrow \Delta$ we say that it is $\mathbf{Q\lambda.L}$-*valid* iff for every pair $\mathcal{M}, f$ where $\mathcal{M}$ is a model for $\mathbf{Q\lambda.L}$, if $\mathcal{M}$ satisfies under f all formulas in Ω, Γ then $\mathcal{M}$ satisfies under f some formula in Δ.

Theorem 13 (Soundness)**.** *If a sequent* $\Omega; \Gamma \Rightarrow \Delta$ *is* $\mathbf{G3Q\lambda.L}$*-derivable, then it is* $\mathbf{Q\lambda.L}$*-valid.*

Proof. The proof is by induction on the height of the $\mathbf{G3Q\lambda.L}$-derivation of $\Omega; \Gamma \Rightarrow \Delta$. The base case holds since Γ and Δ have one formula in common, and it is easy to see that the propositional rules, the rules for $\forall$, and the rules for $\Box$ preserve validity on every model.

For rule $L\lambda$, let the last step of $\mathcal{D}$ be:

$$\frac{y \overset{w}{\approx} t, \Omega; w : A[y/x], \Gamma \Rightarrow \Delta}{\Omega; w : \lambda x.A.t, \Gamma \Rightarrow \Delta} \; L\lambda$$

Let $\mathcal{M}$ and f be such that $\mathcal{M}$ satisfies under f all formulas in Ω, Γ and the formula $w : \lambda x.A.t$. We have to prove that $\mathcal{M}$ satisfies under f also some formula in Δ. Since $f \models^{\mathcal{M}}_{f(w)} \lambda x.A.t$, we know that, in $f(w)$, the term t denotes some object $a \in D$ and that $f^{y \triangleright a} \models^{\mathcal{M}}_{f(w)} A[y/x]$, where y does not occur in Ω, Γ, A. This implies that $\mathcal{M}$ satisfies under $f^{y \triangleright a}$ all formulas in $y \overset{w}{\approx} t, \Omega; w : A[y/x], \Gamma$, and therefore, by IH, $\mathcal{M}$ satisfies under $f^{y \triangleright a}$ also some formula in Δ. Since y does not occur in Δ, we conclude that $\mathcal{M}$ satisfies under f some formula in Δ.

For rule $R\lambda$, let the last step of $\mathcal{D}$ be:

$$\frac{y \overset{w}{\approx} t, \Omega; \Gamma \Rightarrow \Delta, w : \lambda x.A.t, w : A[y/x]}{y \overset{w}{\approx} t, \Omega; \Gamma \Rightarrow \Delta, w : \lambda x.A.t} \; R\lambda$$

We consider an arbitrary pair $\mathcal{M}, f$ satisfying all formulas in $y \overset{w}{\approx} t, \Omega, \Gamma$. By IH we know that they satisfy also some formula in $\Delta, w : \lambda x.A.t, w : A[y/x]$. If they satisfy some formula in $\Delta, w : \lambda x.A.t$ there is nothing to prove. Else, $\mathcal{M}$ satisfies under f the formulas $w : A[y/x]$ and $y \overset{w}{\approx} t$, in this case it is easy to see that $\mathcal{M}$ satisfies under f also $\lambda x.A.t$.

The rules for identity preserves validity on every model: the proof is standard for rules $Ref_=$ and *Repl*, and for $RigVar$, it depends on the fact that variables are rigid designators. Also the rules $DenVar$ and $DenId$ preserves validity on every model. For $DenVar$ this depends on the fact that variables denote in every world. For $DenId$, this holds because the fact that $\mathcal{M}$ satisfies $y \overset{w}{\approx} x$ under f means that $f(x) = f(y)$. Therefore, $\mathcal{M}$ must also satisfy under f the formula $w : x = y$.

If the last step in $\mathcal{D}$ is by a non-logical rule R of $\mathbf{G3Q\lambda.L}$, we can show that R preserves $\mathbf{Q\lambda.L}$-validity. To illustrate, if R is Rig, we consider a model $\mathcal{M}$ where constants are rigid designators, see Table 4. Given a generic f such that $\mathcal{M}$ satisfies under f all formulas in

$w\mathscr{R}v, x \overset{w}{\approx} t, \Omega, \Gamma$. We have to prove that $\mathcal{M}$ satisfies under f also some formula in Δ. By rigidity, $\mathcal{M}$ satisfies under f also $x \overset{v}{\approx} t$ and, by IH, we conclude that it satisfies some formula in Δ. □

Theorem 14 (Completeness). *If a sequent $\Omega; \Gamma \Rightarrow \Delta$ is* **Q**λ**.L***-valid, it is* **G3Q**λ**.L***-derivable.*

Proof. The proof is organized in four main steps. First, in Def. 15, we sketch a root-first **G3Q**λ**.L**-proof-search procedure. Second, in Def. 16, we define the notion of saturation for a branch of a **G3Q**λ**.L**-proof-search tree and, in Proposition 17, we show that, for every sequent, a **G3Q**λ**.L**-proof-search either gives us a **G3Q**λ**.L**-derivation of that sequent, or it has a saturated branch. Third, in Def. 19, we define a model $\mathcal{M}^{\mathcal{B}}$ out of a saturated branch $\mathcal{B}$. Finally, in Lemma 20, we prove that $\mathcal{M}^{\mathcal{B}}$ is a model for **Q**λ**.L** that falsifies $\Omega; \Gamma \Rightarrow \Delta$. □

Definition 15. A **G3Q**λ**.L**-*proof-search tree* for a sequent $\Omega; \Gamma \Rightarrow \Delta$ is a tree of sequents generated according to the following inductive procedure. At **step 0** we write the one node tree $\Omega; \Gamma \Rightarrow \Delta$. At **step** $\mathbf{n+1}$, if all leaves of the tree generated at step n are initial sequents, the procedure ends. Else, we continue the bottom-up construction by applying, to each leaf that is not an initial sequent, each applicable instance of a rule of **G3Q**λ**.L** or, if no rule instance is applicable, we copy the leaf on top of itself. For rules $Ref_=, Ref_{\mathcal{W}}, Ser, NonEm, Cons, DenVar$ and Tot, we consider applicable only instances where, save for *eigenvariables*, all terms and labels occurring in the active formula of that instance already occur in the leaf. See [10, Thm. 12.14] for the details of the inductive procedure.

Definition 16 (Saturation)**.** A branch $\mathcal{B}$ of a **G3Q**λ**.L**-proof-search tree for a sequent is **Q**λ**L**-*saturated* if it satisfies the following conditions, where $\mathbf{\Gamma}$ ($\mathbf{\Delta}$) is the union of the antecedents (succedents) occurring in that branch,

1. no $w : p$ occurs in $\mathbf{\Gamma} \cap \mathbf{\Delta}$;
2. if $w : \neg A$ is in $\mathbf{\Gamma}$, then $w : A$ is in $\mathbf{\Delta}$;
3. if $w : \neg A$ is in $\mathbf{\Delta}$, then $w : A$ is in $\mathbf{\Gamma}$;
4. if $w : A \wedge B$ is in $\mathbf{\Gamma}$, then both $w : A$ and $w : B$ are in $\mathbf{\Gamma}$;
5. if $w : A \wedge B$ is in $\mathbf{\Delta}$, then at least one of $w : A$ and $w : B$ is in $\mathbf{\Delta}$;
6. if both $w : \forall x A$ and $y \in w$ are in $\mathbf{\Gamma}$, then $w : A(y/x)$ is in $\mathbf{\Gamma}$;
7. if $w : \forall x A$ is in $\mathbf{\Delta}$, then, for some z, $w : A(z/x)$ is in $\mathbf{\Delta}$ and $z \in w$ is in $\mathbf{\Gamma}$;
8. if both $w : \Box A$ and $w\mathscr{R}v$ are in $\mathbf{\Gamma}$, then $v : A$ is in $\mathbf{\Gamma}$;
9. if $w : \Box A$ is in $\mathbf{\Delta}$, then, for some u, $u : A$ is in $\mathbf{\Delta}$ and $w\mathscr{R}u$ is in $\mathbf{\Gamma}$;
10. if $w : \lambda x.A.t$ is in $\mathbf{\Gamma}$, then, for some z, both $z \overset{w}{\approx} t$ and $w : A[z/x]$ are in $\mathbf{\Gamma}$;
11. if $w : \lambda x.A.t$ is in $\mathbf{\Delta}$ and $y \overset{w}{\approx} t$ is in $\mathbf{\Gamma}$, then $w : A[y/x]$ is in $\mathbf{\Delta}$;
12. if the principal formulas of some instance of one of $Ref_=$, $Repl$, $RigVar$, $DenVar$, and $DenId$ is in $\mathbf{\Gamma}$, then also the corresponding active formulas are in $\mathbf{\Gamma}$.

13_R. if R is a non-logical rule of **G3tm.L**, then for each set of principal formulas of R that are in $\mathbf{\Gamma}$ also the corresponding active formulas are in $\mathbf{\Gamma}$ (for some *eigenvariable* of R, if any).

Proposition 17. Let us consider a **G3Qλ.L**-proof-search tree for a sequent $\mathcal{S}$, two cases are possible: either the tree is finite or not. If the tree is finite, given that all of its leaves are initial sequents and that it grows by applying rules of **G3Qλ.L**, it is a **G3Qλ.L**-derivation of $\mathcal{S}$ and, by Theorem 13, $\mathcal{S}$ is **Qλ.L**-vaild. Else, by König's Lemma, the tree has an infinite branch $\mathcal{B}$ that is **Qλ.L**-saturated since every applicable rule instance has been applied at some step of the construction of the tree.

Proposition 18. It is immediate to notice that, by saturation under rule $Ref_=$ and *Repl* the set of variables x, y such that $w : x = y$ is in $\mathbf{\Gamma}$ form an equivalence class and that, by saturation under $RigVar$, the same equivalence class holds with respect to each label v occurring in $\mathbf{\Gamma}, \mathbf{\Delta}$.

Definition 19. Let $\mathcal{B}$ be a saturated branch of a **G3Qλ.L**-proof-search tree for a sequent. The model $\mathcal{M}^\mathcal{B} =< \mathcal{W}^\mathcal{B}, \mathcal{R}^\mathcal{B}, \mathcal{D}^\mathcal{B}, \mathcal{V}^\mathcal{B} >$ is defined from $\mathcal{B}$ as follows:

- $\mathcal{W}^\mathcal{B}$ is the set of all labels occurring in $\mathcal{B}$;
- $\mathcal{R}^\mathcal{B}$ is such that $w\mathcal{R}v$ iff $w\mathscr{R}v$ occurs in $\mathcal{B}$;
- $\mathcal{D}^\mathcal{B}$ is such that, for each $w \in \mathcal{W}^\mathcal{B}$, D_w is the set containing, for each variable x such that $x \in w$ occurs in $\mathcal{B}$, the equivalence class $[x]$ of all x, y such that $w : x = y$ occurs in $\mathcal{B}$;
- $\mathcal{V}^\mathcal{M}$ is defined as follows:
  - for every predicate $P^n \in \mathcal{S}^\lambda$, $V(P^n, w)$ is the set of all n-tuples of equivalence classes of variables $< [x_1], \dots, [x_n] >$ such that $w : Px_1, \dots, x_n$ occurs in $\mathbf{\Gamma}$;
  - for every constant $c \in \mathcal{S}^\lambda$, $V(c, w)$ is $[x]$ if $x \overset{w}{\approx} c$ occurs in $\mathbf{\Gamma}$, else it is undefined.

Lemma 20. *If $\mathcal{M}^\mathcal{B}$ is the model defined from a saturated branch $\mathcal{B}$ of a* **G3Qλ.L**-*proof-search tree for a sequent $\Omega; \Gamma \Rightarrow \Delta$ and σ is the assignment defined by $\sigma(x) = [x]$, then, for each labelled formula $w : A$ occurring in $\mathcal{B}$,*

1. $\sigma \models^{\mathcal{M}^\mathcal{B}}_w A$ *iff* $w : A$ *occurs in* $\mathbf{\Gamma}$
2. $\mathcal{M}^\mathcal{B}$ *is a model for* **Qλ.L**.

Proof. The proof of **claim 1** is by induction on the weight of $w : A$. The base case holds thanks to the definition of $\mathcal{V}^\mathcal{B}$, and the inductive cases depends on the construction of $\mathcal{M}^\mathcal{B}$ and on properties 2–12 of the definition of saturated branch.

To illustrate, suppose $w : A \equiv w : \lambda xB.t$. If $w : A$ occurs in $\mathbf{\Gamma}$, then, by Def. 16.10, for some z, $z \overset{w}{\approx} t$ and $w : B[z/x]$ are in $\mathbf{\Gamma}$. This implies that $\sigma_w(t) = [z]$ and, by IH, that $\sigma^{x \triangleright [z]} \models^\mathcal{B}_w B$. Thus, $\sigma \models^\mathcal{B}_w \lambda x.B.t$. Else, $w : \lambda x.B.t$ is in $\mathbf{\Delta}$ and, for each t such that $y \overset{w}{\approx} t$ is in $\mathbf{\Gamma}$ (if any), Def. 16.11 entails that $w : B[y/x]$ is in $\mathbf{\Delta}$. By construction we have that that $\sigma_w(t) = [y]$ and, thanks to IH (and 16.1), $\sigma^{x \triangleright [y]} \not\models^{\mathcal{M}^\mathcal{B}}_w B$. We conclude that $\sigma \not\models^{\mathcal{M}^\mathcal{B}}_w \lambda x.B.t$.

Claim 2 holds thanks to property 13_R of saturated branch: if a non-logical rule R is in **G3Qλ.L**, then we have to show that $\mathcal{M}^\mathcal{B}$ satisfies the semantic property corresponding to R. This holds by construction of $\mathcal{M}^\mathcal{B}$ since $\mathcal{B}$ is saturated with respect to rule R. For example, for rule Rig, we have to prove that $w\mathcal{R}^\mathcal{B}v$ and $s_w(t) \in D^\mathcal{B}$ – i.e., $s_w(t)$ is defined in $\mathcal{M}^\mathcal{B}$ – implies that $s_v(t) = \sigma_w(t)$. Suppose that $w\mathscr{R}v$ is in $\mathbf{\Gamma}$, if t is a variable, then, 16.12 entails that, for all y, if $y \overset{w}{\approx} t$ is in $\mathbf{\Gamma}$, then, also $y \overset{v}{\approx} t$ is in $\mathbf{\Gamma}$ (by saturation under $DenVar$, $DenId$, $RigVar$, and *Repl*). Else, t is a constant and if, for some y, $y \overset{w}{\approx} t$ is in $\mathbf{\Gamma}$, saturation under rule Rig entails that $y \overset{v}{\approx} t$ is in $\mathbf{\Gamma}$. In both cases $\mathcal{M}^\mathcal{B}$ behaves as desired. □

4 Conclusion

We have introduced labelled sequent calculi that characterize the QMLs with non-rigid and non-denoting terms introduced in [6, 7], and we have studied their structural properties. To the best of our knowledge, this is the first proof-theoretic study of these logics. In [6, 7] prefixed tableaux for these logics have been considered, but there is no study of their structural properties. Notice that, even if we have considered only the $Q\lambda$-extensions of propositional modal logics $\mathbf{L}$ in the cube of normal modalities, the present approach can be extended, in a modular way, to the $Q\lambda$-extensions of any propositional modal logic whose class of models is defined by first-order definable modal logics (by applying, if needed, the *geometrisation technique* introduced in [5]). For example, we can introduce a calculus characterizing validity in the class of all constant domain models satisfying *confluence*: $\forall w, v, u \in \mathcal{W}(w\mathcal{R}v \wedge w\mathcal{R}u \supset \exists w' \in \mathcal{W}(v\mathcal{R}w' \wedge u\mathcal{R}w')$. From [4], we know that confluence corresponds to Geach's axiom $2 := \Diamond\Box A \supset \Box\Diamond A$ and that the quantified modal axiomatic system $Q.2 \oplus BF$ is incomplete with respect to the class of all confluent constant domain models. Nevertheless, confluence is a geometric property, and it can be expressed in labelled calculi by the rule:

$$\frac{v\mathscr{R}w', u\mathscr{R}w', w\mathscr{R}v, w\mathscr{R}u, \Omega; \Gamma \Rightarrow \Delta}{w\mathscr{R}v, w\mathscr{R}u, \Omega; \Gamma \Rightarrow \Delta}\ {\scriptstyle Conf,\ w' \text{ fresh}}$$

It can be proved that the labelled calculus $\mathbf{G3Q\lambda.K}+\{Cons, Conf\}$ is sound and complete with respect to the class of confluent constant domain models.

The calculi introduced here are somehow related to the ones we gave in [3] for the *indexed epistemic logics* studied in [2]. The QMLs studied here are less general than the ones in [2], but they have the advantage of being simpler and of involving no major departure from the standard modal language. Therefore the present approach to non-rigid and non-denoting terms can easily be extended to most variants of QMLs. For example, it carries over to labelled calculi for *term-modal logics* [11].

References

[1] Agata Ciabattoni, Revantha Ramanayake, and Heinrich Wansing. Hypersequent and display calculi - a unified perspective. *Studia Logica*, 102(6):1245–1294, 2014.

[2] Giovanna Corsi and Eugenio Orlandelli. Free quantified epistemic logics. *Studia Logica*, 101(6):1159–1183, 2013.

[3] Giovanna Corsi and Eugenio Orlandelli. Sequent calculi for indexed epistemic logics. In *Proceedings of ARQNL 2016*, pages 21–35. CEUR-WS, 2016.

[4] Max Cresswell. Incompleteness and the barcan formula. *J. Phil. Logic*, 24(4):379–403, 1995.

[5] Roy Dyckhoff and Sara Negri. Geometrisation of first-order logic. *Bulletin of Symbolic Logic*, 21(2):123–163, 2015.

[6] Melvin Fitting. On quantified modal logic. *Fundamenta Informaticae*, 39(1–2):105–121, 1999.

[7] Melvin Fitting and Richard L. Mendelsohn. *First-Order Modal Logic*. Springer, 1998.

[8] Gottlob Frege. Über sinn und bedeutung. *Zeitschrift für Philosophie Und Philosophische Kritik*, 100(1):25–50, 1892.

[9] Sara Negri and Jan von Plato. Cut elimination in the presence of axioms. *Bullettin of Symbolic Logic*, 4(4):418–435, 1998.

[10] Sara Negri and Jan von Plato. *Proof Analysis*. Cambridge University Press, 2011.

[11] Eugenio Orlandelli and Giovanna Corsi. Decidable term-modal logics. In *Proceedings of EUMAS 2017 and AT 2017*, pages 1–15. Springer, Forthcoming.

[12] Bertrand Russell. On denoting. *Mind*, 14(56):479–493, 1905.

System Demonstration: The Higher-Order Prover Leo-III

Alexander Steen[1] and Christoph Benzmüller[2,1]

[1] Freie Universität Berlin, Institute for Computer Science, Berlin, Germany
[2] University of Luxembourg, FSTC, Luxembourg
{a.steen|c.benzmueller}@fu-berlin.de

Abstract

The higher-order ATP system Leo-III is demonstrated. Leo-III supports flexible and effective reasoning in every common semantical variation of normal modal logics.

Many powerful automated and interactive theorem proving systems for first-order and higher-order logics have been developed over the past decades. However, with a few notable exceptions, most available systems focus on classical logics only. In particular for quantified non-classical logics only a small number of implemented systems is available to date. This is in contrast to an increasing number of challenging and interesting applications for such systems in artificial intelligence, computer science, mathematics and philosophy [10, 8, 9, 5, 6, 7]. Metaphysics, for example, is an area where higher-order modal logics (HOMLs) play an important role. The development of ATPs for HOMLs, however, is still in its infancy. The Leo-III prover, which is presented here, is addressing this gap.

Leo-III [4] is in the first place an automated theorem prover for classical higher-order logic (HOL) with Henkin semantics and choice [1]. Despite its primary focus on HOL, Leo-III comes with effective means for reasoning in HOMLs. In fact, reasoning in every normal modal logic variant is supported in Leo-III. To achieve this, the prover internally implements a shallow semantical embedding approach [2, 3]. The key idea of this approach is to provide and exploit faithful mappings for HOML input problems to HOL. This is orthogonal to the direct implementation of specialised theorem provers, which usually focus on a small subset of modal logic systems only. The semantical embedding approach realised in Leo-III, in contrast, allows for a quick adaptation to a broad variety of expressive, non-classical logics.

Leo-III in particular supports (but is not limited to) first-order and higher-order extensions of the well known modal logic cube for different concrete choices of

Quantification semantics, including cumulative, decreasing, constant and varying domains,

Rigidity, including rigid and world-dependent constant symbols, and

Consequence, including the usual notions of local and global consequence.

When taking all possible parameter combinations into account this amounts to more than 120 supported HOMLs [3, §2.2]. The exact number of logics is in fact much higher, since Leo-III also supports multi-modal logics and offers fine-grained control over more specific combinations of the above semantical parameters (e.g. different quantification semantics per type).

Higher-order modal logics. HOMLs as addressed here are extensions of HOL, which has been proposed by Church, and further studied by Henkin, Andrews and others. HOL provides lambda-notation as an elegant means to denote unnamed functions, predicates and sets (by their characteristic functions). HOML, in turn, augments HOL with a set of modal operators $\Box^i$, $i \in I$, for some index set I, and is equipped with a suitable combination of HOL semantics and a Kripke-style modal semantics. In our approach an adequate notion of Henkin semantics for both HOML and HOL is assumed.

Figure 1: Example modal logic problem input for Leo-III. The first three lines specify the exact modal logic (here a S5 logic with rigid constants, constant domain quantification and global consequence) under which the problem is to be analyzed. The conjecture is represented by the last two lines and encodes the formula $\forall P_{\iota\to o} \forall F_{\iota\to\iota} \forall X_\iota \exists G_{\iota\to\iota} (\Diamond\Box P(F(X)) \Rightarrow \Box P(G(X)))$.

```
thf(s5_spec, logic, ($modal := [
      $constants := $rigid, $quantification := $constant,
      $consequence := $global, $modalities := $modal_system_S5 ])).
thf(becker,conjecture,( ! [P:$i>$o,F:$i>$i, X:$i]: (? [G:$i>$i]:
      (($dia @ ($box @ (P @ (F @ X)))) => ($box @ (P @ (G @ X))))))).
```

Automation of HOML. In order to automate reasoning in HOMLs, Leo-III exploits the semantical embedding approach and internally translates modal logic problems into equivalent problems formulated within classical higher-order logic. To that end, the de-facto standard TPTP THF input syntax is augmented to include the modal connectives. Fig. 1 displays an example modal logic formula that is an instance of a corollary of Becker's postulate, with `$box` and `$dia` representing the (mono-)modal operators $\Box$ and $\Diamond$, respectively, and the usual TPTP text representatives of the remaining logical connectives. This example formula is valid in S5 but not in any weaker system.

The logic specification format displayed in the example from Fig. 1 is stemming from an ongoing TPTP language extension proposal.[1] In this logic specification, the identifiers `$constants`, `$quantification` and `$consequence` specify the exact semantical settings for the rigidity of constant symbols, the quantification semantics and the consequence relation, respectively. Finally, `$modalities` specify the properties of the modal connectives. Valid values are either pre-defined identifiers representing the usual modal logic systems, as in `$modalities := $modal_system_S5` for the specification of an S5 modal logic, or lists of individual modal axiom schemes, as in `$modalities := [$modal_axiom_K, $modal_axiom_B]`.

The reasoning process of Leo-III proceeds as follows:

1. The user inputs a HOML problem in the adapted TPTP syntax from above (Fig. 1).
2. Leo-III analyses the logic specification contained within the input and automatically selects the definitions and axioms to be added to the embedded problem representation.
3. The problem statement itself is translated into its embedded equivalent using the definitions from the previous step.
4. Finally, Leo-III starts reasoning in (meta-logic) HOL and returns SZS compliant result information and, if successful, also a proof object just as for standard HOL problems.

Summary. At the ARQNL 2018 event we will demonstrate Leo-III, which, in terms of supported logics, is the most widely applicable automated theorem prover available to date. The embedding procedure is also available as stand-alone implementation at github.com/leoprover and can be used in conjunction with every THF-compliant ATP.

[1] See http://www.cs.miami.edu/~tptp/TPTP/Proposals/LogicSpecification.html for more details.

A Installation and Usage of Leo-III

Acquisition and Installation

Leo-III is freely available on GitHub (https://github.com/leoprover/Leo-III) under BSD-3 license. The most current release (version 1.2) is accessible under https://github.com/leoprover/Leo-III/releases/latest. To get it, simply download the source archive and extract it so some location.

```
> wget https://github.com/leoprover/Leo-III/archive/v1.2.tar.gz
> tar -xvzf v1.2.tar.gz
```

After extraction, Leo-III can be built using Make. Simply `cd` to the extracted directory and run `make`:

```
> cd Leo-III-1.2/
> make
```

After building, there should be a directory `bin/`, relative from the current directory. This directory contains the binary `leo3` of Leo-III.

Leo-III can optionally be installed by invoking

```
> make install
```

which copies the binary to the directory `$HOME/bin` by default.

Usage

Leo-III is invoked via command-line (assuming the leo3 executable is in `$PATH`):

For the example of Becker's postulate of Fig. 1, running

```
> leo3 becker.p -p
```

will invoke Leo-III for proving this conjecture (the `-p` option enables the output of a proof certificate). This will produce the following result:

```
% Axioms used in derivation (1): mrel_meuclidean
% No. of inferences in proof: 22
% No. of processed clauses: 14
% No. of generated clauses: 77
[...]
% SZS status Theorem for becker.p : 4179 ms resp. 1443 ms w/o parsing
% SZS output start CNFRefutation for becker.p
thf(mworld_type, type, mworld: $tType).
thf(mrel_type, type, mrel: (mworld > (mworld > $o))).
thf(meuclidean_type, type, meuclidean: ((mworld > (mworld > $o)) > $o)).
thf(meuclidean_def, definition, (meuclidean = (^ [A:(mworld > (mworld > $o))]: ! [B:mworld,C:mworld,D:mworld
     ]: (((A @ B @ C) & (A @ B @ D)) => (A @ C @ D))))).
thf(mvalid_type, type, mvalid: ((mworld > $o) > $o)).
thf(mvalid_def, definition, (mvalid = ('!' @ mworld))).
thf(mimplies_type, type, mimplies: ((mworld > $o) > ((mworld > $o) > (mworld > $o)))).
thf(mimplies_def, definition, (mimplies = (^ [A:(mworld > $o),B:(mworld > $o),C:mworld]: ((A @ C) => (B @ C))
     ))).
thf(mdia_type, type, mdia: ((mworld > $o) > (mworld > $o))).
thf(mdia_def, definition, (mdia = (^ [A:(mworld > $o),B:mworld]: ? [C:mworld]: ((mrel @ B @ C) & (A @ C))))).
thf(mbox_type, type, mbox: ((mworld > $o) > (mworld > $o))).
thf(mbox_def, definition, (mbox = (^ [A:(mworld > $o),B:mworld]: ! [C:mworld]: ((mrel @ B @ C) => (A @ C)))))
     .
thf(mexists_const__o__d_i_t__d_i_c__type, type, mexists_const__o__d_i_t__d_i_c_: ((($i > $i) > (mworld > $o))
      > (mworld > $o))).
thf(mexists_const__o__d_i_t__d_i_c__def, definition, (mexists_const__o__d_i_t__d_i_c_ = (^ [A:(($i > $i) > (
     mworld > $o)),B:mworld]: ? [C:($i > $i)]: (A @ C @ B)))).
thf(mforall_const__o__d_i_t__o_mworld_t__d_o_c__c__type, type, mforall_const__o__d_i_t__o_mworld_t__d_o_c__c_
     : ((($i > (mworld > $o)) > (mworld > $o)) > (mworld > $o))).
```

```
thf(mforall_const__o__d_i_t__o_mworld_t__d_o_c__c__def, definition, (
      mforall_const__o__d_i_t__o_mworld_t__d_o_c__c_ = (^ [A:(($i > (mworld > $o)) > (mworld > $o)),B:mworld]:
       ! [C:($i > (mworld > $o))]: (A @ C @ B)))).
thf(mforall_const__o__d_i_c__type, type, mforall_const__o__d_i_c_: (($i > (mworld > $o)) > (mworld > $o))).
thf(mforall_const__o__d_i_c__def, definition, (mforall_const__o__d_i_c_ = (^ [A:($i > (mworld > $o)),B:mworld
      ]: ! [C:$i]: (A @ C @ B)))).
thf(mforall_const__o__d_i_t__d_i_c__type, type, mforall_const__o__d_i_t__d_i_c_: ((($i > $i) > (mworld > $o))
       > (mworld > $o))).
thf(mforall_const__o__d_i_t__d_i_c__def, definition, (mforall_const__o__d_i_t__d_i_c_ = (^ [A:(($i > $i) > (
      mworld > $o)),B:mworld]: ! [C:($i > $i)]: (A @ C @ B)))).
thf(sk1_type, type, sk1: mworld).
thf(sk2_type, type, sk2: ($i > (mworld > $o))).
thf(sk3_type, type, sk3: ($i > $i)).
thf(sk4_type, type, sk4: $i).
thf(sk5_type, type, sk5: mworld).
thf(sk6_type, type, sk6: (($i > $i) > mworld)).
thf(1,conjecture,((mvalid @ (mforall_const__o__d_i_t__o_mworld_t__d_o_c__c_ @ (^ [A:($i > (mworld > $o))]: (
      mforall_const__o__d_i_t__d_i_c_ @ (^ [B:($i > $i)]: (mforall_const__o__d_i_c_ @ (^ [C:$i]: (
      mexists_const__o__d_i_t__d_i_c_ @ (^ [D:($i > $i)]: (mimplies @ (mdia @ (mbox @ (A @ (B @ C)))) @ (mbox
      @ (A @ (D @ C))))))))))))))),file('becker.p',1)).
thf(2,negated_conjecture,((~ (mvalid @ (mforall_const__o__d_i_t__o_mworld_t__d_o_c__c_ @ (^ [A:($i > (mworld
      > $o))]: (mforall_const__o__d_i_t__d_i_c_ @ (^ [B:($i > $i)]: (mforall_const__o__d_i_c_ @ (^ [C:$i]: (
      mexists_const__o__d_i_t__d_i_c_ @ (^ [D:($i > $i)]: (mimplies @ (mdia @ (mbox @ (A @ (B @ C)))) @ (mbox
      @ (A @ (D @ C)))))))))))))))),inference(neg_conjecture,[status(cth)],[1])).
thf(5,plain,((~ (! [A:mworld,B:($i > (mworld > $o)),C:($i > $i),D:$i]: ? [E:($i > $i)]: ((? [F:mworld]: ((
      mrel @ A @ F) & ! [G:mworld]: ((mrel @ F @ G) => (B @ (C @ D) @ G)))) => (! [F:mworld]: ((mrel @ A @ F)
      => (B @ (E @ D) @ F))))))),inference(defexp_and_simp_and_etaexpand,[status(thm)],[2])).
thf(6,plain,((~ (! [A:mworld,B:($i > (mworld > $o)),C:($i > $i),D:$i]: ((? [E:mworld]: ((mrel @ A @ E) & ! [F
      :mworld]: ((mrel @ E @ F) => (B @ (C @ D) @ F)))) => (? [E:($i > $i)]: ! [F:mworld]: ((mrel @ A @ F) =>
      (B @ (E @ D) @ F))))))),inference(miniscope,[status(thm)],[5])).
thf(10,plain,((mrel @ sk1 @ sk5)),inference(cnf,[status(esa)],[6])).
thf(4,axiom,((meuclidean @ mrel)),file('becker.p',mrel_meuclidean)).
thf(15,plain,((! [A:mworld,B:mworld,C:mworld]: (((mrel @ A @ B) & (mrel @ A @ C)) => (mrel @ B @ C)))),
      inference(defexp_and_simp_and_etaexpand,[status(thm)],[4])).
thf(16,plain,(! [C:mworld,B:mworld,A:mworld] : ((~ (mrel @ A @ B)) | (~ (mrel @ A @ C)) | (mrel @ B @ C))),
      inference(cnf,[status(esa)],[15])).
thf(17,plain,(! [C:mworld,B:mworld,A:mworld] : ((~ (mrel @ A @ C)) | (mrel @ B @ C) | ((mrel @ sk1 @ sk5) !=
      (mrel @ A @ B)))),inference(paramod_ordered,[status(thm)],[10,16])).
thf(18,plain,(! [A:mworld] : ((~ (mrel @ sk1 @ A)) | (mrel @ sk5 @ A))),inference(pattern_uni,[status(thm)
      ],[17:[bind(A, $thf(sk1)),bind(B, $thf(sk5))]])).
thf(40,plain,(! [A:mworld] : ((~ (mrel @ sk1 @ A)) | (mrel @ sk5 @ A))),inference(simp,[status(thm)],[18])).
thf(9,plain,(! [A:mworld] : ((~ (mrel @ sk5 @ A)) | (sk2 @ (sk3 @ sk4) @ A))),inference(cnf,[status(esa)
      ],[6])).
thf(7,plain,(! [A:($i > $i)] : ((~ (sk2 @ (A @ sk4) @ (sk6 @ (A)))))),inference(cnf,[status(esa)],[6])).
thf(11,plain,(! [A:($i > $i)] : ((~ (sk2 @ (A @ sk4) @ (sk6 @ (A)))))),inference(simp,[status(thm)],[7])).
thf(206,plain,(! [B:($i > $i),A:mworld] : ((~ (mrel @ sk5 @ A)) | ((sk2 @ (sk3 @ sk4) @ A) != (sk2 @ (B @ sk4
      ) @ (sk6 @ (B)))))),inference(paramod_ordered,[status(thm)],[9,11])).
thf(212,plain,((~ (mrel @ sk5 @ (sk6 @ (^ [A:$i]: (sk3 @ sk4)))))),inference(pre_uni,[status(thm)],[206:[bind
      (A, $thf(sk6 @ (^ [C:$i]: (sk3 @ sk4)))),bind(B, $thf(^ [C:$i]: (sk3 @ sk4)))]])).
thf(259,plain,(! [A:mworld] : ((~ (mrel @ sk1 @ A)) | ((mrel @ sk5 @ A) != (mrel @ sk5 @ (sk6 @ (^ [B:$i]: (
      sk3 @ sk4))))))),inference(paramod_ordered,[status(thm)],[40,212])).
thf(260,plain,((~ (mrel @ sk1 @ (sk6 @ (^ [A:$i]: (sk3 @ sk4)))))),inference(pattern_uni,[status(thm)],[259:[
      bind(A, $thf(sk6 @ (^ [B:$i]: (sk3 @ sk4))))]])).
thf(8,plain,(! [A:($i > $i)] : ((mrel @ sk1 @ (sk6 @ (A))))),inference(cnf,[status(esa)],[6])).
thf(12,plain,(! [A:($i > $i)] : ((mrel @ sk1 @ (sk6 @ (A))))),inference(simp,[status(thm)],[8])).
thf(269,plain,((~ ($true)),inference(rewrite,[status(thm)],[260,12])).
thf(270,plain,($false),inference(simp,[status(thm)],[269])).
% SZS output end CNFRefutation for becker.p
```

The line starting with "`% SZS status Theorem`" confirms that the conjecture is indeed a theorem and the contents between "`% SZS output start`" and "`% SZS output end`" are the proof certificate for this claim.

Becker's Postulate Embedded

The semantically embedded variant of `becker.p` that is used internally by Leo-III is as follows (this can also be generated using the stand-alone embedding tool available at https://github.com/leoprover/embed_modal):

```
% declare type for possible worlds
thf(mworld_type,type,(
    mworld: $tType )).

% declare accessibility relations
thf(mrel_type,type,(
    mrel: mworld > mworld > $o )).

% define accessibility relation properties
thf(mreflexive_type,type,(
    mreflexive: ( mworld > mworld > $o ) > $o )).

thf(mreflexive_def,definition,
    ( mreflexive
    = ( ^ [R: mworld > mworld > $o] :
        ! [A: mworld] :
          ( R @ A @ A ) ) )).

thf(meuclidean_type,type,(
    meuclidean: ( mworld > mworld > $o ) > $o )).

thf(meuclidean_def,definition,
    ( meuclidean
    = ( ^ [R: mworld > mworld > $o] :
        ! [A: mworld,B: mworld,C: mworld] :
          ( ( ( R @ A @ B )
            & ( R @ A @ C ) )
         => ( R @ B @ C ) ) ) )).

% assign properties to accessibility relations
thf(mrel_mreflexive,axiom,(
    mreflexive @ mrel )).

thf(mrel_meuclidean,axiom,(
    meuclidean @ mrel )).

% define valid operator
thf(mvalid_type,type,(
    mvalid: ( mworld > $o ) > $o )).

thf(mvalid_def,definition,
    ( mvalid
    = ( ^ [S: mworld > $o] :
        ! [W: mworld] :
          ( S @ W ) ) )).

% define nullary, unary and binary connectives which are no quantifiers
thf(mimplies_type,type,(
    mimplies: ( mworld > $o ) > ( mworld > $o ) > mworld > $o )).

thf(mimplies,definition,
    ( mimplies
    = ( ^ [A: mworld > $o,B: mworld > $o,W: mworld] :
          ( ( A @ W )
         => ( B @ W ) ) ) )).

thf(mdia_type,type,(
    mdia: ( mworld > $o ) > mworld > $o )).

thf(mdia_def,definition,
    ( mdia
    = ( ^ [A: mworld > $o,W: mworld] :
        ? [V: mworld] :
          ( ( mrel @ W @ V )
          & ( A @ V ) ) ) )).
```

```
thf(mbox_type,type,(
    mbox: ( mworld > $o ) > mworld > $o )).

thf(mbox_def,definition,
    ( mbox
    = ( ^ [A: mworld > $o,W: mworld] :
        ! [V: mworld] :
          ( ( mrel @ W @ V )
         => ( A @ V ) ) ) )).

% define exists quantifiers
thf(mexists_const_type__o__d_i_t__d_i_c_,type,(
    mexists_const__o__d_i_t__d_i_c_: ( ( $i > $i ) > mworld > $o ) > mworld > $o )).

thf(mexists_const__o__d_i_t__d_i_c_,definition,
    ( mexists_const__o__d_i_t__d_i_c_
    = ( ^ [A: ( $i > $i ) > mworld > $o,W: mworld] :
        ? [X: $i > $i] :
          ( A @ X @ W ) ) )).

% define for all quantifiers
thf(mforall_const_type__o__d_i_t__o_mworld_t__d_o_c__c_,type,(
    mforall_const__o__d_i_t__o_mworld_t__d_o_c__c_: ( ( $i > mworld > $o ) > mworld > $o ) > mworld > $o )).

thf(mforall_const__o__d_i_t__o_mworld_t__d_o_c__c_,definition,
    ( mforall_const__o__d_i_t__o_mworld_t__d_o_c__c_
    = ( ^ [A: ( $i > mworld > $o ) > mworld > $o,W: mworld] :
        ! [X: $i > mworld > $o] :
          ( A @ X @ W ) ) )).

thf(mforall_const_type__o__d_i_c_,type,(
    mforall_const__o__d_i_c_: ( $i > mworld > $o ) > mworld > $o )).

thf(mforall_const__o__d_i_c_,definition,
    ( mforall_const__o__d_i_c_
    = ( ^ [A: $i > mworld > $o,W: mworld] :
        ! [X: $i] :
          ( A @ X @ W ) ) )).

thf(mforall_const_type__o__d_i_t__d_i_c_,type,(
    mforall_const__o__d_i_t__d_i_c_: ( ( $i > $i ) > mworld > $o ) > mworld > $o )).

thf(mforall_const__o__d_i_t__d_i_c_,definition,
    ( mforall_const__o__d_i_t__d_i_c_
    = ( ^ [A: ( $i > $i ) > mworld > $o,W: mworld] :
        ! [X: $i > $i] :
          ( A @ X @ W ) ) )).


% transformed problem
thf(1,conjecture,
    ( mvalid
    @ ( mforall_const__o__d_i_t__o_mworld_t__d_o_c__c_
      @ ^ [P: $i > mworld > $o] :
          ( mforall_const__o__d_i_t__d_i_c_
          @ ^ [F: $i > $i] :
              ( mforall_const__o__d_i_c_
              @ ^ [X: $i] :
                  ( mexists_const__o__d_i_t__d_i_c_
                  @ ^ [Q: $i > $i] :
                      ( mimplies @ ( mdia @ ( mbox @ ( P @ ( F @ X ) ) ) ) @ ( mbox @ ( P @ ( Q @ X ) ) ) ) ) ) )
                           ) ) )).
```

References

[1] Peter Andrews. Church's type theory. In Edward N. Zalta, editor, *The Stanford Encyclopedia of Philosophy*. Metaphysics Research Lab, Stanford University, spring 2014 edition, 2014.

[2] Christoph Benzmüller and Lawrence Paulson. Quantified Multimodal Logics in Simple Type Theory. *Logica Universalis (Special Issue on Multimodal Logics)*, 7(1):7–20, 2013.

[3] Tobias Gleißner, Alexander Steen, and Christoph Benzmüller. Theorem provers for every normal modal logic. In *LPAR-21*, volume 46 of *EPiC Series in Computing*, pages 14–30. EasyChair, 2017.

[4] Alexander Steen and Christoph Benzmüller. The higher-order prover Leo-III. In *IJCAR 2018*, LNCS. Springer, 2018. forthcoming.

[5] Daniel Kirchner and Christoph Benzmüller and Edward N. Zalta. Mechanizing Principia Logico-Metaphysica in Functional Type Theory (Extended Abstract). In 3rd Conference on Artificial Intelligence and Theorem Proving (AITP 2018), Book of Abstracts, 2018.

[6] Christoph Benzmüller, Xavier Parent, and Leendert van der Torre. A Deontic Logic Reasoning Infrastructure. In 14th Conference on Computability in Europe, CiE 2018, Kiel, Germany, July 30-August, 2018, Proceedings, LNAI Vol. 10505, pages 114–127, Springer, 2018.

[7] David Fuenmayor and Christoph Benzmüller. A Case Study on Computational Hermeneutics: E. J. Lowe's Modal Ontological Argument. PhilPapers, https://philpapers.org/rec/FUEACS, 2017.

[8] David Fuenmayor and Christoph Benzmüller. Types, Tableaus and Gödel's God in Isabelle/HOL. *Archive of Formal Proofs*, 2017.

[9] Christoph Benzmüller and Bruno Woltzenlogel Paleo. The Inconsistency in Gödel's Ontological Argument: A Success Story for AI in Metaphysics. In IJCAI 2916, pages 936–942, AAAI Press, 2016.

[10] Christoph Benzmüller, Leon Weber, and Bruno Woltzenlogel Paleo. Computer-Assisted Analysis of the Anderson-Hájek Controversy. *Logica Universalis*, 11(1):139–151, 2017.

Evidence Extraction from Parameterised Boolean Equation Systems

Wieger Wesselink and Tim A.C. Willemse

Eindhoven University of Technology, Eindhoven, The Netherlands
{j.w.wesselink,t.a.c.willemse}@tue.nl

Abstract

Model checking is a technique for automatically assessing the quality of software and hardware systems and designs. Given a formalisation of both the system behaviour and the requirements the system should meet, a model checker returns either a *yes* or a *no*. In case the answer is not as expected, it is desirable to provide feedback to the user as to why this is the case. Providing such feedback, however, is not straightforward if the requirement is expressed in a highly expressive logic such as the modal μ-calculus, and when the decision problem is solved using intermediate formalisms. In this paper, we show how to extract witnesses and counterexamples from parameterised Boolean equation systems encoding the model checking problem for the first-order modal μ-calculus. We have implemented our technique in the modelling and analysis toolset mCRL2 and showcase our approach on a few illustrative examples.

1 Introduction

The complexity of the average computer-controlled system has reached a point at which it has become impossible to fully understand a system. By modelling a system and subsequently analysing whether the crucial safety and liveness requirements of the system are upheld, some confidence in the system's correctness can be obtained. The complexity of the average system, however, precludes that such an analysis can be conducted manually.

Model checking is an automated technique for assessing whether a requirement holds for a model of a system. This technique requires as input a mathematical description of the behaviour of a system, often given in terms of (a high-level description of) a *Kripke Structure* or *Labelled Transition System*, and a logical formula, often given in some appropriate temporal logic. By feeding both artefacts to a tool, colloquially referred to as the *model checker*, the *yes* or *no* verdict produced by the tool states whether (the model of) the system meets the requirement. Knowing that a system fails to meet a requirement, however, does not help to improve on the system design. For that, richer feedback in the form of evidence (*i.e.* a counterexample or a witness) of the model checker is required.

Depending on the logic that is required to perform the verification, however, it is not always clear what type of evidence must be extracted from a negative model checking exercise. While for linear time logic (LTL), a lasso, or a prefix of a lasso typically suffices, the problem becomes more pronounced for branching time logics such as CTL, CTL* or the modal μ-calculus. The reason for this is that the formulae over such logics are essentially interpreted over infinite computation trees.

In this paper, we describe how evidence can be constructed for an extension of the modal μ-calculus, *viz.* the *first-order modal μ-calculus* [12], within the context of the analysis toolset mCRL2 [5]. This logic extends the standard modal μ-calculus by adding first-order quantification and parameterised fix-points. We draw inspiration from previous work [7] explaining how, in theory, evidence can be extracted from decision problems encoded in the logic of *Least Fixed Point* (LFP). The main idea is that a counterexample or witness for the model checking problem is a 'submodel' of the original model, which can be used to reconstruct the proof of the original negative model checking result.

One obstacle in applying the techniques outlined for LFP is that the model checker in mCRL2 uses *parameterised Boolean equation systems* (PBESs) [13] to solve the model checking problem. While LFP and PBESs have much in common, they differ in exactly those aspects used for evidence extraction. Our *first* contribution is therefore to show how this problem can effectively be overcome without changing the underlying theory for PBESs. Our *second* contribution is an implementation of our solution, illustrating the feasibility and appeal of the approach. As far as we are aware, ours is the first work demonstrating the feasibility of constructing diagnostics in this way for the full (first-order) modal μ-calculus with arbitrary alternation.

Related Work. We refer to Busard's PhD thesis [3] for a thorough overview of literature on generating diagnostics for model checking for the modal μ-calculus, but also other logics such as CTL and epistemic logics; we here give a concise overview of the most relevant related works. There are several works that address the problem of constructing diagnostics for the modal μ-calculus model checking problem. In [20], diagnostics is presented as a *game* played on the model checking game for the modal μ-calculus. A related approach is given by [14] who essentially suggests to generate *tableaux* as witnesses to the model checking problem. In [17], diagnostics are defined as explanations for the truth values of the underlying Boolean equation system that is used to solve a model checking problem for the alternation-free fragment of the modal μ-calculus. This technique is closely related to [20]. In contrast to our work, these approaches require the user conducting the verification to understand the underlying mechanism for conducting the verification. Finally, in the work by Tan and Cleaveland [21], which can be seen as a generalisation of [17], evidence is presented as information extracted from *(decorated) support sets*. These are closely related to the proof graphs underlying [6] and the idea of using decorations is reminiscent of the idea underlying [7] to allow first-order relations in proof graphs.

Outline. We give a cursory overview of the necessary background in Section 2; *i.e.* we briefly address the underlying theory for modelling data and system behaviours, the first-order modal μ-calculus, and we formalise the model checking problem. In Section 3, we introduce PBESs, we explain how the model checking problem can be converted to the problem of solving a PBES, and we illustrate the problem of extracting evidence from a PBES encoding a model checking problem. In Section 4, we illustrate how we can extract evidence from a PBES by modifying the encoding of the model checking problem and in Section 5, we illustrate how our solution works on practical examples. We conclude in Section 6.

2 Preliminaries

Our work is set in a context in which we rely on abstract data types to describe (and reason about) data. That is, we assume a given algebraic specification $\mathscr{S} = \langle S, \Sigma, E\rangle$ where S is a set of *sorts*, Σ is a family of *operation(s)* and E is a family of *equations*. As a convention, we write data sorts using letters D, E, *etcetera*. We have a set $\mathscr{D}$ of *data variables*, with typical elements $d, d_1, \ldots$

The semantics of an algebraic specification $\mathscr{S} = \langle S, \Sigma, E\rangle$ is given by a many-sorted Σ-algebra consisting of data domains corresponding to S and operations corresponding to Σ, satisfying the identities of E. We here adopt an initial algebra semantics point of view. For every sort $D, E, \ldots$, we denote the domains they represent by $\mathbb{D}, \mathbb{E}, \ldots$. For a closed term t of a given sort, say D (denoted $t{:}D$), we assume an interpretation function $[\![t]\!]$ that maps t to a value of $\mathbb{D}$ it represents. For open terms we use a *data environment* ε that maps each variable from $\mathscr{D}$ to a value of the proper domain. If we wish to indicate which variables may occur freely within a term t, we add these as 'parameters' to t; *i.e.* we write $t(d)$ to indicate that d is the only free variable in t. The interpretation of an open term t, denoted as $[\![t]\!]\varepsilon$ is obtained in the standard way. We write $\varepsilon[d \mapsto v]$ for the environment that maps variable d to value v and all other variables d' are mapped to $\varepsilon(d')$.

We require the presence of a sort B representing the Booleans $\mathbb{B} = \{true, false\}$. For convenience we also assume the existence of the sort N representing the natural numbers $\mathbb{N}$. For both sorts we assume the usual operators are available and we do not write constants or operators in the syntactic domain any different from their semantic counterparts. For example, the Boolean value *true* is represented by the constant *true* and the value *false* is represented by the constant *false*. The syntactic operator $_\wedge_{:}B \times B \to B$ corresponds to the usual, semantic conjunction $_\wedge_{:}\mathbb{B} \times \mathbb{B} \to \mathbb{B}$; *etcetera*.

Note that for readability, and without loss of generality, we use a single—possibly compound—data type in our definitions and formal statements.

2.1 Processes

The behaviours of software and hardware systems can be adequately modelled using labelled transition systems (LTSs). Modelling languages, such as process algebras and I/O-automata, provide language constructs such as conditional choice, interleaving parallelism and sequential composition through which one can compactly specify such systems. A useful normal form to which many such specifications can be compiled automatically is the *Linear process equation* (LPE) format, see *e.g.* [11].

An LPE typically models the (global) state of a (software or hardware) system by means of a finite vector of formal data parameters, ranging over adequately chosen sorts. The behaviours are described by a non-deterministic choice (denoted by the $+$ operator) among rules taken from a finite set of *condition-action-effect* rules, prescribing under which condition an action (modelling a message exchange or an event) is enabled, leading to an update of the vector of formal parameters. A formal definition of an LPE, employing syntax that stays true to its process-algebraic heritage, is given below.

Definition 1. A *linear process equation* is an equation of the following form:

$$L(d_L{:}D_L) = +\{\sum_{e_{\mathtt{a}}:E_{\mathtt{a}}} c_{\mathtt{a}}(d_L, e_{\mathtt{a}}) \to \mathtt{a}(f_{\mathtt{a}}(d_L, e_{\mathtt{a}})) \cdot L(g_{\mathtt{a}}(d_L, e_{\mathtt{a}})) \mid \mathtt{a} \in \mathscr{A}ct\}$$

The sort D_L is used to represent the set of states and variable d_L represents a state; $\mathtt{a}$ is a (typed) *action label* taken from a finite set $\mathscr{A}ct$ of (typed) action labels. Each action label $\mathtt{a}$ is associated with a *local variable* $e_{\mathtt{a}}$ of sort $E_{\mathtt{a}}$, an expression $c_{\mathtt{a}}{:}B$, which acts as a *condition*, an expression $f_{\mathtt{a}}{:}D_{\mathtt{a}}$, which yields an argument emitted along $\mathtt{a}$ when executed, and an operation $g_{\mathtt{a}}{:}D_L$, which yields a new state. We require that d_L and e_L are the only free variables occurring in these expressions.

The *interpretation* of L with an *initial state* represented by closed term $e{:}D_L$, denoted by $L(e)$, is a labelled transition system $M = \langle S, A, \to, s_0\rangle$, where:

- $S = \mathbb{D}_L$ with initial state $s_0 = [\![e]\!] \in S$;
- $A = \{\mathtt{a}(v_{\mathtt{a}}) \mid \mathtt{a} \in \mathscr{A}ct, v_{\mathtt{a}} \in \mathbb{D}_{\mathtt{a}}\}$ is the (possibly infinite) set of *actions*;
- $\to \subseteq S \times A \times S$ is the set of transitions, where $v \xrightarrow{\mathtt{a}(v_{\mathtt{a}})} w$ holds if and only if for some $u \in \mathbb{E}_{\mathtt{a}}$ and environment ε:
  - $[\![f_{\mathtt{a}}(d_L, e_{\mathtt{a}})]\!]\varepsilon[d_L \mapsto v, e_{\mathtt{a}} \mapsto u] = v_{\mathtt{a}}$ and $[\![g_{\mathtt{a}}(d_L, e_{\mathtt{a}})]\!]\varepsilon[d_L \mapsto v, e_{\mathtt{a}} \mapsto u] = w$,
  - $[\![c_{\mathtt{a}}(d_L, e_{\mathtt{a}})]\!]\varepsilon[d_L \mapsto v, e_{\mathtt{a}} \mapsto u]$ evaluates to true.

Throughout this paper, whenever we refer to an LPE L we implicitly mean the LPE as given by Def. 1, with d_L being the state parameter of sort D_L of process L. Note that an LPE L compactly expresses that in a state, represented by parameter d_L, whenever condition $c_{\mathtt{a}}(d_L, d_{\mathtt{a}})$ holds (for some non-deterministically chosen value for variable $d_{\mathtt{a}}$), then action $\mathtt{a}$ carrying data parameter $f_{\mathtt{a}}(d_L, d_{\mathtt{a}})$ can be executed, effectively changing the global state to $g_{\mathtt{a}}(d_L, d_{\mathtt{a}})$.

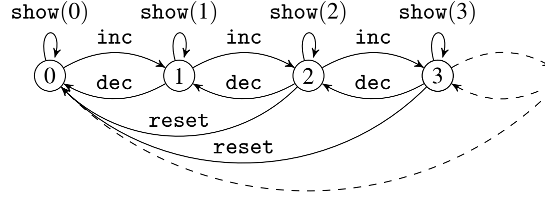
Notation 1. *In our examples we permit ourselves to be less strict in following the LPE format and allow for multiple summands ranging over the same action label. Moreover, in case a summand binds a local*

variable $d_{\mathtt{a}}$ which does not occur in the expressions $c_{\mathtt{a}}, f_{\mathtt{a}}$ and $g_{\mathtt{a}}$, we omit the $\sum$-symbol altogether. In our examples, action labels can carry zero or more arguments. Whenever condition $c_{\mathtt{a}}$ is the constant true, we omit both the condition and the $\rightarrow$ symbol.

Example 1. As a running example, we consider a small system modelling a simple counter that can increase and decrease a parameter n, reset that parameter to 0 when it has a positive value, and show the current value. The system is described by $L(0)$, where LPE L is given below.

$$\begin{array}{rcl} L(n{:}N) & = & \mathtt{inc} \cdot L(n+1) \\ & + & (n>0) \rightarrow \mathtt{dec} \cdot L(n-1) \\ & + & (n>1) \rightarrow \mathtt{reset} \cdot L(0) \\ & + & \mathtt{show}(n) \cdot L(n) \end{array}$$

Parts of the (infinite) labelled transition system associated to the LPE are depicted below; the dashed edges indicate the presence of one (or more) edges.



□

2.2 First-Order Modal μ-Calculus

Model checking is concerned with checking whether a modal property holds for a given system or not. The *first-order modal μ-calculus* (μ-calculus for short) of [18, 12] is a highly-expressive language for stating such properties. This logic is based on the standard modal μ-calculus [2], extended with first-order quantification and parameterised fixpoints, adding data as a first-class citizen. We briefly review its syntax and semantics, and we demonstrate its use by means of several small examples.

Definition 2. The μ-calculus, ranging over a set of (typed) action labels $\mathscr{A}ct$ is given by the following BNF grammar; formula φ represents a *state formula* and formula α represents an *action formula*:

$$\begin{array}{rcl} \varphi & ::= & b \mid Z(e) \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid [\alpha]\varphi_1 \mid \forall d{:}D.\ \varphi \mid \nu Z(d{:}D=e).\ \varphi \\ \alpha & ::= & \mathtt{a}(e_{\mathtt{a}}) \mid b \mid \neg\alpha_1 \mid \alpha_1 \wedge \alpha_2 \mid \forall d{:}D.\ \alpha_1 \end{array}$$

b is an expression of sort B; e is a data expression of type D; $Z{:}D \rightarrow B$ is a fixpoint variable from a set of fixpoint variables $\mathscr{P}$; for simplicity, we assume all fixpoint variables range over the same sort D. Expressions of the form $(\nu Z(d{:}D=e).\ \varphi)$ are subject to the restriction that any free occurrence of Z in φ must be within the scope of an even number of negation symbols $\neg$. Finally, $\mathtt{a}$ is an action label from the set $\mathscr{A}ct$ and expression $e_{\mathtt{a}}$ of sort $D_{\mathtt{a}}$ is a parameter of $\mathtt{a}$.

For reference we include the semantics of a μ-calculus formula in Table 1. Note that the ordered set $\langle [\mathbb{D} \rightarrow 2^S], \sqsubseteq \rangle$ is a complete lattice, where $[\mathbb{D} \rightarrow 2^S]$ is the set of functions from $\mathbb{D}$ to subsets of S and $\sqsubseteq$ is defined as $f \sqsubseteq g$ iff for all $v \in \mathbb{D}$, we have $f(v) \subseteq g(v)$. Since the functionals $\Phi_{\rho\varepsilon}$ are monotonic over this lattice, the interpretation of fixpoint expressions is then justified [22].

In the remainder of this paper, we use the following standard abbreviations for μ-calculus formulae φ, action formulae α and (both μ-calculus formulae and action formulae) ψ.

$$\begin{array}{lcl} (\psi_1 \vee \psi_2) & = & \neg(\neg\psi_1 \wedge \neg\psi_2) \\ \langle\alpha\rangle\varphi & = & \neg[\alpha]\neg\varphi \\ \exists d{:}D.\psi & = & \neg\forall d{:}D.\neg\psi \\ \mu Z(d{:}D=e).\varphi & = & \neg\nu Z(d{:}D{=}e).\ \neg\varphi[Z:=\neg Z] \end{array}$$

Table 1: The interpretation of a μ-calculus formula φ and action formula α, denoted by $[\![\varphi]\!]\rho\varepsilon$ and $\|\alpha\|\varepsilon$, respectively, in the context of environments ε and ρ and an LTS $M = \langle S, A, \to, s_0 \rangle$.

$$
\begin{array}{lcl}
[\![b]\!]\rho\varepsilon & = & \begin{cases} S \text{ if } [\![b]\!]\varepsilon \text{ is true} \\ \emptyset \text{ otherwise} \end{cases} \\
[\![Z(e)]\!]\rho\varepsilon & = & \rho(Z)([\![e]\!]\varepsilon) \\
[\![\neg\varphi]\!]\rho\varepsilon & = & S \setminus [\![\varphi]\!]\rho\varepsilon \\
[\![\varphi_1 \wedge \varphi_2]\!]\rho\varepsilon & = & [\![\varphi_1]\!]\rho\varepsilon \cap [\![\varphi_2]\!]\rho\varepsilon \\
[\![[\alpha]\varphi]\!]\rho\varepsilon & = & \{w{\in}S \mid \forall w'{\in}S\ \forall a{\in}A\ (w \xrightarrow{a} w' \wedge a{\in}\|\alpha\|\varepsilon) \Rightarrow w'{\in}[\![\varphi]\!]\rho\varepsilon\} \\
[\![\forall d{:}D.\varphi]\!]\rho\varepsilon & = & \bigcap_{v\in\mathbb{D}} [\![\varphi]\!]\rho(\varepsilon[d \mapsto v]) \\
[\![\nu Z(d{:}D{=}e).\ \varphi]\!]\rho\varepsilon & = & (\nu\Phi_{\rho\varepsilon})([\![e]\!]\varepsilon),\ \textit{where}\ \Phi_{\rho\varepsilon}{:}(\mathbb{D}\to 2^S)\to(\mathbb{D}\to 2^S)\ \textit{is defined as:} \\
 & & \Phi_{\rho\varepsilon}(F) = \lambda v \in \mathbb{D}.[\![\varphi]\!](\rho[Z\mapsto F])(\varepsilon[d\mapsto v])\ \textit{for}\ F{:}\mathbb{D}\to 2^S \\
 & & \\
\|b\|\varepsilon & = & \begin{cases} A \text{ if } [\![b]\!]\varepsilon \text{ is true} \\ \emptyset \text{ otherwise} \end{cases} \\
\|\mathtt{a}(e_{\mathtt{a}})\|\varepsilon & = & \{\mathtt{a}([\![e_{\mathtt{a}}]\!]\varepsilon)\} \\
\|\neg\alpha\|\varepsilon & = & A \setminus \|\alpha\|\varepsilon \\
\|\alpha_1 \wedge \alpha_2\|\varepsilon & = & \|\alpha_1\|\varepsilon \cap \|\alpha_2\|\varepsilon \\
\|\forall d{:}D.\alpha\|\varepsilon & = & \bigcap v{\in}D\ \|\alpha\|\varepsilon[d\mapsto v]
\end{array}
$$

A μ-calculus formula (in the language enriched with the above abbreviations) is in *Positive Normal Form* (PNF) whenever negation only occurs at the lowest level and all bound variables are distinct. We only consider formulae in PNF. Note that this is no restriction as every μ-calculus formula can be converted to PNF using suitable α-renaming and logical rules such as *De Morgan*. Moreover, we only consider μ-calculus formulae that are *closed*: data variables d only occur in the scope of a quantifier or a fixpoint binding it, and each fixpoint variable Z only occurs in the scope of a fixpoint that binds it. Since the semantics of closed formulae is independent from the environments ε and ρ, we typically write $[\![\varphi]\!]$ rather than $[\![\varphi]\!]\rho\varepsilon$ for closed φ. A formula φ is *normalised* whenever none of its subformulae of the form $\sigma X(d{:}D = e).\ \psi$ contain unbound data variables. Every closed formula can be converted (in linear time) to an equivalent normalised formula, see *e.g.* [16].

We are mainly concerned with the problem of deciding (and explaining) whether a given LPE $L(e)$ meets a logical specification φ given by a μ-calculus formula; this is known as the *model checking problem*. That is, we wish to decide whether $[\![e]\!] \in [\![\varphi]\!]$; we write $L(e) \models \varphi$ to denote just this.

We finish this section with a few illustrative examples of the use of data in formulae and parameterisation of fixpoints.

Example 2. Consider the LPE modelling the counter. Below are some simple properties of the counter, expressed in the μ-calculus.

1. the counter is *deadlock-free*: $\nu X.([\mathit{true}]X \wedge \langle \mathit{true}\rangle \mathit{true})$;
2. the counter can be incremented *ad infinitum*: $\nu X.\langle \mathtt{inc}\rangle X$;
3. the counter can alternatingly increase and decrease *ad infinitum*: $\nu X.\langle \mathtt{inc}\rangle\langle \mathtt{dec}\rangle X$;
4. the counter can be decreased *infinitely often*: $\nu X.\mu Y.(\langle \mathtt{dec}\rangle X \vee \langle \mathtt{inc}\rangle Y)$;
5. the counter can take on any natural number: $\forall n{:}N.\mu X.(\langle \mathtt{show}(n)\rangle \mathit{true} \vee \langle \mathit{true}\rangle X)$.

Table 2: The semantics $[\![\varphi]\!]\eta\varepsilon$ of a predicate formula φ is a truth assignment given in the context of a data environment ε and a *predicate* environment $\eta{:}\mathscr{X} \to (\mathbb{D}_{\mathscr{X}} \to \mathbb{B})$.

$$
\begin{array}{lcl}
[\![b]\!]\eta\varepsilon & = & [\![b]\!]\varepsilon \\
[\![X(e)]\!]\eta\varepsilon & = & \eta(X)([\![e]\!]\varepsilon) \\
[\![\varphi_1 \vee \varphi_2]\!]\eta\varepsilon & = & [\![\varphi_1]\!]\eta\varepsilon \text{ or } [\![\varphi_2]\!]\eta\varepsilon \\
[\![\varphi_1 \wedge \varphi_2]\!]\eta\varepsilon & = & [\![\varphi_1]\!]\eta\varepsilon \text{ and } [\![\varphi_2]\!]\eta\varepsilon \\
[\![\exists d{:}D.\varphi]\!]\eta\varepsilon & = & [\![\varphi]\!]\eta(\varepsilon[d \mapsto v]) \text{ for some } v \in \mathbb{D} \\
[\![\forall d{:}D.\varphi]\!]\eta\varepsilon & = & [\![\varphi]\!]\eta(\varepsilon[d \mapsto v]) \text{ for all } v \in \mathbb{D}
\end{array}
$$

6. on all paths, the counter can decrease as often as it has increased:
   $\nu X(m{:}N=0).([\mathtt{inc}]X(m+1) \wedge [\mathtt{dec}]X(m-1) \wedge [\neg\mathtt{inc} \wedge \neg\mathtt{dec}]X(m) \wedge (m=0 \vee \langle\mathtt{dec}\rangle \mathit{true}))$;

Note that all of the above properties hold for the initial state of the counter except for the last one: due to the reset action that can take place at any moment, the system may return to the initial state in which it can no longer perform a `dec` action. □

3 Model Checking using Parameterised Boolean Equation Systems

Solving the first-order modal μ-calculus model checking problem can be done in various ways. We here focus on the use of an intermediate formalism called *parameterised Boolean equation systems* [13]. These equation systems underlie several verification toolsets for specifying and analysing software and hardware systems, such as the CADP and mCRL2 toolsets. The advantage of using an intermediate formalism is that it allows for building dedicated techniques for that formalism [13, 12].

Parameterised Boolean Equation Systems are sequences of fixpoint equations where each equation is of the form $\nu X(d{:}D) = \varphi$ or $\mu X(d{:}D) = \varphi$. A parameterised Boolean equation is, in a sense, a simplified and equational variant of a first-order μ-calculus formula, lacking modal operators. We refer to the left-hand side variable of a parameterised Boolean equation as a *predicate variable*, whereas the right-hand side formula is called a *predicate formulae*.

Definition 3. A *parameterised Boolean equation system* $\mathscr{E}$ is a system of equations defined by:

$$
\begin{array}{rl}
\mathscr{E} ::= & \emptyset \mid (\mu X(d_X{:}D_{\mathscr{X}}) = \varphi)\ \mathscr{E} \mid (\nu X(d_X{:}D_{\mathscr{X}}) = \varphi)\ \mathscr{E} \\
\varphi ::= & b \mid X(e) \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \mid \exists d{:}D.\varphi \mid \forall d{:}D.\varphi
\end{array}
$$

b is an expression of sort B, X is a predicate variable taken from some sufficiently large set of typed predicate variables $\mathscr{X}$, d_X is a data parameter of sort $D_{\mathscr{X}}$, d is a data variable of sort D and e is a data expression of the appropriate sort. Again for simplicity, we assume that all predicate variables range over the same sort $D_{\mathscr{X}}$.

We only consider *well-formed, closed* PBESs in this paper. A PBES $\mathscr{E}$ is said to be *well-formed* iff every predicate variable $X \in \mathsf{bnd}(\mathscr{E})$, where $\mathsf{bnd}(\mathscr{E})$ is the set of *bound* variables (those predicate variables occurring at the left-hand side of the equations), occurs at the left-hand side of precisely one equation of $\mathscr{E}$. A PBES $\mathscr{E}$ is said to be *closed* iff (1) for each right-hand side predicate formula φ occurring in $\mathscr{E}$ we have $\mathsf{occ}(\varphi) \subseteq \mathsf{bnd}(\mathscr{E})$, where $\mathsf{occ}(\varphi)$ contains all predicate variables occurring in

φ, and, (2) whenever the only free data variable occurring in φ is the data parameter occurring at the left-hand side of φ's equation.

The interpretation of predicate formulae is listed in Table 2; for the fixpoint semantics of a PBES we refer to *e.g.* [13]. Instead of the fixpoint semantics we here focus on the equivalent *proof graph* semantics provided in [6]. This semantics is both more accessible (operational) and better suits our needs of computing counterexamples and witnesses. The proof graph semantics of a PBES explains the *value*, or *truth assignment* for each bound predicate variable by means of a directed graph with vertices ranging over the *signatures* of a PBES. A signature is a tuple (X,v) for $X \in \mathsf{bnd}(\mathscr{E})$ and $v \in \mathbb{D}_{\mathscr{X}}$ a value taken from the domain underlying the type of X; that is, we set $\mathsf{sig}(\mathscr{E}) = \{(X,v) \mid v \in \mathbb{D}_{\mathscr{X}}, X \in \mathsf{bnd}(\mathscr{E})\}$. We order each pair of variables $X,Y \in \mathsf{bnd}(\mathscr{E})$ as follows: $X \leq Y$ iff the equation for Y follows that of X and we write $X < Y$ iff $X \leq Y$ and $X \neq Y$. Moreover, we say that a variable X is a ν-variable whenever X occurs at the left-hand side of a greatest fixpoint equation; otherwise, X is a μ-variable.

Definition 4. A graph $\mathscr{G} = (V,E)$, for $V \subseteq \mathsf{sig}(\mathscr{E})$ and $E \subseteq V \times V$ is a *proof graph* iff:

1. for all equations $(\sigma X(d_X{:}D_{\mathscr{X}}) = \varphi)$ in $\mathscr{E}$ for which $(X,v) \in V$, $[\![\varphi]\!]\Theta_{(X,v)}\varepsilon[d_X \mapsto v]$ holds for some ε and where $\Theta_{(X,v)}$ is the environment defined as follows: $\Theta_{(X,v)}(Y) = \{w \in \mathbb{D}_{\mathscr{X}} \mid \langle (X,v),(Y,w)\rangle \in E\}$ for all Y;
2. for all infinite paths $(X_1,v_1)\ (X_2,v_2)\ \dots$ through $\mathscr{G}$, the smallest variable (w.r.t. the ordering $<$) occurring infinitely often on that path is a ν-variable.

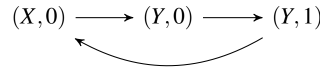
The semantics (often referred to as the (partial) *solution*) of a PBES is as follows; the correspondence with the more traditional fixpoint semantics follows from, *e.g.* [6].

Definition 5. Let $\mathscr{E}$ be a PBES. The semantics of $\mathscr{E}$ is a predicate environment $[\![\mathscr{E}]\!]$ defined as follows: $v \in [\![\mathscr{E}]\!](X)$ iff $X \in \mathsf{bnd}(\mathscr{E})$ and there is a proof graph $\mathscr{G} = (V,E)$ such that $(X,v) \in V$.

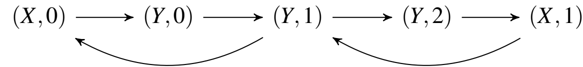
The *dual* of a proof graph is called a *refutation graph*. Such a graph explains that a value v does *not* belong to the set of values defined by some variable X. The notion of a refutation graph is defined analogously to a proof graph, using complementation for $\Theta_{(X,v)}$ in condition 1 and requiring that the least variable occurring on any infinite path in the graph is a μ-variable.

A proof graph containing a vertex (X,v) provides *evidence* that the value v belongs to the set of values defined by X in the PBES. In that case, the first condition essentially states that v belongs to X because all successors of (X,v) together yield an environment that makes the right-hand side formula for X hold when parameter d_X is assigned value v. The second condition ensures that the graph respects the *parity condition* typically associated with (nested) fixpoint formulae.

Example 3. Consider the PBES $(\nu X(n{:}N) = Y(n))\ (\mu Y(n{:}N) = (n > 0 \wedge X(n-1)) \vee Y(n+1))$. A proof graph for this PBES is given below:



Note that there are an infinite number of proof graphs containing $(X,0)$. An alternative proof graph is, *e.g.* the following:



From the first proof graph it follows that $\{0\} \subseteq [\![\mathscr{E}]\!](X)$, whereas from the second proof graph it follows that $\{0,1\} \subseteq [\![\mathscr{E}]\!](X)$ and $\{0,1,2\} \subseteq [\![\mathscr{E}]\!](Y)$. Note that it is straightforward to show, by extending the given proof graphs, that for any natural number $k \in \mathbb{N}$, we have $k \in [\![\mathscr{E}]\!](X)$ and $k \in [\![\mathscr{E}]\!](Y)$. $\square$

A proof graph is *minimal* if leaving out vertices or edges yields a graph that violates the proof graph conditions. We here note that the problem of computing a minimal proof graph (*i.e.* a subgraph of a proof graph that is as small as possible), is NP hard [6]. However, some techniques for computing a proof graph, such as solving a parity game induced by a PBES and using the winning strategies to filter unreachable vertices, yield minimal proof graphs by definition. Note that minimality of a proof graph does not imply the non-existence of a proof graph that is strictly smaller, see the example below.

Example 4. Consider the two proof graphs of Example 3. The first graph is a *minimal* proof graph and it is also the *smallest possible* proof graph. The second proof graph is clearly not minimal; however, by omitting the edge from $(Y,1)$ to $(X,0)$ we obtain another minimal proof graph. The thus obtained proof graph is clearly not the smallest possible proof graph. □

There are numerous ways in which a PBES $\mathcal{E}$ can be partially solved, *i.e.* for which we can decide, for a given bound predicate variable $X \in \mathsf{bnd}(\mathcal{E})$ and value v, whether $v \in [\![\mathcal{E}]\!](X)$. For instance, one can use *Gauß Elimination and symbolic approximation* [12], or by constructing and solving a parity game that is induced by a PBES [18].

As we remarked at the start of this section, the usefulness of PBESs lies in the observation that there is a linear-time reduction of the first-order modal μ-calculus model checking problem for LPEs to PBESs, see *e.g.* [12]. This transformation generalises the reduction of the (standard) modal μ-calculus for LTSs to *Boolean equation systems* [15]. For reference, we provide the details for this reduction in Table 3; in the next section, we present our modifications to this transformation. The correctness of the original transformation is given by the following theorem, taken from [12].

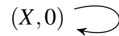
Theorem 1. Let formula $\sigma X(d{:}D = e').\ \psi$ be a closed, normalised first-order modal μ-calculus formula and $L(e)$ be an LPE. Then $L(e) \models \sigma X(d{:}D = e').\ \psi$ iff $([\![e]\!],[\![e']\!]) \in [\![\mathbf{E}_L(\sigma X(d{:}D = e').\ \psi)]\!](X)$. □

Example 5. We illustrate the transformation on the counter modelled by LPE $L(0)$ of Example 1. Say that we wish to analyse whether the counter can be decreased infinitely often, *i.e.* property (4) of Example 2: $\nu X.\mu Y.(\langle \mathtt{dec}\rangle X \vee \langle \mathtt{inc}\rangle Y)$. The PBES resulting from the transformation (after logical simplification, eliminating all subformulae obtained from summands not matching the action labels in the modal operators) is the PBES of Example 3. Note that, per Theorem 1, the proof graphs given there illustrate that the property holds for $L(0)$.

Next, suppose we wish to verify the property that we can infinitely often alternatingly increase and decrease parameter n, *i.e.* property (3) of Example 2. Recall that this is formalised by the following μ-calculus formula $\nu X.\langle \mathtt{inc}\rangle\langle \mathtt{dec}\rangle X$. The PBES $\mathcal{E}$ we obtain from this property (again after some logical simplification) is as follows:

$$\nu X(n{:}N) = n+1 > 0 \wedge X((n+1)-1)$$

It follows, by definition, that $0 \in [\![\mathcal{E}]\!](X)$, see the proof graph given below.



Consequently, by Theorem 1 we find that also $L(0) \models \nu X.\langle \mathtt{inc}\rangle\langle \mathtt{dec}\rangle X$. □

In [6], proof graphs and refutation graphs for PBESs were introduced in an effort to provide a meaningful explanation of the answer to a decision problem encoded in a PBES. While at the level of a PBES these graphs indeed meet their objective, as they provide the exact reasoning underlying the (partial) solution of a PBES, they do not aid in understanding the solution at the level of the decision problem that is encoded in the PBES. This is clearly illustrated in, *e.g.* the last proof graph for the PBES underlying the model checking problem of Example 5: the single vertex with a self-loop, constituting the proof graph, does not explain which transitions of the LTS of Example 5 are involved.

Table 3: Translation scheme for encoding the problem $L \models \sigma X(d{:}D_{\mathscr{X}} = e').\ \psi$ into an equation system. Recall that parameter d_L of sort D_L, occurring in the rules, originates from LPE L of Def. 1; likewise, the expressions c_{a}, f_{a} and g_{a} originate from this LPE.

$$
\begin{array}{lcll}
\mathbf{E}_L(b) & = & \varepsilon & \\
\mathbf{E}_L(X(e)) & = & \varepsilon & \\
\mathbf{E}_L(\varphi \oplus \psi) & = & \mathbf{E}_L(\varphi)\ \mathbf{E}_L(\psi) & \text{for } \oplus \in \{\wedge, \vee\} \\
\mathbf{E}_L(\mathsf{Q}\, d{:}D.\varphi) & = & \mathbf{E}_L(\varphi) & \text{for } \mathsf{Q} \in \{\forall, \exists\} \\
\mathbf{E}_L([\alpha]\varphi) & = & \mathbf{E}_L(\varphi) & \\
\mathbf{E}_L(\langle\alpha\rangle\varphi) & = & \mathbf{E}_L(\varphi) & \\
\mathbf{E}_L(\sigma X(d{:}D_{\mathscr{X}} = e).\ \psi) & = & (\sigma X(d_L{:}D_L, d{:}D_{\mathscr{X}}) = \mathbf{RHS}_L(\psi))\ \mathbf{E}_L(\psi) & \\
\mathbf{RHS}_L(b) & = & b & \\
\mathbf{RHS}_L(X(e)) & = & X(d_L, e) & \\
\mathbf{RHS}_L(\varphi \oplus \psi) & = & \mathbf{RHS}_L(\varphi) \oplus \mathbf{RHS}_L(\psi) & \text{for } \oplus \in \{\wedge, \vee\} \\
\mathbf{RHS}_L(\mathsf{Q}\, d{:}D.\varphi) & = & \mathsf{Q}\, d{:}D.\ \mathbf{RHS}_L(\varphi) & \text{for } \mathsf{Q} \in \{\forall, \exists\} \\
\mathbf{RHS}_L([\alpha]\varphi) & = & \bigwedge_{\mathsf{a} \in \mathscr{A}ct} \forall e_{\mathsf{a}}{:}D_{\mathsf{a}}.\ (c_{\mathsf{a}}(d_L, e_{\mathsf{a}}) \wedge \mathsf{match}(\mathsf{a}(f_{\mathsf{a}}(d_L, e_{\mathsf{a}})), \alpha)) & \\
 & & \quad \implies (\mathbf{RHS}_L(\varphi)[g_{\mathsf{a}}(d_L, e_{\mathsf{a}})/d_L]) & \\
\mathbf{RHS}_L(\langle\alpha\rangle\varphi) & = & \bigvee_{\mathsf{a} \in \mathscr{A}ct} \exists e_{\mathsf{a}}{:}D_{\mathsf{a}}.\ (c_{\mathsf{a}}(d_L, e_{\mathsf{a}}) \wedge \mathsf{match}(\mathsf{a}(f_{\mathsf{a}}(d_L, e_{\mathsf{a}})), \alpha) & \\
 & & \quad \wedge (\mathbf{RHS}_L(\varphi)[g_{\mathsf{a}}(d_L, e_{\mathsf{a}})/d_L])) & \\
\mathbf{RHS}_L(\sigma X(d{:}D_{\mathscr{X}} = e).\ \varphi) & = & X(d_L, e) & \\
\mathsf{match}(\mathsf{a}(v), b) & = & b & \\
\mathsf{match}(\mathsf{a}(v), \mathsf{a}'(e')) & = & (v = e') \wedge (\mathsf{a} = \mathsf{a}') & \\
\mathsf{match}(\mathsf{a}(v), \neg\alpha) & = & \neg\mathsf{match}(\mathsf{a}(v), \alpha) & \\
\mathsf{match}(\mathsf{a}(v), \alpha \wedge \beta) & = & \mathsf{match}(\mathsf{a}(v), \alpha) \wedge \mathsf{match}(\mathsf{a}(v), \beta) & \\
\mathsf{match}(\mathsf{a}(v), \forall d{:}D.\ \alpha) & = & \forall d{:}D.\ \mathsf{match}(\mathsf{a}(v), \alpha) &
\end{array}
$$

4 Evidence Extraction from PBESs

Whenever a model checking problem yields an unexpected verdict, evidence in the form of a *witness* or *counterexample* supporting that verdict can be helpful in analysing the root cause. Following [7], such a witness or counterexample is a fragment of the original labelled transition system (or LPE) that can be used to reconstruct the model checking verdict. However, a proof graph underlying a PBES that encodes a model checking problem lacks some essential information about the LPE to extract a counterexample or witness, see Example 5. This issue was recognised and further studied in [7] in a slightly different setting, *viz.* in the setting of the logic *Least Fixed Point* (LFP).

The solution proposed in [7] is to extend the proof graphs with information about *first-order* relation symbols from the structures involved in the encoded decision problem. Using proof graphs enriched in this way, evidence (*e.g.* counterexamples or witnesses) can be extracted from the proof graphs by projecting onto the vertices referring to the relational symbols of the desired structure(s). The thus obtained structures are weak substructures of the original structures that allow for reconstructing the proof graph underlying the original PBES.

Porting the proposed solution to the setting to PBESs is, however, not straightforward. The reason for this is that, unlike in LFP formulae, one cannot refer to first-order relational symbols; one can essentially only refer to Boolean expressions and predicate variables. Consequently, proof graphs for PBESs cannot be enriched in a similar fashion.

We propose to solve this problem by adding the information from the structures, relevant for constructing proper diagnostics, to the encoding of the decision problem. For instance, if the diagnostics

for a model checking problem requires that a weak substructure of the original structures can be reconstructed, then we add information about that structure to our encoding. The extra information is added in the form of additional equations and by adding predicate variables that refer to those equations. For the model checking problem, this only requires changing the rules for $\mathbf{RHS}_L([\alpha]\varphi)$ and $\mathbf{RHS}_L(\langle\alpha\rangle\varphi)$:

$$\begin{array}{rcl}\mathbf{RHS}_L([\alpha]\varphi) & = & \bigwedge_{\mathsf{a}\in\mathscr{A}ct} \forall e_\mathsf{a}{:}E_\mathsf{a}.\ (c_\mathsf{a}(d_L,e_\mathsf{a}) \wedge \mathsf{match}(\mathsf{a}(f_\mathsf{a}(d_L,e_\mathsf{a})),\alpha)) \\ & & \quad \implies \Big(\big(\mathbf{RHS}_L(\varphi)[g_\mathsf{a}(d_L,e_\mathsf{a})/d_L] \wedge L^+_\mathsf{a}(d_L,f_\mathsf{a}(d_L,e_\mathsf{a}),g_\mathsf{a}(d_L,e_\mathsf{a}))\big) \\ & & \quad\quad \vee L^-_\mathsf{a}(d_L,f_\mathsf{a}(d_L,e_\mathsf{a}),g_\mathsf{a}(d_L,e_\mathsf{a}))\Big) \\ \mathbf{RHS}_L(\langle\alpha\rangle\varphi) & = & \bigvee_{\mathsf{a}\in\mathscr{A}ct} \exists e_\mathsf{a}{:}E_\mathsf{a}.\ \Big(c_\mathsf{a}(d_L,e_\mathsf{a}) \wedge \mathsf{match}(\mathsf{a}(f_\mathsf{a}(d_L,e_\mathsf{a})),\alpha) \\ & & \quad \wedge \Big(\big(\mathbf{RHS}_L(\varphi)[g_\mathsf{a}(d_L,e_\mathsf{a})/d_L] \vee L^-_\mathsf{a}(d_L,f_\mathsf{a}(d_L,e_\mathsf{a}),g_\mathsf{a}(d_L,e_\mathsf{a}))\big) \\ & & \quad\quad \wedge L^+_\mathsf{a}(d_L,f_\mathsf{a}(d_L,e_\mathsf{a}),g_\mathsf{a}(d_L,e_\mathsf{a}))\Big)\Big)\end{array}$$

All remaining rules remain as before. The two additional equations that must be added to the translation for each action label a, are as follows:

$$(\nu L^+_\mathsf{a}(d_L{:}D_L,d_\mathsf{a}{:}D_\mathsf{a},d'_L{:}D_L) = \mathit{true})\ \ (\mu L^-_\mathsf{a}(d_L{:}D_L,d_\mathsf{a}{:}D_\mathsf{a},d'_L{:}D_L) = \mathit{false})$$

Note that their exact position in the resulting PBES does not matter (their solutions are independent of other equations as they are simple constants, so in a proof graph or refutation graph, vertices of the form $(L^+_\mathsf{a},v,v_\mathsf{a},w)$ or $(L^-_\mathsf{a},v,v_\mathsf{a},w)$ never need to be part of an infinite path); for simplicity, we add these equations at the end of the PBES. We denote the updated translation by $\mathbf{E}^c_L$.

Theorem 2. Let formula $\sigma X(d{:}D = e').\ \psi$ be a closed, normalised first-order modal μ-calculus formula and $L(e)$ be an LPE. Then $L(e) \models \sigma X(d{:}D = e').\ \psi$ iff $([\![e]\!],[\![e']\!]) \in [\![\mathbf{E}^c_L(\sigma X(d{:}D = e').\ \psi)]\!](X)$.

Proof. The correctness of the new transformation follows immediately from the fact that we can substitute 'solved' equations for their references, see [13]. That is, by replacing L^+_a by the constant *true* and replacing L^-_a by the constant *false* we can effectively reduce the new rules for translating $[\alpha]\varphi$ and $\langle\alpha\rangle\varphi$ to the ones from Table 3. □

Intuitively, the modification in the translation of the $\langle\alpha\rangle\varphi$ and $[\alpha]\varphi$ construction allows for extracting evidence from a proof graph because whenever the proof graph records information about which predicate variables are required to make φ hold true, also information about the *transition*, encoded by the predicate variable L^+_a must be recorded in the proof graph. This follows from the fact that, *e.g.* in the new translation for $\langle\alpha\rangle\varphi$, the expression L^+_a is added as a conjunct in the new translation. For similar reasons, whenever there is no α-matching transition leading to a state satisfying φ, the proof graph will contain *all* α-matching transitions, encoded by the predicate variable L^-_a.

Before we formally define how we can extract diagnostic information from a proof graph, we illustrate the basic idea by showing how to solve the problem we encountered in Example 5.

Example 6. Reconsider the second model checking problem of Example 5, *i.e.* the problem of deciding whether the counter system satisfies property $\nu X.\langle\mathtt{inc}\rangle\langle\mathtt{dec}\rangle X$. Whereas the standard transformation yields a PBES consisting of only one equation, we now obtain a PBES $\mathscr{E}$ with seven equations (two of

which are redundant):

$$
\begin{array}{l}
\Big(\nu X(n{:}N) \;=\; n+1>0 \wedge \\
\qquad\qquad \Big(\big(((X((n+1)-1) \wedge L^{+}_{\mathtt{dec}}(n+1,(n+1)-1)) \vee L^{-}_{\mathtt{dec}}(n+1,(n+1)-1)) \\
\qquad\qquad\qquad \wedge L^{+}_{\mathtt{inc}}(n,n+1) \\
\qquad\qquad \big) \vee L^{-}_{\mathtt{inc}}(n,n+1)\Big)\Big) \\
(\nu L^{+}_{\mathtt{inc}}(n{:}N,n'{:}N) \;=\; \mathit{true})\;(\nu L^{+}_{\mathtt{dec}}(n{:}N,n'{:}N) \;=\; \mathit{true})\;(\nu L^{+}_{\mathtt{reset}}(n{:}N,n'{:}N) \;=\; \mathit{true}) \\
(\mu L^{-}_{\mathtt{inc}}(n{:}N,n'{:}N) \;=\; \mathit{false})\;(\mu L^{-}_{\mathtt{dec}}(n{:}N,n'{:}N) \;=\; \mathit{false})\;(\mu L^{-}_{\mathtt{reset}}(n{:}N,n'{:}N) \;=\; \mathit{false})
\end{array}
$$

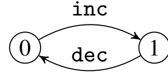
Note that the smaller proof graph of Example 5 we could use to demonstrate that $0 \in [\![\mathscr{E}]\!](X)$ now no longer is appropriate since it misses relevant information about the signatures for $L^{+}_{\mathtt{inc}}$ and $L^{+}_{\mathtt{dec}}$. Adding this extra information yields the following proof graph:

$$
(L^{+}_{\mathtt{dec}},1,0) \longleftarrow \overset{\circlearrowright}{(X,0)} \longrightarrow (L^{+}_{\mathtt{inc}},0,1)
$$

The signatures containing the $L^{+}_{\mathtt{inc}}$ and $L^{+}_{\mathtt{dec}}$ predicate variables encode information about the transitions in the LPE that were involved in constructing the proof that the property we verify holds. Following the basic ideas outlined in [7], we extract the relevant information by filtering the relevant vertices from the proof graph and construct an LPE. In our case, we can extract the following LPE L^{+}:

$$
\begin{array}{rcl}
L^{+}(n{:}N) & = & (n=0) \rightarrow \mathtt{inc} \cdot L^{+}(1) \\
 & + & (n=1) \rightarrow \mathtt{dec} \cdot L^{+}(0)
\end{array}
$$

The LTS induced by $L^{+}(0)$, providing diagnostics at the level of the system, is as follows:



Note that the LTS is a subgraph of the LTS underlying $L(0)$ (see the original LTS of Example 1), which, moreover, provides a compact and intuitive argument why the property holds. Moreover, the proof graph underlying the PBES encoding the same μ-calculus model checking problem on $L^{+}(0)$ is isomorphic to the one we provided above. □

We next formalise the notions of *witness* and *counterexample*.

Definition 6. Let L be an LPE and φ a μ-calculus formula. Let $\mathscr{G} = (V,E)$ be a finite, minimal proof graph proving $L(e) \models \varphi$. Let $W(\mathtt{a}) = \{(e_L, e_{\mathtt{a}}, e'_L) \mid (L^{+}, [\![e_L]\!], [\![e_{\mathtt{a}}]\!], [\![e'_L]\!]) \in V\}$. The *witness* extracted from $\mathscr{G}$ is the LPE L_w defined as follows:

$$
L_w(d_L{:}D_L) = +\{ \sum_{(e_L,e_{\mathtt{a}},e'_L):D_L \times D_{\mathtt{a}} \times D_L} (e_L,e_{\mathtt{a}},e'_L) \in W(\mathtt{a}) \rightarrow \mathtt{a}(e_{\mathtt{a}}) \cdot L_w(e'_L) \mid \mathtt{a} \in \mathscr{A}ct \,\}
$$

In a similar vein, a *counterexample* LPE L_c is obtained by filtering all L_c signatures from V in the refutation graph.

We note that the LTS underlying LPE L_w (and, likewise, the LTS underlying LPE L_c) is a substructure of the LTS underlying L; this follows from minimality of the proof graph from which these LPEs are extracted, plus the fact that the conditions under which the transitions are present are enforced in the translations of $\mathbf{RHS}_L([\alpha]\varphi)$ and $\mathbf{RHS}_L(\langle\alpha\rangle\varphi)$. The theorem below states that the witness extracted from the PBES indeed contains enough information to reconstruct the proof graph from which it was extracted; a dual result holds for counterexamples extracted from a refutation graph.

Theorem 3. Let L be an LPE and φ a μ-calculus formula. Let $\mathscr{G} = (V,E)$ be a finite, minimal proof graph proving $L(e) \models \varphi$, and let L_w be the witness LPE extracted from $\mathscr{G}$. Then any proof graph proving $L_w(e) \models \varphi$ is isomorphic to some proof graph proving $L(e) \models \varphi$.

Proof. This follows essentially from the observation that the LTS underlying the LPE L_w, extracted from $\mathscr{G}$, is a substructure of the LTS underlying L. The bijection that maps all vertices of the form (L_w^+, v) to (L^+, v) and all other vertices to their identities in the proof graph proving $L_w(e) \models \varphi$ then yields an isomorphic proof graph proving $L(e) \models \varphi$; see also Theorem 10 in [7]. □

5 Applications

We have implemented the modified transformation in the mCRL2 toolset in the existing tool `lps2pbes`. Moreover, we have developed a new tool, called `pbessolve`, which extracts evidence from a PBES that is obtained via the new transformation. The latter tool uses *instantiation* to a parity game (much like the technique outlined in [23, 18]), which is then solved using Zielonka's recursive algorithm [24, 10] and from which we extract a witness or counterexample as per Def. 6.

We illustrate the effectiveness of the techniques we outlined in this paper through three examples taken from the mCRL2 repository. The first example is a scheduling problem sometimes referred to as the *bridge-crossing puzzle*. Our second example concerns three communication protocols. The third example we consider is the *Storage Management System* [19] of the DIRAC Community Grid Solution for the LHCb experiment at CERN.

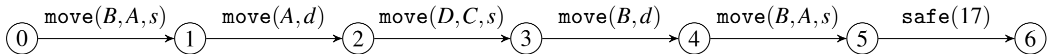
5.1 Bridge Crossing

The bridge-crossing puzzle centres around a scheduling problem with limited access to resources. The problem is essentially as follows. A family of four people (person A, B, C and D), chased by a pack of wolves, needs to cross a narrow bridge at night. The bridge can only hold two persons at a time and the damages to the bridge require the persons to carry a torch to avoid falling off the bridge. Unfortunately, there is only one torch; its battery is running out and can only last another 18 minutes. Persons A and B can cross the bridge in 1, resp. 2 minutes but person C requires 5 minutes and D even requires 10 minutes to cross the bridge. The problem is whether they have a strategy to safely cross the bridge before the battery of the torch runs out.

We model the puzzle as an LPE, where we model the state space by keeping track of the positions of the four persons (*i.e.* which side of the bridge they are), and the time that has passed since switching on the torch. We consider two types of move actions: $\mathtt{move}(p_1,l)$, modelling the person p_1 who is carrying a torch to move to location l (which can be s for safe, or d, for dangerous) and $\mathtt{move}(p_1,p_2,l)$, modelling the person p_1 who is carrying a torch, crossing the bridge together with p_2. Moreover, action $\mathtt{safe}(i)$ signals that the family managed to safely cross the bridge, taking i minutes; action `fail` indicates that the family did not manage to cross the bridge before the battery of the torch ran out. To verify whether the family has a strategy to safely cross the bridge, we verify the following formula:

$$\mu X.(\langle true\rangle X \vee \langle \exists i{:}N.\mathtt{safe}(i)\rangle true)$$

The witness proving the formula holds is depicted below, with 0 being the initial state. It shows a schedule by which the family can safely cross the bridge.



5.2 Communication Protocols

Communication protocols allow for exchanging information between devices through networks using strict rules and conventions. In general, the communication medium (*i.e.* the network) may not be perfectly reliable: it may re-order, scramble or even lose data that it transports. Part of the problem solved by a communication protocol is to disassemble and reassemble messages sent via such a network, working around the assumed characteristics of the network to achieve reliable information exchange.

A variety of communication protocols exist; we here consider the classic *Alternating Bit Protocol* (ABP), the *One Bit Sliding Window Protocol*, which is a simple bidirectional sliding window protocol with piggy backing and window sizes as the receiving and sending side of size 1 [1], and the full *Sliding Window Protocol* [9]. The property that we check for all three protocols is the following: there are no paths on which reading of a datum d is enabled infinitely often, but occurs only finitely often. Assuming that the data domain we range over is D and $\mathtt{read}(d)$ models reading datum d, this can be formalised as follows (see also, *e.g.* [2]):

$$\forall d{:}D.\nu X.\mu Y.\nu Z.([\mathtt{read}(d)]X \wedge ([\mathtt{read}(d)]\mathit{false} \vee [\neg\mathtt{read}(d)]Y) \wedge [\neg\mathtt{read}(d)]Z)$$

None of the three protocols satisfy the property for $|D| > 1$. The counterexamples we can extract all have a similar flavour but differ in the number of actions involved and the underlying reason for violating the property. We depict these counterexamples in Figure 1; we have omitted all other involved actions in these counterexamples.

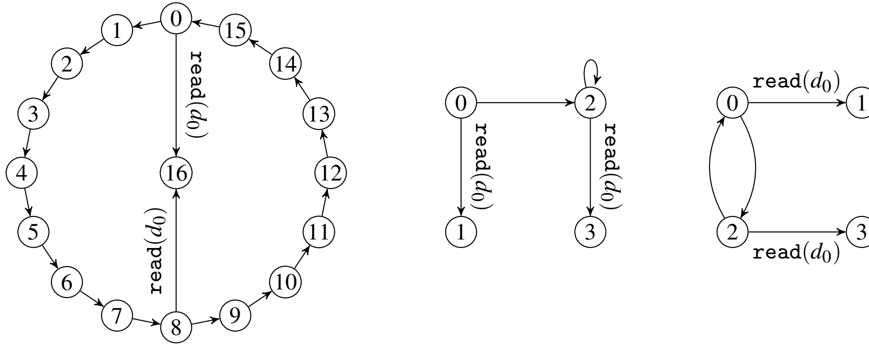


Figure 1: Counterexamples for the *if reading of a datum is infinitely often enabled then it occurs infinitely often* requirement. Left: counterexample for the ABP, middle: counterexample for the One Bit protocol, right: counterexample for the Sliding Window Protocol. In all cases, state 0 indicates the initial state; non-$\mathtt{read}(0)$ edge-labels have been omitted from our graphs.

All three counterexamples show the presence of an infinite path along which a $\mathtt{read}(d_0)$ is enabled, but never taken. Note that since this is a typical branching-time property with a strong fairness constraint, the counterexample cannot simply be represented by an infinite path represented by a lasso.

5.3 The DIRAC Storage Management System

DIRAC (Distributed Infrastructure with Remote Agent Control) is a grid solution that is designed to support both production activities as well as user data analysis for the Large Hadron Collider 'beauty' experiment. It consists of distributed services that cooperate with light-weight agents that deliver workload to the grid resources: services accept requests from running jobs and agents, whereas agents actively

work towards specific goals. The logic of each individual agent is fairly simple but the main source of complexity arises from their cooperation. Agents communicate using the services' databases as a shared memory for synchronising the state transitions. A formal model and analysis of two critical subsystems of DIRAC is described in [19]; one of these is the *Storage Management System* (SMS). In [19], it was shown that this system violated several requirements. Most of these requirements were safety requirements. A violation of such a requirement is a simple trace in the system, which is fairly easy to generate and visualise. A requirement that defied such a simple approach is the following liveness requirement:

$$\nu X.([true]X \wedge [\mathtt{state}([tStaged]) \vee \mathtt{state}([tFailed])]\nu Y.([\neg\mathtt{state}([tDeleted])]Y \wedge \mu Z.(\langle true\rangle Z \vee \langle \mathtt{state}([tDeleted])\rangle true)))$$

The requirement essentially states that each task that is in a terminating state (*i.e.* in state *tFailed* or *tStaged*) is eventually removed from the DIRAC system (*i.e.* *tDeleted*). The only action of concern is the $\mathtt{state}(s)$ action, which emits the current state s of a task. The counterexample to the property is depicted in Figure 2; the original LTS contains 142 states, which can be further reduced (without affecting the behaviours encoded by the LTS) to the depicted 26 states.

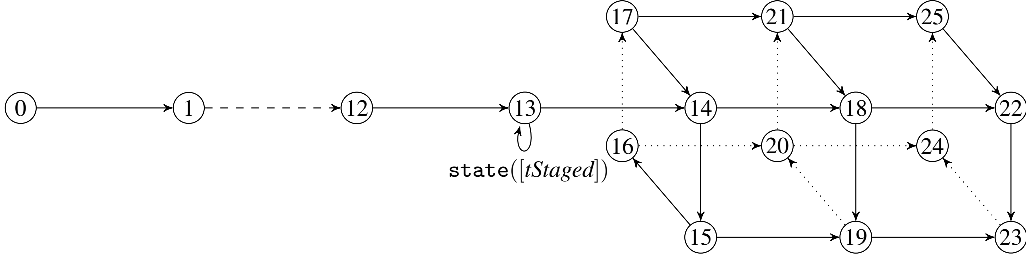


Figure 2: Counterexamples for the requirement that *each task in a terminating state is eventually removed* for the Storage Management Systems. State 0 indicates the initial state; we omitted all edge labels except for the trigger $\mathtt{state}([tStaged])$. The dashed line between state 1 and 12 indicates a single path through states 2, 3,..., 12; the dotted transitions are 3D artefacts.

The counterexample clearly indicates a path towards a part of the system where the task, once staged (in state 13), will never be removed from the system.

6 Conclusions and Future Work

We studied and implemented the extraction of useful evidence from parameterised Boolean equation systems (PBESs) encoding the model checking problem for the first-order modal μ-calculus. Our solution is inspired by the LFP-approach outlined (but not implemented) in [7]. Our solution, which we have also implementation and which is made available through the mCRL2 [5] toolset, shows the appeal of the technique even when used for explaining the failure for complex requirements to hold.

Apart from the model checking problem, PBESs can also be used for checking behavioural equivalence of processes [4]. Diagnostics for such problems are typically presented in the form of a game [8]. We expect to be able to apply the ideas outlined in the current paper to such problems as well, leading to evidence in the form of substructures for *both* input models, which, combined, explain the differences between both models. Implementing this problem and investigating whether such a solution would provide intelligible feedback to the user is left for future work.

References

[1] M. Bezem and J.F. Groote. A correctness proof of a one-bit sliding window protocol in μCRL. *Comput. J.*, 37(4):289–307, 1994.

[2] J. Bradfield and C. Stirling. Modal logics and mu-calculi. In *Handbook of Process Algebra*, pages 293–332. Elsevier, North-Holland, 2001.

[3] S. Busard. *Symbolic Model Checking of Multi-Modal Logics: Uniform Strategies and Rich Explanations*. PhD thesis, Université catholique de Louvain (UCL), 2017.

[4] T. Chen, B. Ploeger, J. van de Pol, and T.A.C. Willemse. Equivalence checking for infinite systems using parameterized Boolean equation systems. In *CONCUR*, volume 4703 of *LNCS*, pages 120–135. Springer, 2007.

[5] S. Cranen, J.F. Groote, J.J.A. Keiren, F.P.M. Stappers, E.P. de Vink, W. Wesselink, and T.A.C. Willemse. An overview of the mCRL2 toolset and its recent advances. In *TACAS*, volume 7795 of *LNCS*, pages 199–213. Springer, 2013.

[6] S. Cranen, B. Luttik, and T.A.C. Willemse. Proof graphs for parameterised Boolean equation systems. In *CONCUR*, volume 8052 of *LNCS*, pages 470–484. Springer, 2013.

[7] S. Cranen, B. Luttik, and T.A.C. Willemse. Evidence for fixpoint logic. In *CSL*, volume 41 of *LIPIcs*, pages 78–93. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2015.

[8] D. de Frutos-Escrig, J.J.A. Keiren, and T.A.C. Willemse. Games for bisimulations and abstraction. *Logical Methods in Computer Science*, 13(4), 2017.

[9] W. Fokkink, J.F. Groote, J. Pang, B. Badban, and J. van de Pol. Verifying a sliding window protocol in μCRL. In *AMAST*, volume 3116 of *LNCS*, pages 148–163. Springer, 2004.

[10] M. Gazda and T.A.C. Willemse. Zielonka's recursive algorithm: dull, weak and solitaire games and tighter bounds. In *GandALF*, volume 119 of *EPTCS*, pages 7–20, 2013.

[11] J.F. Groote and M.R. Mousavi. *Modeling and Analysis of Communicating Systems*. MIT Press, 2014.

[12] J.F. Groote and T.A.C. Willemse. Model-checking processes with data. *Sci. Comput. Program.*, 56(3):251–273, 2005.

[13] J.F. Groote and T.A.C. Willemse. Parameterised Boolean equation systems. *Theor. Comput. Sci.*, 343(3):332–369, 2005.

[14] A. Kick. Tableaux and witnesses for the μ-calculus, 1995. Tech. Rep. 44/95, University of Karlsruhe.

[15] A. Mader. *Verification of Modal Properties Using Boolean Equation Systems*. PhD thesis, Technische Universität München, 1997.

[16] R. Mateescu. *Vérification des propriétés temporelles des programmes parallèles*. PhD thesis, Institut National Polytechnique de Grenoble, 1998.

[17] R. Mateescu. Efficient diagnostic generation for Boolean equation systems. In *TACAS*, volume 1785 of *LNCS*, pages 251–265. Springer, 2000.

[18] B. Ploeger, W. Wesselink, and T.A.C. Willemse. Verification of reactive systems via instantiation of parameterised Boolean equation systems. *Inf. Comput.*, 209(4):637–663, 2011.

[19] D. Remenska, T.A.C. Willemse, K. Verstoep, J. Templon, and H.E. Bal. Using model checking to analyze the system behavior of the LHC production grid. *Future Generation Comp. Syst.*, 29(8):2239–2251, 2013.

[20] P. Stevens and C. Stirling. Practical model-checking using games. In *TACAS*, volume 1384 of *LNCS*, pages 85–101. Springer, 1998.

[21] L. Tan and R. Cleaveland. Evidence-based model checking. In *CAV*, volume 2404 of *LNCS*, pages 455–470. Springer, 2002.

[22] A. Tarski. A lattice-theoretical fixpoint theorem and its applications. *Pac. J. Math.*, 5(2):285–309, June 1955.

[23] A. van Dam, B. Ploeger, and T.A.C. Willemse. Instantiation for parameterised Boolean equation systems. In *ICTAC*, volume 5160 of *LNCS*, pages 440–454. Springer, 2008.

[24] W. Zielonka. Infinite games on finitely coloured graphs with applications to automata on infinite trees. *Theor. Comput. Sci.*, 200(1-2):135–183, 1998.