Christoph Benzmüller
Jens Otten (Eds.)

Automated Reasoning in Quantified Non-Classical Logics

1st International Workshop, ARQNL 2014,
Vienna, Austria, July 23, 2014

Proceedings

FLoC 2014
VIENNA SUMMER OF LOGIC

To appear in EasyChair Proceedings in Computing (EPiC) Series

Preface

This volume contains the proceedings of the First International Workshop on Automated Reasoning in Quantified Non-Classical Logics (ARQNL 2014), held July 23, 2014, in Vienna, Austria. The workshop is affiliated with the International Joint Conference on Automated Reasoning (IJCAR 2014), which is part of the Federated Logic Conference (FLoC 2014) and the Vienna Summer of Logic (VSL 2014). The aim of the ARQNL 2014 Workshop is to foster the development of proof calculi, automated theorem proving (ATP) systems and model finders for all sorts of quantified non-classical logics. The workshop 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, dynamic logic, fuzzy logic, paraconsistent logic, relevance 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 significant 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 workshop to initiate and foster practical implementations and evaluations of such ATP systems for non-classical logics.

The ARQNL 2014 Workshop received 10 paper submissions. Each paper was reviewed by at least three referees, and following an online discussion, 8 research papers were selected to be included in the proceedings. We would like to sincerely thank both all the authors for their submissions and the members of the Program Committee of ARQNL 2014 for their professional work in the review process. Furthermore, we would like to thank the Workshop Chairs and the Organizing Committee of the Vienna Summer of Logic. Finally, many thanks to all active participants at the ARQNL 2014 Workshop.

Berlin and Potsdam, July 2014

Christoph Benzmüller
Jens Otten

Organization

Program Committee

Carlos Areces	FaMAF - Universidad Nacional de Córdoba, Argentina
Christoph Benzmüller	Freie Universität Berlin, Germany – chair
Walter Carnielli	Centre for Logic, Epistemology and the History of Science, Brazil
Valeria De Paiva	Nuance Communications, USA
Christian Fermüller	TU Wien, Austria
Didier Galmiche	Université de Lorraine - LORIA, France
Rajeev Goré	The Australian National University, Australia
Reiner Hähnle	Technical University of Darmstadt, Germany
Andreas Herzig	IRIT-CNRS, France
Till Mossakowski	University of Magdeburg, Germany
Aniello Murano	Università di Napoli "Federico II", Italy
Jens Otten	University of Potsdam, Germany – chair
Cesare Tinelli	The University of Iowa, USA
Luca Viganò	King's College London, UK
Arild Waaler	University of Oslo, Norway
Frank Wolter	University of Liverpool, UK

Workshop Chairs

Christoph Benzmüller
Freie Universität Berlin
Arnimallee 7, 14195 Berlin, Germany
E-mail: `c.benzmueller@fu-berlin.de`

Jens Otten
University of Potsdam
August-Bebel-Straße 89, 14482 Potsdam-Babelsberg, Germany
E-mail: `jeotten@cs.uni-potsdam.de`

Program

Wednesday, 23 July 2014

10:15-10:45 Coffee Break

10:50-11:00 Opening

11:00-13:00 Session 1

11:00 *Damein Doligez, Jael Kriener, Leslie Lamport, Tomer Libal, and Stephan Merz*
Coalescing: Syntactic Abstraction for Reasoning in First-Order Modal Logics

11:30 *Ping Hou and Yifei Chen*
A Logic for Verifying Metric Temporal Properties in Distributed Hybrid Systems

12:00 *Jens Otten and Thomas Raths*
Problem Libraries for Non-Classical Logics

12:30 *Christoph Benzmüller*
HOL Provers for First-order Modal Logics — Experiments

13:00-14:30 Lunch Break

14:30-16:00 Session 2

14:30 *Till Mossakowski, Mihai Codescu, Oliver Kutz, Christoph Lange, and Michael Grüninger*
Proof Support for Common Logic

15:00 *Max Wisniewski and Alexander Steen*
Embedding of Quantified Higher-Order Nominal Modal Logic into Classical Higher-Order Logic

15:30 *Jesse Alama*
Dialogues for proof search

16:00-16:30 Coffee Break

16:30-18:00 Session 3

16:30 *Hans De Nivelle*
Theorem Proving for Logic with Partial Functions Using Kleene Logic and Geometric Logic

17:00 *Alexander Lyaletski* (presentation only)
Computer-oriented Inference Search in First-order Sequent Logics

17:30 Open Discussion

Coalescing: Syntactic Abstraction for Reasoning in First-Order Modal Logics [*]

Damien Doligez[1], Jael Kriener[2], Leslie Lamport[3],
Tomer Libal[2], and Stephan Merz[4]

[1] Inria, Paris, France
[2] MSR-Inria Joint Centre, Saclay, France
[3] Microsoft Research, Mountain View, CA, U.S.A.
[4] Inria, Villers-lès-Nancy, France

Abstract

We present a syntactic abstraction method to reason about first-order modal logics by using theorem provers for standard first-order logic and for propositional modal logic.

1 Introduction

Verification of distributed and concurrent systems requires reasoning about temporal behaviors. A common approach is to express the properties to be proved in a modal logic having one or more temporal modalities. For verifying real-world systems, a proof language must also include equality, quantification, interpreted theories, and local definitions. It must therefore encompass FOML (*F*irst *O*rder *M*odal *L*ogic) and support operator definitions. One such language is TLA$^+$ [10], based on the logic TLA that has two temporal modalities: the usual $\Box$ (always) operator of linear-time temporal logic and $'$ (prime), a restricted next-state operator such that e' is the value of e at the next state if e is an expression that does not contain a modal operator.

A common way to prove an FOML sequent $\Gamma \models \varphi$ (φ holds in context Γ) is to translate it to a semantically equivalent FOL sequent $\Gamma^* \models_{FOL} \varphi^*$ and to prove this FOL sequent. For some FOMLs, this method is semantically complete—that is, $\Gamma \models \varphi$ is valid iff $\Gamma^* \models_{FOL} \varphi^*$ is [12]. This approach has been followed for embedding FOML in SPASS [9], Saturate [7], and other theorem provers.

Such a semantic translation may be appropriate for completely automatic provers. However, we are very far from being able to automatically prove a formula that expresses a correctness property of a non-trivial system. A person must break the proof into smaller steps that we call *proof obligations*, usually by interacting with the prover. Requiring the user to interactively prove the semantic translation of the FOML formula destroys the whole purpose of using modal logic, which is to allow her to think in terms of the simpler FOML abstraction of the theorem. The user should therefore decompose the FOML proof into FOML proof obligations.

In this paper we describe a method called *coalescing* that handles many FOML proof obligations by soundly abstracting them into formulas of either FOL or propositional modal logic (ML). The resulting formulas are dealt with by existing theorem provers for these logics. Although the basic idea of coalescing is simple, some care has to be taken in the presence of equality and bound variables. The translation becomes trickier in the presence of defined operators.

[*]This work has been partially funded by the Microsoft Research-Inria Joint Centre, France. It has also been supported by the European Union Seventh Framework Programme under grant agreement no. 295261 (MEALS) and by the French BGLE Project ADN4SE.

Outline of this Paper. Section 2 motivates our proposal by its application within the TLA$^+$ Proof System TLAPS. Section 3 formally introduces FOML and its two fragments, FOL and ML. Sections 4 and 5 present coalescing for modal and first-order expressions respectively, proving their soundness. Section 6 extends the results to languages containing local definitions. In Section 7 we outline a proof of the completeness of coalescing for proving safety properties. Section 8 discusses semantic translation vs. coalescing and suggests some optimizations and future work.

2 Motivation

2.1 A Sample TLA$^+$ Proof

Our motivation comes from designing the TLAPS proof system [5] for TLA$^+$, which can check correctness proofs of complex, real-world algorithms [11]. The essence of TLA proofs is to decompose proofs of temporal logic formulas so that most of the obligations contain no modal operator except *prime*. Figure 1 contains the outline of the proof of a simple safety property in TLAPS that illustrates this decomposition. The system specification is formula *Spec*, defined to equal $Init \wedge \Box[Step]_v$. In this formula, *Init* is a *state predicate* that describes the possible initial states, and *Step* is an *action predicate* that describes possible state transitions. Syntactically: *Init* is a FOL formula containing state (a.k.a. flexible) variables; *Step* is a formula containing state variables, FOL operators, and the *prime* operator; and v is a tuple of all state variables in the specification. The formula $[Step]_v$ is a shorthand for $Step \vee (v' = v)$, and $\Box$ is the usual "always" operator of temporal logic. The temporal logic formula *Spec* is evaluated over ω-sequences of states; it is true of a sequence $s_0 s_1 \ldots$ iff *Init* is true at state s_0 and, for all pairs of states s_i and s_{i+1}, either *Step* is true or the value of v does not change. The definitions of the formulas *Init* and *Step*, and the reason for writing $\Box[Step]_v$ instead of $\Box Step$, are irrelevant in the context of this paper. We wish to prove that a state formula $Safe(p)$ is true throughout any behavior described by *Spec*, for every process $p \in Proc$.

The right-hand side of Figure 1 shows the assertion and proof of the theorem. The first step in the proof is purely first-order: it introduces a fresh constant p, assumes $p \in Proc$, and reduces the overall proof to showing the implication $Spec \Rightarrow \Box Safe(p)$. Step $\langle 1 \rangle 1$ asserts that the initial condition implies $Safe(p)$. This formula does not contain any modal operators. Step $\langle 1 \rangle 2$ shows that $Safe(p)$ is preserved by every transition (as specified by $[Step]_v$). The proof of this step is essentially first-order, although TLAPS must handle the *prime* modality. The basic idea is to distribute primes inward in expressions using rules such as $(x + y)' = x' + y'$,

```
$Init \triangleq \ldots$                 THEOREM $Spec \Rightarrow \forall\, p \in Proc : \Box Safe(p)$
$Step \triangleq \ldots$                 $\langle 1 \rangle$. SUFFICES ASSUME NEW $p \in Proc$
$v \triangleq \ldots$                                     PROVE  $Spec \Rightarrow \Box Safe(p)$
$Spec \triangleq Init \wedge \Box[Step]_v$      OBVIOUS
$Safe(p) \triangleq \ldots$              $\langle 1 \rangle 1$. $Init \Rightarrow Safe(p)$
                                  BY DEF $Init$, $Safe$
                                $\langle 1 \rangle 2$. $Safe(p) \wedge [Step]_v \Rightarrow Safe(p)'$
                                  BY DEF $Safe$, $Step$, $v$
                                $\langle 1 \rangle 3$. QED
                                  BY $\langle 1 \rangle 1$, $\langle 1 \rangle 2$, PTL DEF $Spec$
```

Figure 1: Proof of a safety property in TLAPS.

and then to replace the remaining primed expressions by new atoms. For this example, we are assuming that the specification is so simple that, after the definitions of *Init*, *Next*, v, and *Safe* have been expanded, the FOL proof obligations generated for these two steps can be discharged by a theorem prover.

Step $\langle 1 \rangle 3$ concludes the proof. It is justified by propositional temporal reasoning, in particular the principle

$$\frac{P \land A \Rightarrow P'}{P \land \Box A \Rightarrow \Box P}$$

The *PTL* in the step's proof tells TLAPS to invoke a PTL decision procedure, which it does after replacing *Spec* by its definition and the formulas *Init*, *Safe*(p) and $[\mathit{Next}]_v$ by fresh atoms. This effectively hides all operators other than those of propositional logic, $\Box$, and *prime*.

We call *coalescing* the process of replacing expressions by atoms. It is similar to the introduction of names for subformulas that theorem provers apply during pre-processing steps such as CNF transformation. However, it has a different purpose: the fresh names hide complex formulas that are meaningless to a proof backend for a fragment of the original logic. As explained in the example above, TLAPS uses coalescing in its translations to invoke FOL and PTL backend provers, where the first do not support the modal operators $\Box$ and *prime*, and the second do not support first-order constructs such as quantification, equality or terms. An idea similar to coalescing underlies the KSAT decision procedure [8] for propositional multi-modal logic in that modal top-level literals are abstracted by fresh atoms. Unlike KSAT, we consider first-order modal logic, and we do not recursively construct formulas that must be analyzed at deeper modal levels.

Coalescing cannot in itself be semantically complete because it cannot support proof steps that rely on the interplay of the sublogics. For example, separate FOL and PTL provers cannot prove rules that distribute quantifiers over temporal modalities. Similarly, proofs of liveness properties via well-founded orderings essentially mix quantification and temporal logic. However, we need very few such proof steps in actual proofs, and we can handle them using a more traditional backend that relies on a FOL translation of temporal modalities. Coalescing is complete for a class of temporal logic properties that includes safety properties, which can be established by propositional temporal logic from action-level hypotheses. For these applications, we have found coalescing to be more flexible and more powerful in practice than a more traditional FOL translation. In particular, proofs need not follow the simple schema of the proof shown in Figure 1 but can invoke auxiliary invariants or lemmas. The inductive reasoning underlying much of temporal logic is embedded in PTL decision procedures but would be difficult to automate in a FOL prover. On the other hand, the *prime* modality by itself is simple enough that it can be handled by a pre-processing step applied before passing the proof obligation to a FOL prover.

2.2 Coalescing In First-Order Modal Logic

We believe that coalescing will be useful for proofs in modal logics other than TLA$^+$. We therefore present its fundamental principles here using a simpler FOML containing a single modal operator $\Box$. Corresponding to the translations we have implemented in TLAPS, we give two translations of FOML obligations, one into FOL and the other into ML, and we prove their soundness.

The idea underlying coalescing is very simple: abstract away a class of operators by introducing a fresh atom in place of a subformula whose principal operator is in that class. However, doing this in a sound way in the presence of equality is not trivial because of the *Leibniz prin-*

ciple, which asserts $(d = e) \Rightarrow (P(d) = P(e))$ for any expressions d and e and operator P. The Leibniz principle is valid in FOL but not FOML, which makes translating from FOML obligations to FOL obligations tricky [6].

For example, the formula $(v = 0) \Rightarrow \Box(v = 0)$ is not valid in TLA$^+$ or more generally in FOML when v is flexible. A naive application of standard FOL provers could propagate the equality in the antecedent by substituting 0 for v throughout this formula, effectively applying the instance $((v = 0) = \text{TRUE}) \Rightarrow (\Box(v = 0) = \Box\text{TRUE})$ of the Leibniz principle, and consequently prove the formula using the axiom $\Box\text{TRUE}$. Such an approach is clearly unsound. The standard translation of FOML into predicate logic [12] avoids this problem by making explicit the states at which formulas are evaluated, but at the price of adding significant complexity to the formula. Moreover, one typically assumes specific properties about the accessibility relation(s) underlying modal logics. Incorporating these into first-order reasoning may not be easy. For example, the $\Box$ modality of TLA$^+$ corresponds to the transitive closure of the *prime* modality, and this is not first-order axiomatizable. Of course, whether this is an issue or not depends on the particular modal logic one is interested in: semantic translation works very well in applications such as [3] that are based on a modal logic whose frame conditions are first-order axiomatizable.

Our approach is to coalesce expressions and formulas that are outside the scope of a given theorem prover. For the example above, coalescing to FOL yields $(v = 0) \Rightarrow \boxed{\Box(v = 0)}$ where $\boxed{\Box(v = 0)}$ is a new 0-ary predicate symbol, and this formula is clearly not provable. Similarly, coalescing to ML yields $\boxed{v = 0} \Rightarrow \Box\boxed{v = 0}$ of propositional modal logic, and again, this formula is not provable. We give a detailed description of how to derive a new symbol $\boxed{exp}$ for an arbitrary expression *exp*. Care has to be taken when the coalesced expression contains bound variables. For example, a naive coalescing into FOL of the formula $\forall\, a : \Box(a = 1) \Rightarrow a = 1$, which is valid over reflexive frames, would yield $\forall\, a : \boxed{\Box(a = 1)} \Rightarrow a = 1$, from which we can deduce $\boxed{\Box(a = 1)} \Rightarrow 0 = 1$ and then $\forall\, a : \neg\Box(a = 1)$, which is clearly not valid. A correct coalescing yields $\forall\, a : \boxed{\Box(a = 1)}(a) \Rightarrow a = 1$.

Operator Definitions. Coalescing is trickier for a language with operator definitions like $P(x, y) \triangleq exp$, where *exp* does not contain free variables other than x and y. Definitions are necessary for structuring specifications and for managing the complexity of proofs through lemmas about the defined operators. We therefore do not want to systematically expand all defined operators in order to obtain formulas of basic FOML. The Leibniz principle may not hold for an expression $P(a, b)$ if the operator P is defined in terms of modal operators—that is, $(a = c) \wedge (b = d)$ need not imply $P(a, b) = P(c, d)$. It would therefore be unsound to encode P as an uninterpreted predicate symbol in first-order logic. We show how soundness is preserved by replacing an expression $P(a, b)$ with $\boxed{P, \epsilon_1, \epsilon_2}(a, b)$, for some suitable expressions ϵ_1 and ϵ_2 (described in Section 6.3 below), where $\boxed{P, \epsilon_1, \epsilon_2}$ can be defined so it satisfies the Leibniz principle and also satisfies $\boxed{P, \epsilon_1, \epsilon_2}(a, b) = P(a, b)$ in suitably extended models of FOML, ensuring equisatisfiability of the original and the coalesced formula. Since it satisfies the Leibniz principle, $\boxed{P, \epsilon_1, \epsilon_2}$ can be taken to be an uninterpreted predicate symbol by a first-order theorem prover. Our construction extends to the case of definitions of second-order operators, which are allowed in TLA$^+$.

3 First-Order Modal Logic

3.1 Syntax.

We introduce a language of first-order modal logic whose modal operator we denote by ∇ in order to avoid confusion with the $\Box$ of TLA$^+$. The language omits the customary distinction between function and predicate symbols, and hence between terms and formulas. This simplifies notation and allows our results to apply to TLA$^+$ as well as to a conventional language that does distinguish terms and formulas—the conventional language just having a smaller set of legal formulas.

We assume a first-order signature consisting of non-empty distinct denumerable sets $\mathcal{X}$ of rigid variables, $\mathcal{V}$ of flexible variables, and $\mathcal{O}$ of operator symbols. Operator symbols have arities in $\mathbb{N}$ and generalize both function and predicate symbols. Expressions e of FOML are then inductively defined by the following grammar:

$$e \;::=\; x \mid v \mid op(e, \dots, e) \mid e = e \mid \text{FALSE} \mid e \Rightarrow e \mid \forall x : e \mid \nabla e$$

where $x \in \mathcal{X}$, $v \in \mathcal{V}$, $op \in \mathcal{O}$, and arities are respected (empty parentheses are omitted for 0-ary symbols). We do not allow quantification over flexible variables, so our flexible variables are really "flexible function symbols of arity 0". While TLA$^+$ allows quantification over flexible variables, it can be considered as another modal operator for the purposes of coalescing.

The notions of free and bound (rigid) variables are the usual ones. We say that an expression is *rigid* iff it contains neither flexible variables nor subexpressions of the form ∇e. The standard propositional (TRUE, $\neg$, $\wedge$, $\vee$, $\equiv$) and first-order ($\exists$) connectives are defined in the usual way. The dual modality Δ is introduced by defining Δe as $\neg\nabla\neg e$. The extension to a multi-modal language is straightforward.

3.2 Semantics.

A *Kripke model* $\mathcal{M}$ for FOML is a 6-tuple $(\mathcal{I}, \xi, \mathcal{W}, R, \zeta, \nabla_{\mathcal{M}})$, where:

- $\mathcal{I}$ is a standard first-order interpretation consisting of a universe $|\mathcal{I}|$ and, for every operator symbol op, an interpretation $\mathcal{I}(op) : |\mathcal{I}|^n \to |\mathcal{I}|$ where n agrees with the arity of op. We assume that the universe $|\mathcal{I}|$ contains two distinguished, distinct values tt and ff.
- $\xi : \mathcal{X} \to |\mathcal{I}|$ is a valuation of the rigid variables.
- $\mathcal{W}$ is a non-empty set of states, and $R \subseteq \mathcal{W} \times \mathcal{W}$ is the accessibility relation.
- $\zeta : \mathcal{V} \times \mathcal{W} \to |\mathcal{I}|$ is a valuation of the flexible variables at the different states of the model.
- $\nabla_{\mathcal{M}} : 2^{|\mathcal{I}|} \to |\mathcal{I}|$ is a function such that $\nabla_{\mathcal{M}}(S) = \mathsf{tt}$ iff $S \subseteq \{\mathsf{tt}\}$.

Note that we assume a constant universe, independent of the states of the model, and we also assume that all operators in $\mathcal{O}$ are rigid—i.e., interpreted independently of the states.

We inductively define the interpretations of expressions $[\![e]\!]^{\mathcal{M}}_w$ at state w of model $\mathcal{M}$. When the model $\mathcal{M}$ is understood from the context, we drop it from the notation.

- $[\![x]\!]^{\mathcal{M}}_w \;=_{\mathsf{def}}\; \xi(x) \quad \text{for } x \in \mathcal{X}$
- $[\![v]\!]^{\mathcal{M}}_w \;=_{\mathsf{def}}\; \zeta(v, w) \quad \text{for } v \in \mathcal{V}$
- $[\![op(e_1, \dots, e_n)]\!]^{\mathcal{M}}_w \;=_{\mathsf{def}}\; \mathcal{I}(op)([\![e_1]\!]^{\mathcal{M}}_w, \dots, [\![e_n]\!]^{\mathcal{M}}_w) \quad \text{for } op \in \mathcal{O}$
- $[\![e_1 = e_2]\!]^{\mathcal{M}}_w \;=_{\mathsf{def}}\; \begin{cases} \mathsf{tt} & \text{if } [\![e_1]\!]^{\mathcal{M}}_w = [\![e_2]\!]^{\mathcal{M}}_w \\ \mathsf{ff} & \text{otherwise} \end{cases}$

- $[\![\text{FALSE}]\!]_w^{\mathcal{M}} \;=_{\mathsf{def}}\; \mathsf{ff}$
- $[\![\varphi \Rightarrow \psi]\!]_w^{\mathcal{M}} \;=_{\mathsf{def}}\; \begin{cases} \mathsf{tt} & \text{if } [\![\varphi]\!]_w^{\mathcal{M}} \neq \mathsf{tt} \text{ or } [\![\psi]\!]_w^{\mathcal{M}} = \mathsf{tt} \\ \mathsf{ff} & \text{otherwise} \end{cases}$
- $[\![\forall\, x : \varphi]\!]_w^{\mathcal{M}} \;=_{\mathsf{def}}\; \begin{cases} \mathsf{tt} & \text{if } [\![\varphi]\!]_w^{\mathcal{M}'} = \mathsf{tt} \text{ for all } \mathcal{M}' = (\mathcal{I}, \xi', \mathcal{W}, R, \zeta, \nabla_{\mathcal{M}}) \text{ such that} \\ & \quad \xi'(y) = \xi(y) \text{ for all } y \in \mathcal{X} \text{ different from } x \\ \mathsf{ff} & \text{otherwise} \end{cases}$
- $[\![\nabla\varphi]\!]_w^{\mathcal{M}} \;=_{\mathsf{def}}\; \nabla_{\mathcal{M}}(\{[\![\varphi]\!]_{w'}^{\mathcal{M}} : (w, w') \in R\})$

We write $\mathcal{M}, w \models \varphi$ instead of $[\![\varphi]\!]_w^{\mathcal{M}} = \mathsf{tt}$. We say that φ is *valid* iff $\mathcal{M}, w \models \varphi$ holds for all $\mathcal{M}$ and w, and that it is *satisfiable* iff $\mathcal{M}, w \models \varphi$ for some $\mathcal{M}$ and w. We define a consequence relation $\models$ as follows (where Γ is a set of formulas): $\Gamma \models \varphi$ iff for all $\mathcal{M}$, if $\mathcal{M}, w \models \psi$ for all $\psi \in \Gamma$ and $w \in \mathcal{W}$, then $\mathcal{M}, w \models \varphi$ for all $w \in \mathcal{W}$.

Our definition of the semantics is a straightforward extension of the standard Kripke semantics to our setting, where ∇e need not denote a truth value. The condition on the function $\nabla_{\mathcal{M}}$ used for interpreting ∇ ensures that $\mathcal{M}, w \models \nabla\varphi$ iff $\mathcal{M}, w' \models \varphi$ for all w' such that $(w, w') \in R$ as in the standard Kripke semantics. Because we assume a constant domain of interpretation, both Barcan formulas are valid—that is, we have validity of

$$(\forall x : \nabla\varphi) \;\equiv\; \nabla(\forall x : \varphi). \tag{1}$$

Moreover, since all operator symbols have rigid interpretations, it is easy to prove by induction on the expression syntax that $[\![e]\!]_w = [\![e]\!]_{w'}$ holds for all states w, w' whenever e is a rigid expression. It follows that implications of the form $\varphi \Rightarrow \nabla\varphi$ are valid for rigid φ—for example:

$$\forall\, x, y : (x = y) \Rightarrow \nabla(x = y). \tag{2}$$

3.3 FOL and ML fragments of FOML

Two natural sublogics of FOML are first-order logic (FOL) and propositional modal logic (ML).

FOL does not have flexible variables $\mathcal{V}$ or expressions ∇e. A first-order structure $(\mathcal{I}, \xi)$ consists of an interpretation $\mathcal{I}$ as above and a valuation ξ of the (rigid) variables. The inductive definition of the semantics consists of the relevant clauses of the one given above for FOML, and the notions of first-order validity $\models_{FOL} \varphi$, satisfiability, and consequence carry over in the usual way.

ML does not have rigid variables, quantifiers, operator symbols or equality. A (propositional) Kripke model for ML is given as $\mathcal{K} = (\mathcal{W}, R, \zeta)$ where the set of states $\mathcal{W}$ and the accessibility relation R are as for FOML, and the valuation $\zeta : \mathcal{V} \times \mathcal{W} \to \{\mathsf{tt}, \mathsf{ff}\}$ assigns truth values to flexible variables at every state. The inductive definition of $[\![e]\!]_w^{\mathcal{K}} \in \{\mathsf{tt}, \mathsf{ff}\}$ specializes to the following clauses:

- $[\![v]\!]_w^{\mathcal{K}} \;=\; \zeta(v, w)$ for $v \in \mathcal{V}$
- $[\![\text{FALSE}]\!]_w^{\mathcal{K}} \;=\; \mathsf{ff}$
- $[\![\varphi \Rightarrow \psi]\!]_w^{\mathcal{K}} \;=\; \mathsf{tt}$ iff $[\![\varphi]\!]_w^{\mathcal{K}} = \mathsf{ff}$ or $[\![\psi]\!]_w^{\mathcal{K}} = \mathsf{tt}$
- $[\![\nabla\varphi]\!]_w^{\mathcal{K}} \;=\; \mathsf{tt}$ iff $[\![\varphi]\!]_{w'}^{\mathcal{K}} = \mathsf{tt}$ for all $w' \in W$ such that $(w, w') \in R$

The notions of validity $\models_{ML} \varphi$, satisfiability, and consequence carry over as usual.

4 Coalescing Modal Expressions

4.1 Definition of the abstraction e_{FOL}

One of our objectives is to apply standard first-order theorem provers for proving theorems of FOML that are instances of first-order reasoning. Since the operator ∇ is not available in first-order logic, we must translate FOML formulas ψ to purely first-order formulas ψ_{FOL} such that the consequence $\Gamma_{FOL} \models_{FOL} \varphi_{FOL}$ entails $\Gamma \models \varphi$. A naive but unsound approach would be to replace the modal operator ∇ by a fresh monadic operator symbol *Nec*. As explained in Section 2, this approach is not sound. As we observed, a sound approach is to define φ_{FOL} by using the well-known standard translation from modal logic to first-order logic [4, 12] that makes explicit the FOML semantics. However, that translation introduces additional complexity—complexity that is unnecessary for proof obligations that follow from ordinary first-order reasoning.

Instead, we define φ_{FOL} to be a syntactic first-order abstraction of φ in which modal subexpressions are coalesced—that is, replaced by fresh operators. If φ is $(v = 0) \Rightarrow \nabla(v = 0)$, then φ_{FOL} is $(v = 0) \Rightarrow \boxed{\nabla(v = 0)}$, where $\boxed{\nabla(v = 0)}$ is a new 0-ary operator symbol.

We want to ensure that subexpressions appearing more than once are abstracted by the same operators, allowing for instances of first-order theorems to remain valid. This requires some care for expressions that contain bound variables. For example, we expect to prove

$$(\exists\, x, z : \nabla(v = x)) \equiv (\exists\, y : \nabla(v = y)) \tag{3}$$

We therefore define the fresh operator symbols $\boxed{\nabla e}$ as λ-abstractions over the bound variables occurring in e, and these are identified modulo α-equivalence. Formally, we let $e_{FOL} = e^{\varepsilon}_{FOL}$ where ε denotes the empty list and, for a list $\mathbf{y}$ of rigid variables, the first-order expression $e^{\mathbf{y}}_{FOL}$ over the extended set of variables $\mathcal{X} \cup \mathcal{V}$ is defined inductively as follows.

- $x^{\mathbf{y}}_{FOL} =_{\mathsf{def}} x$ for $x \in \mathcal{X}$ a rigid variable,
- $v^{\mathbf{y}}_{FOL} =_{\mathsf{def}} v$ for $v \in \mathcal{V}$ a flexible variable,
- $(op(e_1, \ldots, e_n))^{\mathbf{y}}_{FOL} =_{\mathsf{def}} op((e_1)^{\mathbf{y}}_{FOL}, \ldots, (e_n)^{\mathbf{y}}_{FOL})$ for $op \in \mathcal{O}$,
- $(e_1 = e_2)^{\mathbf{y}}_{FOL} =_{\mathsf{def}} (e_1)^{\mathbf{y}}_{FOL} = (e_2)^{\mathbf{y}}_{FOL}$,
- $\text{FALSE}^{\mathbf{y}}_{FOL} =_{\mathsf{def}} \text{FALSE}$
- $(e_1 \Rightarrow e_2)^{\mathbf{y}}_{FOL} =_{\mathsf{def}} (e_1)^{\mathbf{y}}_{FOL} \Rightarrow (e_2)^{\mathbf{y}}_{FOL}$,
- $(\forall\, x : e)^{\mathbf{y}}_{FOL} =_{\mathsf{def}} \forall\, x : e^{x,\mathbf{y}}_{FOL}$,
- $(\nabla e)^{\mathbf{y}}_{FOL} =_{\mathsf{def}} \boxed{\lambda \mathbf{z} : \nabla e}(\mathbf{z})$ where $\mathbf{z}$ is the subsequence of rigid variables in $\mathbf{y}$ that appear free in e. (If z is the empty sequence, this is simply $\boxed{\nabla e}$.)

With these definitions, the formula (3) is coalesced as

$$(\exists\, x, z : \boxed{\lambda x : \nabla(v = x)}(x)) \equiv (\exists\, y : \boxed{\lambda y : \nabla(v = y)}(y)) \tag{4}$$

which is an instance of the valid first-order equivalence

$$(\exists\, x, z : P(x)) \equiv (\exists\, y : P(y))$$

In particular, the two operator symbols occurring in (4) are identified because the two λ-expressions are α-equivalent. Identification of coalesced formulas modulo α-equivalence ensures that the translation is insensitive to the names of bound (rigid) variables. Section 8 discusses techniques for abstracting from less superficial differences in first-order expressions, such as between $\lambda x, y$ and $\lambda y, x$ and between $a = b$ and $b = a$.

4.2 Soundness of coalescing to FOL

For a set Γ of FOML formulas, we denote by Γ_{FOL} the set of all formulas ψ_{FOL}, for $\psi \in \Gamma$. We now show the soundness of the abstraction.

Theorem 1. *For any set Γ of FOML formulas and any FOML formula φ, if $\Gamma_{FOL} \models_{FOL} \varphi_{FOL}$ then $\Gamma \models \varphi$.*

Proof (sketch). Assume that $\Gamma \not\models \varphi$, so $\mathcal{M} = (\mathcal{I}, \xi, \mathcal{W}, R, \zeta, \nabla_{\mathcal{M}})$ is a Kripke model such that $\mathcal{M}, w' \models \psi$ for all $\psi \in \Gamma$ and $w' \in \mathcal{W}$, but that $\mathcal{M}, w \not\models \varphi$ for some $w \in \mathcal{W}$.

For the extended set of variables $\mathcal{X} \cup \mathcal{V}$, define the first-order structure $\mathcal{S} = (\mathcal{I}', \xi')$ where $\mathcal{I}'$ agrees with $\mathcal{I}$ for all operator symbols that appear in Γ or φ, and where the valuation ξ' is defined by $\xi'(x) = \xi(x)$ for $x \in \mathcal{X}$ and $\xi'(v) = \zeta(w, v)$ for $v \in \mathcal{V}$. For the additional operator symbols introduced in Γ_{FOL} and φ_{FOL}, we define

$$\mathcal{I}'(\boxed{\lambda \mathbf{z} : \nabla e})(d_1, \ldots, d_n) \quad = \quad [\![\nabla e]\!]_w^{\mathcal{M}'}$$

where $\mathcal{M}'$ agrees with $\mathcal{M}$ except for the valuation ξ' that assigns the i^{th} variable of $\mathbf{z}$ to d_i. This interpretation is well-defined: if ∇e_1 and ∇e_2 are two expressions in Γ or φ that give rise to the same operator symbol, then $(\lambda \mathbf{z}_1 : \nabla e_1)$ and $(\lambda \mathbf{z}_2 : \nabla e_2)$ must be α-equivalent, and therefore $\mathcal{I}'(\boxed{\lambda \mathbf{z}_1 : \nabla e_1})(d_1, \ldots, d_n) = \mathcal{I}'(\boxed{\lambda \mathbf{z}_2 : \nabla e_2})(d_1, \ldots, d_n)$.

It is straightforward to prove that $[\![e_{FOL}]\!]^{\mathcal{S}} = [\![e]\!]_w^{\mathcal{M}}$ holds for all expressions e_{FOL} that appear in Γ_{FOL} or φ_{FOL}. In particular, it follows that $\mathcal{S} \models_{FOL} \psi_{FOL}$ for all $\psi \in \Gamma$ and $\mathcal{S} \not\models_{FOL} \varphi_{FOL}$. This shows that $\Gamma_{FOL} \not\models_{FOL} \varphi_{FOL}$ and concludes the proof. Q.E.D.

5 Coalescing First-Order Expressions

We now define an abstraction φ_{ML} of FOML formulas to formulas of propositional modal logic. Again, we require for soundness that $\Gamma \models \varphi$ whenever $\Gamma_{ML} \models_{ML} \varphi_{ML}$—that is, consequence between abstracted formulas implies consequence between the original ones. In this way, we can use theorem provers for propositional modal logic to carry out FOML proofs that are instances of propositional modal reasoning. The abstraction φ_{ML} replaces all first-order subexpressions e of φ by new (propositional) flexible variables $\boxed{e}$, where variables $\boxed{\forall x : e}$ are once again identified modulo α-equivalence. Formally, the translation is defined as follows.

- $x_{ML} =_{\mathsf{def}} \boxed{x}$ for $x \in \mathcal{X}$ a rigid variable,
- $v_{ML} =_{\mathsf{def}} v$ for $v \in \mathcal{V}$ a flexible variable,
- $(op(t_1, \ldots, t_n))_{ML} =_{\mathsf{def}} \boxed{op(t_1, \ldots, t_n)}$ for $op \in \mathcal{O}$,
- $(e_1 = e_2)_{ML} =_{\mathsf{def}} \boxed{e_1 = e_2}$,
- $\text{FALSE}_{ML} =_{\mathsf{def}} \text{FALSE}$,
- $(e_1 \Rightarrow e_2)_{ML} =_{\mathsf{def}} (e_1)_{ML} \Rightarrow (e_2)_{ML}$,
- $(\forall x : e)_{ML} =_{\mathsf{def}} \boxed{\forall x : e}$,
- $(\nabla e)_{ML} =_{\mathsf{def}} \nabla e_{ML}$.

As an example, coalescing the formula

$$(x = y) \land \nabla\Delta\text{TRUE} \Rightarrow \nabla\Delta(x = y)$$

yields the ML-formula

$$\boxed{x = y} \land \nabla\Delta\text{TRUE} \Rightarrow \nabla\Delta\boxed{x = y} \tag{5}$$

The implication (5) is not ML-valid. However, for rigid variables x and y, it follows from the hypothesis $\boxed{x = y} \Rightarrow \nabla\boxed{x = y}$, which is justified by the FOML law (2).

For a set Γ of FOML formulas, we denote by Γ_{ML} the set of modal abstractions ψ_{ML}, for all $\psi \in \Gamma$. Moreover, we define the set $\mathcal{H}(\Gamma)$ to consist of all formulas of the form $\boxed{e} \Rightarrow \nabla\boxed{e}$, for all flexible variables $\boxed{e}$ introduced in Γ_{ML} that correspond to rigid expressions e in Γ.

Theorem 2. *Assume that Γ is a set of FOML formulas and that φ is a FOML formula. If $\Gamma_{ML}, \mathcal{H}(\Gamma \cup \{\varphi\}) \models_{ML} \varphi_{ML}$ then $\Gamma \models \varphi$.*

Proof (sketch). As in Theorem 1, we prove the contra-positive. Assume that $\mathcal{M} = (\mathcal{I}, \xi, \mathcal{W}, R, \zeta, \nabla_{\mathcal{M}})$ is a Kripke model such that $\mathcal{M}, w' \models \psi$ for all $\psi \in \Gamma$ and $w' \in \mathcal{W}$, but $\mathcal{M}, w \not\models \varphi$ for a certain $w \in \mathcal{W}$.

Define the propositional Kripke model $\mathcal{K} = (\mathcal{W}, R, \zeta')$ where ζ' assigns truth values in $\{\mathsf{tt}, \mathsf{ff}\}$ to all states $w' \in \mathcal{W}$ and flexible variables in Γ_{ML} or φ_{ML}:

$$\begin{aligned}
&\zeta'(w', v) = \mathsf{tt} \ \text{ iff } \ \zeta(w', v) = \mathsf{tt} \ \text{ for } v \in \mathcal{V}\\
&\zeta'(w', \boxed{x}) = \mathsf{tt} \ \text{ iff } \ \xi(x) = \mathsf{tt} \ \text{ for } x \in \mathcal{X}\\
&\zeta'(w', \boxed{op(t_1, \ldots, t_n)}) = \mathsf{tt} \ \text{ iff } \ [\![op(t_1, \ldots, t_n)]\!]^{\mathcal{M}}_{w'} = \mathsf{tt}\\
&\zeta'(w', \boxed{e_1 = e_2}) = \mathsf{tt} \ \text{ iff } \ [\![e_1]\!]^{\mathcal{M}}_{w'} = [\![e_2]\!]^{\mathcal{M}}_{w'}\\
&\zeta'(w', \boxed{\forall x : e}) = \mathsf{tt} \ \text{ iff } \ \mathcal{M}, w' \models \forall x : e
\end{aligned}$$

Again, ζ' is well-defined. It is easy to prove, for all $w' \in \mathcal{W}$ and all e such that e_{ML} appears in Γ_{ML} or φ_{ML}, that $\mathcal{K}, w' \models e_{ML}$ iff $[\![e]\!]^{\mathcal{M}}_{w'} = \mathsf{tt}$. In particular, it follows that $\mathcal{K}, w' \models \psi_{ML}$ for all $\psi \in \Gamma$ and that $\mathcal{K}, w \not\models_{ML} \varphi_{ML}$.

Furthermore, the definition of $\mathcal{K}$ ensures that $\mathcal{K}, w' \models \psi$ holds for all $\psi \in \mathcal{H}(\Gamma \cup \{\varphi\})$ and all $w' \in \mathcal{W}$ because $[\![e]\!]^{\mathcal{M}}_{w'} = [\![e]\!]^{\mathcal{M}}_{w''}$ holds for all rigid expressions e and all states $w', w'' \in \mathcal{W}$.

It follows that $\Gamma_{ML}, \mathcal{H}(\Gamma \cup \{\varphi\}) \not\models_{ML} \varphi_{ML}$, which concludes the proof. Q.E.D.

6 Coalescing in the presence of operator definitions

6.1 Operator definitions

We now extend our language to allow definitions of the form

$$d(x_1, \ldots, x_n) \triangleq e$$

where d is a fresh symbol, $x_1, \ldots, x_n$ are pairwise distinct rigid variables, and e is an expression whose free rigid variables are among $x_1, \ldots, x_n$.

For an operator d defined as above and expressions $e_1, \ldots, e_n$, the application $d(e_1, \ldots, e_n)$ is a well-formed expression whose semantics is given by:

$$[\![d(e_1, \ldots, e_n)]\!]^{\mathcal{M}}_{w} = [\![e[e_1/x_1, \ldots, e_n/x_n]]\!]^{\mathcal{M}}_{w}$$

In other words, the defining expression is evaluated when the arguments have been substituted for the variables. However, when reasoning about expressions containing defined operators, one does not wish to systematically expand definitions. If the precise definition is unimportant, it is better to leave the operator unexpanded in order to keep the formulas small. We now extend the coalescing techniques introduced in the preceding sections to handle expressions that may contain defined operators.

It is easy to see that the algorithm introduced in Section 5 for abstracting first-order subexpressions remains sound if we handle defined operators like operators in $\mathcal{O}$. In particular, two expressions $d(\mathbf{e}_1)$ and $d(\mathbf{e}_2)$ are abstracted by the same flexible variable only if they are syntactically equal up to α-equivalence. However, this simple approach does not work for the algorithm of Section 4 that abstracts modal subexpressions. As an example, consider the definition

$$cst(x) \ \triangleq\ \exists\, y : \nabla(x = y) \tag{6}$$

and the formula

$$(u = v) \ \Rightarrow\ (cst(u) \equiv cst(v)) \tag{7}$$

where u and v are flexible variables. An expression e satisfies $cst(e)$ at state w iff the value of e is the same at all reachable states w'. Hence, formula (7) is obviously not valid. If cst were treated like an operator in $\mathcal{O}$, the algorithm of Section 4 would leave (7) unchanged. However, u and v would be considered ordinary (rigid) variables and cst would be considered an uninterpreted operator symbol, so (7), seen as a FOL formula, would be provable. Thus, it would be unsound to simply treat defined operators like operators in $\mathcal{O}$ in our algorithm for coalescing modal subexpressions.

6.2 Rigid arguments and Leibniz positions

The example above shows that in the presence of definitions, FOML formulas without any visible modal operators may violate the Leibniz principle that substituting equals for equals should yield equal results. However, a first observation shows that the Leibniz principle still holds for rigid arguments.

Lemma 3. *For any defined n-ary operator d, expressions $e_1, \ldots, e_n$, any $i \in 1\,..\,n$ with e_i rigid, Kripke model $\mathcal{M}$, state w, and rigid variable x that does not occur free in any e_j, we have*

$$[\![d(e_1, \ldots, e_n)]\!]_w^{\mathcal{M}} = [\![d(e_1, \ldots, e_{i-1}, x, e_{i+1}, \ldots, e_n)]\!]_w^{\mathcal{M}'}$$

where $\mathcal{M}'$ agrees with $\mathcal{M}$ except for the valuation ξ' of rigid variables, which is like ξ but assigns x to $[\![e_i]\!]_w^{\mathcal{M}}$.

Proof (sketch). Since e_i is rigid, the value of $[\![e_i]\!]_{w'}^{\mathcal{M}}$, for any $w' \in W$, is independent of the state w'. The assertion is then proved by induction on the defining expression for operator d. Q.E.D.

For a non-rigid argument of a defined operator, the Leibniz principle is preserved when the argument does not appear in a modal context in the defining expression. We inductively define which argument positions of an FOML operator or connective are Leibniz (satisfy the Leibniz principle).

Definition 4 (Leibniz argument positions)**.**

- *All argument positions of the operators in $\mathcal{O}$ and of all FOML connectives except ∇ are Leibniz. The single argument position of ∇ is not Leibniz.*
- *For an operator defined by $d(x_1, \ldots, x_n) \triangleq e$, the i^{th} argument position of d is Leibniz iff x_i does not occur within a non-Leibniz argument position in e.*

In other words, the i^{th} argument position of a defined operator is Leibniz iff the i^{th} parameter does not appear in the scope of any occurrence of ∇ in the full expansion of the defining expression.

Lemma 5. *Assume that d is a defined n-ary operator whose i^{th} argument position is Leibniz (for $i \in 1 \mathrel{..} n$). For any expressions $e_1, \ldots, e_n$, Kripke model $\mathcal{M}$, state w and rigid variable x that does not occur free in any e_j, we have*

$$[\![d(e_1, \ldots, e_n)]\!]^{\mathcal{M}}_w = [\![d(e_1, \ldots, e_{i-1}, x, e_{i+1}, \ldots, e_n)]\!]^{\mathcal{M}'}_w$$

where $\mathcal{M}'$ agrees with $\mathcal{M}$ except for the valuation ξ' of rigid variables, which is like ξ but assigns x to $[\![e_i]\!]^{\mathcal{M}}_w$.

Proof (sketch). Induction on the syntax of the defining expression for d. Q.E.D.

It follows from Lemmas 3 and 5 that the implication

$$(e_i = f) \Rightarrow (d(e_1, \ldots, e_n) = d(e_1, \ldots, e_{i-1}, f, e_{i+1}, \ldots, e_n))$$

is valid when e_i and f are rigid expressions or when the i^{th} argument position of d is Leibniz.

6.3 Coalescing for defined operators

The definition of the syntactic abstraction e_{FOL} for the extended language is now completed by defining

- $(d(e_1, \ldots, e_n))^{\mathbf{y}}_{FOL} =_{\mathsf{def}} \boxed{d, \epsilon_1, \ldots, \epsilon_n}((e_1)^{\mathbf{y}}_{FOL}, \ldots, (e_n)^{\mathbf{y}}_{FOL})$ for a defined n-ary operator d where

$\epsilon_i = *$	if the i^{th} position of d is Leibniz or e_i is a rigid expression,
$\epsilon_i = e_i$	otherwise.

With these definitions, the single argument position of operator cst introduced by (6) is not Leibniz, and coalescing formula (7) yields

$$(v = w) \Rightarrow (\boxed{cst, v}(v) \equiv \boxed{cst, w}(w))$$

for two distinct fresh operators $\boxed{cst, v}$ and $\boxed{cst, w}$. As expected, this formula cannot be proved. However, the formula $\forall x, y : (x = y) \Rightarrow (cst(x) \equiv cst(y))$ is coalesced as $\forall x, y : (x = y) \Rightarrow (\boxed{cst, *}(x) \equiv \boxed{cst, *}(y))$ and is valid.

Theorem 6. *Theorem 1 remains valid for FOML formulas in the presence of defined operator symbols.*

Proof (sketch). Extending the proof of Theorem 1, we define the interpretation of the fresh operator symbols as follows:

$$\begin{array}{l} \mathcal{I}'(\boxed{d, \epsilon_1, \ldots, \epsilon_n})(d_1, \ldots, d_n) \ = \ [\![d(\alpha_1, \ldots, \alpha_n)]\!]_w^{\mathcal{M}'} \\ \text{where } \alpha_i = \begin{cases} e_i & \text{if } \epsilon_i = e_i \\ x_i & \text{if } \epsilon_i = * \end{cases} \end{array}$$

In this definition, w is the state fixed in the proof and $\mathcal{M}'$ agrees with $\mathcal{M}$ except for the valuation ξ' that assigns the variables x_i to d_i.

Again, one proves that $[\![e_{FOL}]\!]^{\mathcal{S}} = [\![e]\!]_w^{\mathcal{M}}$ for all expressions e_{FOL} that appear in Γ_{FOL} or φ_{FOL}. For the expressions corresponding to applications of defined operators, the proof is obvious for those arguments where $\epsilon_i = e_i$, and it makes use of Lemmas 3 and 5 when $\epsilon_i = *$. Q.E.D.

7 Proving Safety Properties by Coalescing in TLA

We now give evidence for the usefulness of coalescing by showing that it can be the basis for a complete proof system for establishing safety properties in TLA$^+$, assuming the ability to prove any valid first-order formula in the set-theoretic language underlying TLA$^+$. In fact, we argue that proofs of arbitrary safety properties can be transformed into the form used in Section 2.

In TLA$^+$, properties of systems are expressed as temporal logic formulas, which are evaluated over infinite sequences of states. We say that a formula is true of a finite sequence of states if it is true for some infinite extension of that finite sequence. A formula expresses a *safety property* if it holds for an infinite sequence of states if and only if it holds for every finite prefix of that sequence.

The standard form of a TLA$^+$ specification is $Init \wedge \Box[Next]_v \wedge L$ where $Init$ is a state predicate (a first-order formula that contains only unprimed state variables), v is a tuple containing all state variables, $Next$ is an action formula (a formula of first-order logic extended by the *prime* operator), and L is a conjunction of fairness conditions that are irrelevant for proving safety properties. In order to prove that the specification establishes a safety property P, we need to establish the theorem

$$Init \wedge \Box[Next]_v \Rightarrow P.$$

The first step is to reformulate the problem as an *invariant assertion* of the form

$$HInit \wedge \Box[HNext]_{hv} \Rightarrow \Box Inv \tag{8}$$

where Inv is a state predicate. A general way to do this is to add a *history variable* [1] to the original specification. The history variable records the sequence of states seen so far, and Inv is true for a given value of the history variable if and only if the safety property P is true for the corresponding sequence of states.

The second step is to prove the invariance of Inv by using an *inductive invariant*—a state predicate $IInv$ such that all of the following formulas are valid:

1. $HInit \Rightarrow IInv$
2. $IInv \land [HNext]_{hv} \Rightarrow IInv'$
3. $IInv \Rightarrow Inv$

Assuming that formula (8) is valid, a suitable inductive invariant $IInv$ exists under standard assumptions on the expressiveness of the language of state predicates (see, e.g., [2]), and these assumptions are satisfied by the set-theoretic language underlying TLA$^+$.

By coalescing to propositional temporal logic using the techniques described in Section 5, a PTL decision procedure easily checks that (8) follows from facts (1)–(3) above. In practice, we find it preferable to verify inferences in temporal logic by appealing to a decision procedure rather than by applying a fixed set of rules because the user is then free to structure the proof in the most convenient way. For example, the inductive invariant could be split into several mutually inductive formulas, for which the corresponding facts may be easier to check than (1)–(3) in the standard invariance proof above. On the other hand, the inductive nature of the typical temporal logic proofs such as the one above makes it unrealistic to expect that an ordinary FOL prover would be able to establish the FOL translation of (8) from (1)–(3).

Completing the proof of (8) requires proving formulas (1)–(3). Observe that (1) and (3) are state predicates and thus ordinary first-order formulas in TLA$^+$'s set theory. By assumption, their proofs can be discharged by the underlying FOL prover.

Formula (2) is an action formula and therefore contains TLA$^+$'s *prime* operator in addition to first-order logic, but no other operator of temporal logic. We now show how we reduce the proof of an action formula A to FOL proofs with the help of coalescing.

We begin by eliminating all defined operators that occur in A, expanding their definitions. This results in an action formula B that is equivalent to A, but that only contains built-in TLA$^+$ operators. Second, we use the fact that *prime* distributes over all non-modal built-in TLA$^+$ operators and rewrite formula B to an equivalent action formula C in which *prime* is only applied to flexible variables.[1] Finally, we apply the coalescing technique described in Section 4 for abstracting the *prime* modality, and obtain a formula C_{FOL}. It can be shown that for the restricted fragment where all modal expressions are of the form v' for flexible variables v, the formula C is FOML-valid if and only if C_{FOL} is FOL-valid, and therefore the underlying FOL prover will be able to prove formula (2) above.

The procedure above shows how in theory the two forms of coalescing that we have proposed in this paper can be complete for proving safety properties in TLA$^+$, assuming that we have a complete proof procedure for the set theory underlying TLA$^+$. In practice, no such procedure can exist and it is preferable to keep formulas small, so it is important not to expand all definitions. These practical considerations underlie the handling of defined operators described in Section 6. Although we have focused on safety properties, for which we can obtain a completeness result, the same overall technique can also be applied to the proof of liveness properties. In TLA, liveness properties are also proved by combining action-level reasoning and temporal logic proof rules. Most of the steps in such a proof can be reduced by coalescing to FOL or propositional ML reasoning. However, there will often be a few steps that require non-propositional temporal logic reasoning—for example, because they are based on well-founded orderings. Because there are not many such steps, a backend prover for them with a low degree of automation should be acceptable.

[1]The syntax of TLA$^+$ ensures that *prime* cannot be nested.

8 Conclusion

We presented a technique of coalescing that allows a user to decompose the proof of FOML formulas into purely first-order and purely propositional modal reasoning. This technique is inspired by reasoning about TLA$^+$ specifications and has been implemented in the TLA$^+$ Proof System, where we have found it useful for verifying temporal logic properties. In particular, the overwhelming majority of proof obligations that arise during TLA$^+$ proofs contain only the *prime* modal operator. For this fragment, rewriting by the valid equality $op(e_1, \ldots, e_n)' = op(e_1', \ldots, e_n')$, for operators $op \in \mathcal{O}$, followed by coalescing to FOL is complete. Many of the proof obligations that involve the $\Box$ modality of TLA$^+$ are instances of propositional temporal reasoning, and these can be handled by coalescing to ML and invoking a decision procedure for propositional temporal logic.

Coalescing to FOL eschews semantic translation of FOML formulas [12] in favor of replacing a subformula whose principal operator is modal by a fresh operator symbol. The resulting formulas are simpler than those obtained by semantic translation, and they can readily be understood in terms of the original FOML formulation of the problem. Coalescing is not a complete proof procedure in itself. For example, the valid Barcan formula (1) cannot be proved using only our two translations. TLA proofs contain only a small number of such proof obligations, and we expect TLAPS to be able to handle them with a semantic translation to FOL. In our context, the validity problem of first-order temporal logic is Π^1_1-complete, so incompleteness should not be considered an argument against the use of coalescing. Semantic translation of temporal logic would require inductive reasoning over natural numbers, and we could not even expect simple proofs such as the one shown in Section 2 to be discharged automatically, whereas proof by coalescing benefits from efficient decision procedures for propositional temporal logic. For applications other than TLA$^+$ theorem proving that require first-order modal reasoning, the trade-off in choosing between semantic translation and coalescing will depend upon how effective one expects semantic translation and standard first-order theorem proving to work in practice. One recent experiment [3] found this technique entirely satisfactory, but it used a modal logic too weak to handle the applications that concern us.

The definition of coalescing to FOL presented in Section 4 identifies modal subformulas such as (3) that are identical up to the names of bound rigid variables that they contain. This definition can be refined to identify formulas that differ in less superficial ways. For example, it may be desirable to reorder bound variables according to their appearance in coalesced subformulas. This would allow us to coalesce the formula

$$(\exists\, y\, \forall\, x : \Box P(x, y)) \Rightarrow (\forall\, x\, \exists\, y : \Box P(x, y))$$

to the valid FOL formula

$$(\exists\, y\, \forall\, x : \boxed{\lambda x, y : \Box P(x, y)}\,(x, y)) \Rightarrow (\forall\, x\, \exists\, y : \boxed{\lambda x, y : \Box P(x, y)}\,(x, y))$$

rather than the formula

$$(\exists\, y\, \forall\, x : \boxed{\lambda y, x : \Box P(x, y)}\,(y, x)) \Rightarrow (\forall\, x\, \exists\, y : \boxed{\lambda x, y : \Box P(x, y)}\,(x, y))$$

obtained according to the definition given in Section 4, which results in the two fresh operators being distinct. In general, we would like coalesced versions of different expressions to use the same atomic symbol wherever that would be valid. For example, $\boxed{e_1 = e_2}$ and $\boxed{e_2 = e_1}$ could be the same symbol.

Rewriting a formula before coalescing can also make the translated obligation easier to prove. For example, the formula $\Box e$ for a rigid expression e can be replaced by $\boxed{\Box\text{FALSE}} \lor e$. In a modal logic whose $\Box$ modality is reflexive, the disjunct $\boxed{\Box\text{FALSE}}$ is not necessary. In this way, the formula

$$\forall\, x, y : \Box(x = y) \Rightarrow \Box(f(x) = f(y))$$

for $f \in \mathcal{O}$ could be proved directly by translating with coalescing to FOL instead of requiring two steps, the first proving $(x = y) \Rightarrow (f(x) = f(y))$ with FOL and the second being translated to ML. Another such rewriting is distributing TLA's modal *prime* operator over rigid operators used by TLAPS when translating to FOL.

We don't know yet if optimizations of the translations beyond those we have already implemented in TLAPS will be useful in practice. So far, we have proved only safety properties for realistic algorithms, which in TLA requires little temporal reasoning. We have begun writing formal liveness proofs, but TLAPS will not completely check them until we have a translation that can handle formulas which, like the Barcan formula, inextricably mix quantifiers and modal operators.

References

[1] Martín Abadi and Leslie Lamport. The existence of refinement mappings. *Theoretical Computer Science*, 81(2):253–284, May 1991.

[2] Krzysztof R. Apt. Ten years of Hoare's logic: A survey—part I. *ACM Trans. Program. Lang. Syst.*, 3(4):431–483, October 1981.

[3] Christoph Benzmüller and Bruno Woltzenlogel Paleo. Gödel's God on the computer. In Stephan Schulz, Geoff Sutcliffe, and Boris Konev, editors, *10th Intl. Workshop Implementation of Logics*, EPiC Series. EasyChair, 2013.

[4] Torben Braüner and Silvio Ghilardi. First order modal logic. In Patrick Blackburn, Johan van Benthem, and Frank Wolter, editors, *Handbook of Modal Logic*, pages 549–620. Elsevier, 2007.

[5] Denis Cousineau, Damien Doligez, Leslie Lamport, Stephan Merz, Daniel Ricketts, and Hernán Vanzetto. TLA$^+$ proofs. In Dimitra Giannakopoulou and Dominique Méry, editors, *18th Intl. Symp. Formal Methods (FM 2012)*, volume 7436 of *LNCS*, pages 147–154, Paris, France, 2012. Springer.

[6] Melvin Fitting and Richard L. Mendelsohn. *First-Order Modal Logic.* Synthese Library. Springer, 1998.

[7] Harald Ganzinger, Robert Nieuwenhuis, and Pilar Nivela. The Saturate system, 1998. `http://www.mpi-inf.mpg.de/SATURATE/doc/Saturate/Saturate.html`.

[8] Fausto Giunchiglia and Roberto Sebastiani. Building decision procedures for modal logics from propositional decision procedures: The case study of modal k(m). *Information and Computation*, 162(1-2):158–178, 2000.

[9] Ullrich Hustadt and Renate A. Schmidt. MSPASS: Modal reasoning by translation and first-order resolution. In Roy Dyckhoff, editor, *TABLEAUX*, volume 1847 of *LNCS*, pages 67–71. Springer, 2000.

[10] Leslie Lamport. *Specifying Systems, The TLA$^+$ Language and Tools for Hardware and Software Engineers.* Addison-Wesley, 2002.

[11] Leslie Lamport. Byzantizing Paxos by refinement. In David Peleg, editor, *Distributed Computing: 25th Intl. Symp. (DISC 2011)*, pages 211–224. Springer-Verlag, 2011.

[12] Hans Jürgen Ohlbach. Semantics-based translation methods for modal logics. *J. Log. Comput.*, 1(5):691–746, 1991.

A Logic for Verifying Metric Temporal Properties in Distributed Hybrid Systems

Ping Hou and Yifei Chen[1*]

School of Information Science, Nanjing Audit University, Nanjing, China
phou@cs.cmu.edu, yifeichen91@nau.edu.cn

Abstract

We introduce a logic for specifying and verifying metric temporal properties of distributed hybrid systems that combines quantified differential dynamic logic (Qd$\mathcal{L}$) for reasoning about the possible behavior of distributed hybrid systems with metric temporal logic (MTL) for reasoning about the metric temporal behavior during their operation. For our combined logic, we generalize the semantics of dynamic modalities to refer to hybrid traces instead of final states. Further, we prove that this gives a conservative extension of Qd$\mathcal{L}$ for distributed hybrid systems. On this basis, we provide a modular verification calculus that reduces correctness of metric temporal behavior of distributed hybrid systems to generic temporal reasoning and then non-temporal reasoning, and prove that we obtain a complete axiomatization relative to the non-temporal base logic Qd$\mathcal{L}$.

1 Introduction

Functional specifications for distributed hybrid systems [6, 17, 22, 23] with discrete, continuous, structural, and dimension-changing dynamics usually involve a number of critical properties such as timing requirements, stability and bounded response. Metric Temporal Logic (MTL) [16] is a popular formalism for expressing such properties. The problem of verifying MTL specifications is undecidable for hybrid systems and distributed hybrid systems. Consequently, the *bounded-time* verification or falsification of such properties has been studied [10, 11, 18, 20, 27]. In addition to having successful uses in simulation-based methods, metric temporal logic has been used in deductive approaches to prove validity of formulas in calculi [12, 19]. Valid MTL formulas, however, cannot generally characterize the operations of a specific system.

Very recently, a dynamic logic, called *quantified differential dynamic logic* (Qd$\mathcal{L}$) has been introduced as a successful tool for deductively verifying distributed hybrid systems [22, 23]. Qd$\mathcal{L}$ can analyze the behavior of actual distributed hybrid system models, which are specified operationally. Yet, operational distributed hybrid system models are *internalized* within Qd$\mathcal{L}$ formulas. However, Qd$\mathcal{L}$ only considers the behavior of distributed hybrid systems at final states, which is insufficient for verifying properties that hold within certain time bounds, throughout the execution of the system.

We close this gap of expressivity by combining Qd$\mathcal{L}$ with metric temporal logic. In this paper, we introduce a logic, called *quantified differential metric temporal dynamic logic* (QdMTL), which provides modalities for quantifying over traces of distributed hybrid systems based on Qd$\mathcal{L}$. We equip QdMTL with metric temporal operators to state what is true all along a time interval of a trace or at some time point during a trace. In this paper, we modify the semantics of the dynamic modality $[\alpha]$ to refer to all *traces* of α instead of all final states reachable with α

[*]The work of this author is supported by the National Natural Science Foundation of China (No.61202135) and the Natural Science Foundation of Jiangsu Province, China (No.BK2012472).

(similarly for $\langle\alpha\rangle$). For instance, the formula $[\alpha]\Box_{\mathcal{I}}\,\phi$ expresses that ϕ is true at each state within the time interval $\mathcal{I}$ along all traces of the distributed hybrid system α. With this, QdMTL can also be used to verify temporal statements about the behavior of α at intermediate states during system runs. As in our non-temporal dynamic logic Q$\mathsf{d}\mathcal{L}$, we use *quantified hybrid programs* as an operational model for distributed hybrid systems, since they admit a uniform compositional treatment of interacting discrete transitions, continuous evolutions, and structural/dimension changes in logic.

As a semantical foundation for combined quantified temporal dynamic formulas, we introduce a hybrid trace semantics for QdMTL. We prove that QdMTL is a conservative extension of Q$\mathsf{d}\mathcal{L}$: for non-temporal specifications, trace semantics is equivalent to the non-temporal transition semantics of Q$\mathsf{d}\mathcal{L}$ [22, 23].

As a means for verification, we introduce a sequent calculus for QdMTL that successively reduces temporal statements about traces of quantified hybrid programs to non-temporal Q$\mathsf{d}\mathcal{L}$ formulas. In this way, we make the intuition formally precise that metric temporal properties can be checked by augmenting proofs with appropriate assertions about intermediate states. Like in [22, 23], our calculus works compositionally. It decomposes correctness statements about quantified hybrid programs structurally into corresponding statements about its parts by symbolic transformation.

Our approach combines the advantages of Q$\mathsf{d}\mathcal{L}$ in reasoning about the behaviour of operational distributed hybrid system models with those of metric temporal logic to verify temporal statements about traces. Our first contribution is the logic QdMTL, which provides a coherent foundation for reasoning about the metric temporal behaviour of operational models of distributed hybrid systems. The main contribution is our calculus for deductively verifying temporal statements about distributed hybrid systems, which is a complete axiomatization relative to non-temporal Q$\mathsf{d}\mathcal{L}$.

2 Related Work

Multi-party distributed control has been suggested for car control [15] and air traffic control [7]. Due to limits in verification technology, no formal analysis of metric temporal statements about the distributed hybrid dynamics has been possible for these systems yet. Analysis results include discrete message handling [15] or collision avoidance for two participants [7].

The importance of understanding dynamic/reconfigurable distributed hybrid systems was recognized in modeling languages SHIFT [6] and R-Charon [17]. They focused on simulation/-compilation [6] or the development of a semantics [17], so that no verification is possible yet.

A variety of simulation-based approaches, like [10, 11, 18, 20, 27], have been proposed for the verification or falsification of metric temporal logic properties in hybrid systems, which, however, cannot be used to verify temporal properties in distributed hybrid systems.

Our approach is completely different. It is based on first-order structures and dynamic logic. We focus on developing a logic that supports metric temporal, generic temporal, and non-temporal statements about distributed hybrid dynamics and is amenable to automated theorem proving in the logic itself.

Based on [25], Beckert and Schlager [3] added separate trace modalities to dynamic logic and presented a relatively complete calculus. Their approach only handles discrete state spaces and conventional temporal modalities. In contrast, QdMTL works for hybrid programs with continuous and structural/dimensional dynamics and metric temporal quantifiers.

Davoren and Nerode [5] extended the propositional modal μ-calculus with a semantics in hybrid systems and examine topological aspects. In [4], Davoren et al. gave a semantics in terms of general flow systems for a generalisation of CTL* [9]. In both cases, the authors of [5] and [4] provided Hilbert-style calculi to prove formulas that are valid for all systems simultaneously using abstract actions.

The strength of our logic primarily is that it is a first-order quantified dynamic logic: it handles actual quantified hybrid programs rather than only abstract actions of unknown effect. Our calculus directly supports verification of quantified hybrid programs with first-order definable flows and structures.

3 Syntax of Quantified Differential Metric Temporal Dynamic Logic

As a formal logic for verifying metric temporal specifications of distributed hybrid systems, we introduce *quantified differential metric temporal dynamic logic* (QdMTL). QdMTL extends dynamic logic for reasoning about system runs [13] with many-sorted first-order logic for reasoning about all ($\forall i : A\ \phi$) or some ($\exists i : A\ \phi$) objects of a sort A, e.g., the sort of all aircraft, and three other concepts:

Quantified hybrid programs. The behavior of distributed hybrid systems can be described by quantified hybrid programs [22, 23], which generalize regular programs from dynamic logic [13] to distributed hybrid changes. The distinguishing feature of quantified hybrid programs is that they provide uniform discrete transitions, continuous evolutions, and structural/dimension changes along quantified assignments and quantified differential equations, which can be combined by regular control operations.

Modal operators. Modalities of dynamic logic express statements about all possible behavior ($[\alpha]\pi$) of a system α, or about the existence of a trace ($\langle\alpha\rangle\pi$), satisfying condition π. Unlike in standard dynamic logic, α is a model of a distributed hybrid system. We use quantified hybrid programs to describe α as in [22, 23]. Yet, unlike in standard dynamic logic [13] or quantified differential dynamic logic (Qd$\mathcal{L}$) [22, 23], π is a *trace formula* in QdMTL, and π can refer to every state that occur *during* an arbitrary time interval of a trace using metric temporal operators.

Metric temporal operators. For QdMTL, the metric temporal trace formula $\Box_{\mathcal{I}}\,\phi$ expresses that the formula ϕ holds all along the time interval $\mathcal{I}$ of a trace selected by $[\alpha]$ or $\langle\alpha\rangle$. For instance, the state formula $\langle\alpha\rangle\Box_{\mathcal{I}}\,\phi$ says that the state formula ϕ holds at every state within the time interval $\mathcal{I}$ of at least one trace of α. Dually, the trace formula $\Diamond_{\mathcal{I}}\,\phi$ expresses that ϕ holds at some point during the time interval $\mathcal{I}$ of such a trace . It can occur in a state formula $\langle\alpha\rangle\Diamond_{\mathcal{I}}\,\phi$ to express that there is such a state in the time interval $\mathcal{I}$ of some trace of α, or as $[\alpha]\Diamond_{\mathcal{I}}\,\phi$ to say that along each trace there is a state within the time interval $\mathcal{I}$ satisfying ϕ. In this paper, the primary focus of attention is on homogeneous combination of path and trace quantifiers like $[\alpha]\Box_{\mathcal{I}}\,\phi$ or $\langle\alpha\rangle\Diamond_{\mathcal{I}}\,\phi$.

3.1 Quantified Hybrid Programs

We let $\mathbb{R}$ (resp. $\mathbb{R}_{\geq 0}$, $\mathbb{N}$) denote the set of reals (resp. non-negative reals, non-negative integers). An *interval* of $\mathbb{R}$ (resp. $\mathbb{R}_{\geq 0}$, $\mathbb{N}$) is a convex subset of $\mathbb{R}$ (resp. $\mathbb{R}_{\geq 0}$, $\mathbb{N}$). QdMTL supports a (finite) number of object *sorts*, e.g., the sort of all aircraft, or the sort of all cars. For continuous quantities of distributed hybrid systems like positions or velocities, we add the sort $\mathbb{R}$ for real numbers. *Terms* of QdMTL are built from a set of (sorted) function/variable symbols as in many-sorted first-order logic. For representing appearance and disappearance of objects while running quantified hybrid programs, we use an existence function symbol $\mathsf{E}(\cdot)$ that has value $\mathsf{E}(o) = 1$ if object o exists, and has value $\mathsf{E}(o) = 0$ when object o disappears or has not been created yet. We use 0, 1, $+$, $-$, $\cdot$, $/$with the usual notation and fixed semantics for real arithmetic. For $n \geq 0$ we abbreviate $f(s_1, \dots, s_n)$ by $f(\boldsymbol{s})$ using vectorial notation and we use $\boldsymbol{s} = \boldsymbol{t}$ for element-wise equality.

As a system model for distributed hybrid systems, QdMTL uses *quantified hybrid programs* (QHP) [22, 23]. The quantified hybrid programs occurring in dynamic modalities of QdMTL are regular programs from dynamic logic [13] to which quantified assignments and quantified differential equation systems for distributed hybrid dynamics are added. QHPs are defined by the following grammar (α, β are QHPs, θ a term, i a variable of sort A, f is a function symbol, $\boldsymbol{s}$ is a vector of terms with sorts compatible to f, and χ is a formula of first-order logic):

$$\alpha, \beta ::= \forall i : A\ f(\boldsymbol{s}) := \theta \mid \forall i : A\ f(\boldsymbol{s})' = \theta \,\&\, \chi \mid ?\chi \mid \alpha \cup \beta \mid \alpha;\beta \mid \alpha^*$$

The effect of *quantified assignment* $\forall i : A\ f(\boldsymbol{s}) := \theta$ is an instantaneous discrete jump assigning θ to $f(\boldsymbol{s})$ simultaneously for all objects i of sort A. The quantified hybrid program $\forall i : C\ a(i) := a(i) + 1$, for example, expresses that all cars i of sort C simultaneously increase their acceleration $a(i)$. The effect of *quantified differential equation* $\forall i : A\ f(\boldsymbol{s})' = \theta \,\&\, \chi$ is a continuous evolution where, for all objects i of sort A, all differential equations $f(\boldsymbol{s})' = \theta$ hold and formula χ holds throughout the evolution (the state remains in the region described by χ). The dynamics of QHPs changes the interpretation of terms over time: $f(\boldsymbol{s})'$ is intended to denote the derivative of the interpretation of the term $f(\boldsymbol{s})$ over time during continuous evolution, not the derivative of $f(\boldsymbol{s})$ by its argument $\boldsymbol{s}$. For $f(\boldsymbol{s})'$ to be defined, we assume f is an $\mathbb{R}$-valued function symbol. For simplicity, we assume that f does not occur in $\boldsymbol{s}$. In most quantified assignments/differential equations $\boldsymbol{s}$ is just i. For instance, the following QHP expresses that all cars i of sort C drive by $\forall i : C\ x(i)'' = a(i)$ such that their position $x(i)$ changes continuously according to their respective acceleration $a(i)$. Time itself is implicit. If a clock variable c is needed in a QHP, it can be axiomatized by $c' = 1$, which is equivalent to $\forall i : A\ c' = 1$ where i does not occur in c. For such *vacuous quantification* (i does not occur anywhere), we may omit $\forall i : A$ from assignments and differential equations.

The effect of test $?\chi$ is a *skip* (i.e., no change) if formula χ is true in the current state and *abort* (blocking the system run by a failed assertion), otherwise. *Nondeterministic choice* $\alpha\cup\beta$ is for alternatives in the behavior of the distributed hybrid system. In the *sequential composition* $\alpha;\beta$, QHP β starts after α finishes (β never starts if α continues indefinitely). *Nondeterministic repetition* α^* repeats α an arbitrary number of times, possibly zero times.

Structural dynamics of distributed hybrid systems corresponds to quantified assignments to function terms and we model the appearance of new participants in the distributed hybrid system, e.g., new aircraft appearing into the local flight scenario, by a program $n := \mathsf{new}\ A$ (see [22, 23] for details).

3.2 State and Trace Formulas

The formulas of QdMTL are defined similarly to first-order dynamic logic plus many-sorted first-order logic. However, the modalities $[\alpha]$ and $\langle\alpha\rangle$ accept trace formulas that refer to the temporal behavior of *all* states within a time interval $\mathcal{I}$ along a trace. Inspired by CTL and CTL* [8, 9], we distinguish between state formulas, which are true or false in states, and trace formulas, which are true or false for system traces.

The *state formulas* of QdMTL are defined by the following grammar (ϕ, ψ are state formulas, π is a trace formula, θ_1, θ_2 are terms of the same sort, i is a variable of sort A, and α is a QHP):

$$\phi, \psi ::= \theta_1 = \theta_2 \mid \theta_1 \geq \theta_2 \mid \neg\phi \mid \phi \wedge \psi \mid \forall i : A\ \phi \mid \exists i : A\ \phi \mid [\alpha]\pi \mid \langle\alpha\rangle\pi$$

We use standard abbreviations to define $\leq, >, <, \vee, \rightarrow$. Sorts $A \neq \mathbb{R}$ have no ordering and only $\theta_1 = \theta_2$ is allowed. For sort $\mathbb{R}$, we abbreviate $\forall x : \mathbb{R}\ \phi$ by $\forall x \phi$.

The *trace formulas* of QdMTL are defined by the following grammar (π is a trace formula, ϕ is a state formula, and $\mathcal{I}$ is an interval of $\mathbb{R}_{\geq 0}$):

$$\pi ::= \phi \mid \Box_{\mathcal{I}}\, \phi \mid \Diamond_{\mathcal{I}}\, \phi$$

We omit the time constraint on temporal operators $\Box$ and $\Diamond$ when $[0, +\infty)$ is assumed. Thus the metric temporal operators $\Box$ and $\Diamond$ coincide with the conventional *always* and *eventually* temporal operators of CTL.

Formulas with only conventional temporal operators $\Box$ and $\Diamond$, i.e., metric temporal operators $\Box_{[0,+\infty)}$ and $\Diamond_{[0,+\infty)}$, are called *non-metric temporal* formulas. Formulas without metric temporal operators $\Box_{\mathcal{I}}$ and $\Diamond_{\mathcal{I}}$ are called *non-temporal* Qd$\mathcal{L}$ *formulas* [22, 23]. Unlike in CTL, state formulas are true on a trace if they hold for the *last* state of a trace, not for the first. Thus, $[\alpha]\phi$ expresses that ϕ is true at the end of each trace of α. In contrast, $[\alpha]\Box_{\mathcal{I}}\, \phi$ expresses that ϕ is true all along all states within $\mathcal{I}$ of every trace of α. This combination gives a smooth embedding of non-temporal Qd$\mathcal{L}$ into QdMTL and makes it possible to define a compositional calculus. Like CTL, QdMTL allows nesting with a branching time semantics [8], e.g., $[\alpha]\Box_{[0,4)}\, ((\forall i : C\ x(i) \geq 5) \rightarrow \langle\beta\rangle\Diamond_{(2,3]}\, (\forall i : C\ x(i) \leq 1))$. In the following, all formulas and terms have to be well-typed. For short notation, we allow conditional terms of the form $\mathsf{if}\, \phi\, \mathsf{then}\, \theta_1\, \mathsf{else}\, \theta_2\, \mathsf{fi}$ (where θ_1 and θ_2 have the same sort). This term evaluates to θ_1 if the formula ϕ is true and to θ_2 otherwise. We consider formulas with conditional terms as abbreviations, e.g., $\psi(\mathsf{if}\, \phi\, \mathsf{then}\, \theta_1\, \mathsf{else}\, \theta_2)$ for $(\phi \rightarrow \psi(\theta_1)) \wedge (\neg\phi \rightarrow \psi(\theta_2))$.

Example 1. *Let C be the sort of all cars. By $x(i)$, we denote the position of car i, by $v(i)$ its velocity and by $a(i)$ its acceleration. Then the* QdMTL *formula*

$$(\forall i : C\ x(i) \geq 0) \rightarrow [\forall i : C\ x(i)' = v(i), v(i)' = a(i)\ \&\ v(i) \geq 0]\Box_{[0,2]}\, (\forall i : C\ x(i) \geq 0)$$

says that, if all cars start at a point to the right of the origin and we only allow them to evolve as long as all of them have nonnegative velocity, then they will always *stay up to the right of the origin within 2 time units. In this case, the QHP just consists of a quantified differential equation expressing that the position $x(i)$ of car i evolves over time according to the velocity $v(i)$, which evolves according to its acceleration $a(i)$. The constraint $v(i) \geq 0$ expresses that the cars never move backwards, which otherwise would happen eventually in the case of braking $a(i) < 0$. This formula is indeed valid, and we would be able to use the techniques developed in this paper to prove it.*

4 Semantics of Quantified Differential Metric Temporal Dynamic Logic

In standard dynamic logic [13] and the logic Qd$\mathcal{L}$ [22, 23], modalities only refer to the final states of system runs and the semantics is a reachability relation on states: State τ is reachable from state σ using α if there is a run of α which terminates in τ when started in σ. For QdMTL, however, formulas can refer to intermediate states in any time interval along runs as well. Thus, the behavior of distributed hybrid systems is modeled in terms of *hybrid traces.* Hybrid traces describe the evolution of system variables in every point of time. Such evolution is allowed to have discontinuous points corresponding to changes caused by discrete jumps in state space and continuous phases governed by different differential equations with different slopes.

4.1 Trace Semantics of Quantified Hybrid Programs

A *state* σ associates an infinite set $\sigma(A)$ of objects with each sort A, and it associates a function $\sigma(f)$ of appropriate type with each function symbol f, including $\mathsf{E}(\cdot)$. For simplicity, σ also associates a value $\sigma(i)$ of appropriate type with each variable i. The interpretation of $0, 1, +, -, \cdot, /$ is that of real arithmetic. We assume constant domain for each sort A: all states σ, τ share the same infinite domains $\sigma(A) = \tau(A)$. Sorts $A \neq C$ are disjoint: $\sigma(A) \cap \sigma(C) = \emptyset$. The set of all states is denoted by $\mathcal{S}$. The state σ_i^e agrees with σ except for the interpretation of variable i, which is changed to e. In addition, we distinguish a state Λ to denote the failure of a system run when it is *aborted* due to a test $?\chi$ that yields *false.* In particular, Λ can only occur at the end of an aborted system run and marks that there is no further extension. Given an interval I of $\mathbb{R}$ and a real number t, we write $I + t$ and $I - t$ for the intervals $\{t' \in \mathbb{R} \mid t' - t \in I\}$ and $\{t' \in \mathbb{R} \mid t' + t \in I\}$, respectively, and we denote with $le(I)$ and $ue(I)$ the lower and upper endpoints of I, respectively. Given an interval $I = [t, t']$ (resp. $I = (t, t']$, $I = [t, t')$, $I = (t, t')$, $I = [t, +\infty)$, $I = (t, +\infty)$) of $\mathbb{R}$ and a variable x of sort $\mathbb{R}$, we write $x \in I$ for the formula $x \geq t \wedge x \leq t'$ (resp. $x > t \wedge x \leq t'$, $x \geq t \wedge x < t'$, $x > t \wedge x < t'$, $x \geq t$, $x > t$).

Definition 2 (Hybrid Trace). *A* hybrid trace *is either an infinite sequence* $\langle \overline{\nu}, \overline{I} \rangle = (\langle \nu_0, I_0 \rangle, \langle \nu_1, I_1 \rangle, \langle \nu_2, I_2 \rangle, \ldots)$ *such that*

- $\nu_i : [0, r_i] \to \mathcal{S}$ *is a function with duration* $r_i \in \mathbb{R}_{\geq 0}$ *for all* $i \in \mathbb{N}$,
- I_i *is a closed interval of* $\mathbb{R}_{\geq 0}$ ($I_i = [t, t']$ *for some* $t, t' \in \mathbb{R}_{\geq 0}$ *with* $t \leq t'$) *for all* $i \in \mathbb{N}$,
- $le(I_i) = \sum_{k=0}^{i-1} r_k$ *for all* $i > 0$ *and* $ue(I_i) = \sum_{k=0}^{i} r_k$ *for all* $i \in \mathbb{N}$,
- $ue(I_i) = le(I_{i+1})$ *for all* $i \in \mathbb{N}$,
- *the intervals cover* $\mathbb{R}_{\geq 0}$ *: every non-negative real belongs to some interval* I_i (*thus* $le(I_0) = 0$),

or a finite sequence $\langle \overline{\nu}, \overline{I} \rangle = (\langle \nu_0, I_0 \rangle, \langle \nu_1, I_1 \rangle, \ldots, \langle \nu_n, I_n \rangle)$ *such that*

- $\nu_i : [0, r_i] \to \mathcal{S}$ *is a function with duration* $r_i \in \mathbb{R}_{\geq 0}$ *for all* $0 \leq i \leq n$,
- I_i *is an interval of* $\mathbb{R}_{\geq 0}$ *for all* $0 \leq i \leq n$,
- *for all* $0 \leq i < n$, $I_i = [t, t']$ *for some* $t, t' \in \mathbb{R}_{\geq 0}$ *with* $t \leq t'$,
- $le(I_i) = \sum_{k=0}^{i-1} r_k$ *for all* $0 < i \leq n$ *and* $ue(I_i) = \sum_{k=0}^{i} r_k$ *for all* $0 \leq i < n$,
- $ue(I_i) = le(I_{i+1})$ *for all* $0 \leq i < n$,
- *the intervals cover* $\mathbb{R}_{\geq 0}$ *: every non-negative real belongs to some interval* I_i (*thus* $le(I_0) = 0$ *and* $I_n = [le(I_n), +\infty)$).

Further, for a state $\sigma \in \mathcal{S}$, $\hat{\sigma} : 0 \mapsto \sigma$ is the point flow *at σ with duration 0. A hybrid trace* terminates *if it is a finite sequence $(\langle \nu_0, I_0\rangle, \langle \nu_1, I_1\rangle, \ldots, \langle \nu_n, I_n\rangle)$ and $\nu_n(r_n) \neq \Lambda$. In that case, the last state $\nu_n(r_n)$ is denoted by* last $\langle \overline{\nu}, \overline{I}\rangle$ *and the total duration $\sum_{i=0}^{n} r_i$ is denoted by $\mathcal{D}(\langle \overline{\nu}, \overline{I}\rangle)$. The first state $\nu_0(0)$ is denoted by* first $\langle \overline{\nu}, \overline{I}\rangle$*. For a hybrid trace $\langle \overline{\nu}, \overline{I}\rangle$ and a time point $t \in \mathbb{R}_{\geq 0}$, the set of states of $\langle \overline{\nu}, \overline{I}\rangle$ at t, denoted by $\langle \overline{\nu}, \overline{I}\rangle(t)$, is*

$$\begin{cases} \{\nu_k(\zeta) \mid t \in I_k \text{ and } le(I_k) + \zeta = t\} & \text{if } \langle \overline{\nu}, \overline{I}\rangle \text{ is a finite hybrid trace } (\langle \nu_0, I_0\rangle, \\ & \langle \nu_1, I_1\rangle, \ldots, \langle \nu_n, I_n\rangle) \text{ and } t \leq \mathcal{D}(\langle \overline{\nu}, \overline{I}\rangle) \\ \{\text{last } \langle \overline{\nu}, \overline{I}\rangle\} & \text{if } \langle \overline{\nu}, \overline{I}\rangle \text{ is a finite hybrid trace } (\langle \nu_0, I_0\rangle, \\ & \langle \nu_1, I_1\rangle, \ldots, \langle \nu_n, I_n\rangle) \text{ and } t > \mathcal{D}(\langle \overline{\nu}, \overline{I}\rangle) \\ \{\nu_k(\zeta) \mid t \in I_k \text{ and } le(I_k) + \zeta = t\} & \text{if } \langle \overline{\nu}, \overline{I}\rangle \text{ is an infinite hybrid trace} \end{cases}$$

Unlike in [1, 14], the definition of traces also admits finite traces of bounded duration, which is necessary for compositionality of traces in $\alpha; \beta$. The semantics of quantified hybrid programs α as the set $\mu(\alpha)$ of its possible traces depends on valuations $\sigma[\![\cdot]\!]$ of formulas and terms at intermediate states σ. The valuation of formulas will be defined in Definition 4. Especially, we use $\sigma_i^e[\![\cdot]\!]$ to denote the valuations of terms and formulas in state σ_i^e, i.e., in state σ with i interpreted as e.

Definition 3 (Trace Semantics of Quantified Hybrid Programs)**.** *The* trace semantics, $\mu(\alpha)$, of a quantified hybrid program α, *is the set of all its possible hybrid traces and is defined inductively as follows:*

1. $\mu(\forall i : A\ f(\boldsymbol{s}) := \theta) = \{(\langle \hat{\sigma}, [0, 0]\rangle, \langle \hat{\tau}, [0, +\infty)\rangle) : \sigma \in \mathcal{S}$ *and state τ is identical to σ except that at each position $\boldsymbol{o}$ of f: if $\sigma_i^e[\![\boldsymbol{s}]\!] = \boldsymbol{o}$ for some object $e \in \sigma(A)$, then $\tau(f)(\sigma_i^e[\![\boldsymbol{s}]\!]) = \sigma_i^e[\![\theta]\!]$.*$\}$

2. $\mu(\forall i : A\ f(\boldsymbol{s})' = \theta \,\&\, \chi) = \{(\langle \varphi, [0, +\infty)\rangle) : 0 \leq r \in \mathbb{R}$ *and $\varphi : [0, r] \to \mathcal{S}$ is a function satisfying the following conditions. At each time $t \in [0, r]$, state $\varphi(t)$ is identical to $\varphi(0)$, except that at each position $\boldsymbol{o}$ of f : if $\sigma_i^e[\![\boldsymbol{s}]\!] = \boldsymbol{o}$ for some object $e \in \sigma(A)$, then, at each time $\zeta \in [0, r]$:*

   - *The differential equations hold and derivatives exist (trivial for $r = 0$) :*

   $$\frac{\mathsf{d}(\varphi(t)_i^e[\![f(\boldsymbol{s})]\!])}{\mathsf{d}t}(\zeta) = (\varphi(\zeta)_i^e[\![\theta]\!])$$

   - *The evolution domains is respected: $\varphi(\zeta)_i^e[\![\chi]\!] = true$.*$\}$

3. $\mu(?\chi) = \{(\langle \hat{\sigma}, [0, +\infty)\rangle) : \sigma[\![\chi]\!] = true\} \cup \{(\langle \hat{\sigma}, [0, 0]\rangle, \langle \hat{\Lambda}, [0, +\infty)\rangle) : \sigma[\![\chi]\!] = false\}$

4. $\mu(\alpha \cup \beta) = \mu(\alpha) \cup \mu(\beta)$

5. $\mu(\alpha; \beta) = \{\langle \overline{\nu}, \overline{I}\rangle \circ \langle \overline{\varsigma}, \overline{J}\rangle : \langle \overline{\nu}, \overline{I}\rangle \in \mu(\alpha), \langle \overline{\varsigma}, \overline{J}\rangle \in \mu(\beta)$ *when $\langle \overline{\nu}, \overline{I}\rangle \circ \langle \overline{\varsigma}, \overline{J}\rangle$ is*

defined}; the composition of $\langle\overline{\nu},\overline{I}\rangle$ *and* $\langle\overline{\varsigma},\overline{J}\rangle$*, denoted by* $\langle\overline{\nu},\overline{I}\rangle\circ\langle\overline{\varsigma},\overline{J}\rangle$*, is*

$$\begin{cases} \begin{array}{l}(\langle\nu_0,I_0\rangle,\ldots,\langle\nu_n,[\mathrm{le}(I_n),\mathcal{D}(\langle\overline{\nu},\overline{I}\rangle)]\rangle,\\ \langle\varsigma_0,J_0+\mathcal{D}(\langle\overline{\nu},\overline{I}\rangle)\rangle,\ldots,\langle\varsigma_m,J_m+\mathcal{D}(\langle\overline{\nu},\overline{I}\rangle)\rangle)\end{array} & \begin{array}{l}\textit{if } \langle\overline{\nu},\overline{I}\rangle=(\langle\nu_0,I_0\rangle,\ldots,\langle\nu_n,I_n\rangle)\\ \textit{terminates at } \langle\nu_n,I_n\rangle,\ \langle\overline{\varsigma},\overline{J}\rangle=\\ (\langle\varsigma_0,J_0\rangle,\ldots,\langle\varsigma_m,J_m\rangle),\ \textit{and}\\ \mathrm{last}\ \langle\overline{\nu},\overline{I}\rangle=\mathrm{first}\ \langle\overline{\varsigma},\overline{J}\rangle\end{array}\\ \\ \begin{array}{l}(\langle\nu_0,I_0\rangle,\ldots,\langle\nu_n,[\mathrm{le}(I_n),\mathcal{D}(\langle\overline{\nu},\overline{I}\rangle)]\rangle,\\ \langle\varsigma_0,J_0+\mathcal{D}(\langle\overline{\nu},\overline{I}\rangle)\rangle,\langle\varsigma_1,J_1+\mathcal{D}(\langle\overline{\nu},\overline{I}\rangle)\rangle,\ldots)\end{array} & \begin{array}{l}\textit{if } \langle\overline{\nu},\overline{I}\rangle=(\langle\nu_0,I_0\rangle,\ldots,\langle\nu_n,I_n\rangle)\\ \textit{terminates at } \langle\nu_n,I_n\rangle,\ \langle\overline{\varsigma},\overline{J}\rangle=\\ (\langle\varsigma_0,J_0\rangle,\langle\varsigma_1,J_1\rangle,\ldots),\ \textit{and}\\ \mathrm{last}\ \langle\overline{\nu},\overline{I}\rangle=\mathrm{first}\ \langle\overline{\varsigma},\overline{J}\rangle\end{array}\\ \\ \langle\overline{\nu},\overline{I}\rangle & \textit{if } \langle\overline{\nu},\overline{I}\rangle \textit{ does not terminate}\\ \\ \textit{not defined} & \textit{otherwise}\end{cases}$$

6. $\mu(\alpha^*)=\bigcup_{n\in\mathbb{N}}\mu(\alpha^n)$*, where* $\alpha^{n+1}:=(\alpha^n;\alpha)$ *for* $n\geq 1$*, as well as* $\alpha^1:=\alpha$ *and* $\alpha^0:=(?\mathit{true})$*.*

4.2 Valuation of State and Trace Formulas

Definition 4 (Valuation of Formulas)**.** *The valuation of state and trace formulas is defined respectively. For state formulas, the* valuation $\sigma[\![\cdot]\!]$ *with respect to state* σ *is defined as follows:*

1. $\sigma[\![\theta_1=\theta_2]\!]=\mathit{true}$ *iff* $\sigma[\![\theta_1]\!]=\sigma[\![\theta_2]\!]$*; accordingly for* $\geq$*.*
2. $\sigma[\![\phi\wedge\psi]\!]=\mathit{true}$ *iff* $\sigma[\![\phi]\!]=\mathit{true}$ *and* $\sigma[\![\psi]\!]=\mathit{true}$*; accordingly for* $\neg$*.*
3. $\sigma[\![\forall i:A\ \phi]\!]=\mathit{true}$ *iff* $\sigma_i^e[\![\phi]\!]=\mathit{true}$ *for all objects* $e\in\sigma(A)$*.*
4. $\sigma[\![\exists i:A\ \phi]\!]=\mathit{true}$ *iff* $\sigma_i^e[\![\phi]\!]=\mathit{true}$ *for some object* $e\in\sigma(A)$*.*
5. $\sigma[\![[\alpha]\pi]\!]=\mathit{true}$ *iff for each hybrid trace* $\langle\overline{\nu},\overline{I}\rangle\in\mu(\alpha)$ *that starts in* first $\langle\overline{\nu},\overline{I}\rangle=\sigma$*, if* $\langle\overline{\nu},\overline{I}\rangle[\![\pi]\!]$ *is defined, then* $\langle\overline{\nu},\overline{I}\rangle[\![\pi]\!]=\mathit{true}$*.*
6. $\sigma[\![\langle\alpha\rangle\pi]\!]=\mathit{true}$ *iff there is a hybrid trace* $\langle\overline{\nu},\overline{I}\rangle\in\mu(\alpha)$ *starting in* first $\langle\overline{\nu},\overline{I}\rangle=\sigma$ *such that* $\langle\overline{\nu},\overline{I}\rangle[\![\pi]\!]$ *is defined and* $\langle\overline{\nu},\overline{I}\rangle[\![\pi]\!]=\mathit{true}$*.*

For trace formulas, the valuation $\langle\overline{\nu},\overline{I}\rangle[\![\cdot]\!]$ *with respect to hybrid trace* $\langle\overline{\nu},\overline{I}\rangle$ *is defined inductively as follows:*

1. *If* ϕ *is a state formula, then* $\langle\overline{\nu},\overline{I}\rangle[\![\phi]\!]=\mathrm{last}\ \langle\overline{\nu},\overline{I}\rangle[\![\phi]\!]$ *if* $\langle\overline{\nu},\overline{I}\rangle$ *terminates, whereas* $\langle\overline{\nu},\overline{I}\rangle[\![\phi]\!]$ *is not defined if* $\langle\overline{\nu},\overline{I}\rangle$ *does not terminate.*
2. $\langle\overline{\nu},\overline{I}\rangle[\![\Box_{\mathcal{I}}\ \phi]\!]=\mathit{true}$ *iff* $\sigma[\![\phi]\!]=\mathit{true}$ *for every time point* $t\in\mathcal{I}$ *and every state* $\sigma\in\langle\overline{\nu},\overline{I}\rangle(t)$ *with* $\sigma\neq\Lambda$*.*
3. $\langle\overline{\nu},\overline{I}\rangle[\![\Diamond_{\mathcal{I}}\ \phi]\!]=\mathit{true}$ *iff* $\sigma[\![\phi]\!]=\mathit{true}$ *for some time point* $t\in\mathcal{I}$ *and some state* $\sigma\in\langle\overline{\nu},\overline{I}\rangle(t)$ *with* $\sigma\neq\Lambda$*.*

As usual, a (state) formula is *valid* if it is true in all states. Further for (state) formula ϕ and state σ we write $\sigma\models\phi$ iff $\sigma[\![\phi]\!]=\mathit{true}$. We write $\sigma\not\models\phi$ iff $\sigma[\![\phi]\!]=\mathit{false}$. Likewise, for trace formula π and hybrid trace $\langle\overline{\nu},\overline{I}\rangle$ we write $\langle\overline{\nu},\overline{I}\rangle\models\pi$ iff $\langle\overline{\nu},\overline{I}\rangle[\![\pi]\!]=\mathit{true}$ and $\langle\overline{\nu},\overline{I}\rangle\not\models\pi$ iff $\langle\overline{\nu},\overline{I}\rangle[\![\pi]\!]=\mathit{false}$. In particular, we only write $\langle\overline{\nu},\overline{I}\rangle\models\pi$ or $\langle\overline{\nu},\overline{I}\rangle\not\models\pi$ if $\langle\overline{\nu},\overline{I}\rangle[\![\pi]\!]$ is defined, which it is not the case if π is a state formula and $\langle\overline{\nu},\overline{I}\rangle$ does not terminate.

4.3 Conservative Temporal Extension

The following result shows that the extension of QdMTL by metric temporal operators does not change the meaning of non-temporal formulas. The trace semantics given in Definition 4 is equivalent to the final state reachability relation semantics [22, 23] for the sublogic Qd$\mathcal{L}$ of QdMTL.

Proposition 5. *The logic* QdMTL *is a* conservative extension *of non-temporal* Qd$\mathcal{L}$*, i.e., the set of valid* Qd$\mathcal{L}$ *formulas is the same with respect to transition reachability semantics of* Qd$\mathcal{L}$ *as with respect to the trace semantics of* QdMTL*.*

5 Proof Calculus for Metric Temporal Properties

In this section, we introduce a sequent calculus for verifying metric temporal specifications of distributed hybrid systems in QdMTL. With the basic idea being to perform a symbolic decomposition, the calculus transforms quantified hybrid programs successively into simpler logical formulas describing their effects. Statements about the metric temporal behavior of a quantified hybrid program are first converted to equivalent non-metric temporal statements without quantitative time constraints and then reduced to corresponding non-temporal statements about the intermediate states.

5.1 Proof Rules

In Fig. 1, we present a proof calculus for QdMTL that inherits the proof rules of Qd$\mathcal{L}$ from [22, 23] and adds new proof rules for metric temporal modalities. We use the sequent notation informally for a systematic proof structure. A *sequent* is of the form $\Gamma \to \Delta$, where the *antecedent* Γ and *succedent* Δ are finite sets of formulas. The semantics of $\Gamma \to \Delta$ is that of the formula $\bigwedge_{\phi\in\Gamma} \phi \to \bigvee_{\psi\in\Delta} \psi$ and will be treated as an abbreviation. As usual in sequent calculus, the proof rules are applied backwards from the *conclusion* (goal below horizontal bar) to the *premises* (subgoals above bar).

Definition 6. *Let α be a quantified hybrid program and c a clock variable, i,e., a variable of sort $\mathbb{R}$ changing with constant slope 1 (along quantified differential equation $c' = 1$, which is equivalent to $\forall i : A\ c' = 1$ where i does not occur in c). Then $\alpha \oplus c' = 1$ is defined inductively as follows:*

1. $\forall i : A\ f(\boldsymbol{s}) := \theta \oplus c' = 1 := \forall i : A\ f(\boldsymbol{s}) := \theta$
2. $\forall i : A\ f(\boldsymbol{s})' = \theta \,\&\, \chi \oplus c' = 1 := \forall i : A\ f(\boldsymbol{s})' = \theta, c' = 1 \,\&\, \chi$
3. $?\chi \oplus c' = 1 := ?\chi$
4. $\alpha \cup \beta \oplus c' = 1 := \alpha \oplus c' = 1 \cup \beta \oplus c' = 1$
5. $\alpha; \beta \oplus c' = 1 := \alpha \oplus c' = 1; \beta \oplus c' = 1$
6. $\alpha^* \oplus c' = 1 := (\alpha \oplus c' = 1)^*$.

Inherited Non-temporal Rules. The QdMTL calculus inherits the (non-temporal) Qd$\mathcal{L}$ proof rules. For propositional logic, standard rules *ax*–*cut* are listed in Fig. 1. Rules $[;]$–$\langle ? \rangle$ work similar to those in [13]. Rules $[']$, $\langle ' \rangle$ handle continuous evolutions for quantified differential

$$(ax)\ \frac{}{\phi \to \phi} \qquad (\neg\mathrm{r})\ \frac{\phi \to}{\to \neg\phi} \qquad (\neg\mathrm{l})\ \frac{\to \phi}{\neg\phi \to} \qquad (\wedge\mathrm{r})\ \frac{\to \phi \quad \to \psi}{\to \phi \wedge \psi} \qquad (\wedge\mathrm{l})\ \frac{\phi, \psi \to}{\phi \wedge \psi \to} \qquad (cut)\ \frac{\to \phi \quad \phi \to}{\to}$$

$$([;])\ \frac{[\alpha][\beta]\phi}{[\alpha;\beta]\phi} \qquad (\langle;\rangle)\ \frac{\langle\alpha\rangle\langle\beta\rangle\phi}{\langle\alpha;\beta\rangle\phi} \qquad ([\cup])\ \frac{[\alpha]\phi \wedge [\beta]\phi}{[\alpha \cup \beta]\phi} \qquad (\langle\cup\rangle)\ \frac{\langle\alpha\rangle\phi \vee \langle\beta\rangle\phi}{\langle\alpha \cup \beta\rangle\phi} \qquad ([?])\ \frac{\chi \to \phi}{[?\chi]\phi} \qquad (\langle ? \rangle)\ \frac{\chi \wedge \phi}{\langle ?\chi\rangle\phi}$$

$$([']) \ \frac{\forall t \geq 0((\forall 0 \leq \tilde{t} \leq t[\forall i : A\ \mathcal{S}(\tilde{t})]\chi) \to [\forall i : A\ \mathcal{S}(t)]\phi)}{[\forall i : A\ f(\boldsymbol{s})' = \theta \,\&\, \chi]\phi}\,{}_{1} \qquad (\langle ' \rangle)\ \frac{\exists t \geq 0((\forall 0 \leq \tilde{t} \leq t\langle\forall i : A\ \mathcal{S}(\tilde{t})\rangle\chi) \wedge \langle\forall i : A\ \mathcal{S}(t)\rangle\phi)}{\langle\forall i : A\ f(\boldsymbol{s})' = \theta \,\&\, \chi\rangle\phi}\,{}_{1}$$

$$([:=])\ \frac{\mathsf{if}\ \exists i : A\ \boldsymbol{s} = [\mathcal{A}]\boldsymbol{u}\ \mathsf{then}\ \forall i : A\ (\boldsymbol{s} = [\mathcal{A}]\boldsymbol{u} \to \phi(\theta))\ \mathsf{else}\ \phi(f([\mathcal{A}]\boldsymbol{u}))\ \mathsf{fi}}{\phi([\forall i : A\ f(\boldsymbol{s}) := \theta]f(\boldsymbol{u}))}\,{}_{2}$$

$$(\langle := \rangle)\ \frac{\mathsf{if}\ \exists i : A\ \boldsymbol{s} = \langle\mathcal{A}\rangle\boldsymbol{u}\ \mathsf{then}\ \exists i : A\ (\boldsymbol{s} = \langle\mathcal{A}\rangle\boldsymbol{u} \wedge \phi(\theta))\ \mathsf{else}\ \phi(f(\langle\mathcal{A}\rangle\boldsymbol{u}))\ \mathsf{fi}}{\phi(\langle\forall i : A\ f(\boldsymbol{s}) := \theta\rangle f(\boldsymbol{u}))}\,{}_{2}$$

$$(skip)\ \frac{\Upsilon([\forall i : A\ f(\boldsymbol{s}) := \theta]\boldsymbol{u})}{[\forall i : A\ f(\boldsymbol{s}) := \theta]\Upsilon(\boldsymbol{u})}\,{}_{3} \qquad ([:*])\ \frac{\forall j : A\ \phi(\theta)}{[\forall j : A\ n := \theta]\phi(n)} \qquad (\langle :* \rangle)\ \frac{\exists j : A\ \phi(\theta)}{\langle\forall j : A\ n := \theta\rangle\phi(n)} \qquad (ex)\ \frac{true}{\exists n : A\ \mathsf{E}(n) = 0}$$

$$(\forall\mathrm{r})\ \frac{\Gamma \to \phi(f(X_1, \ldots, X_n)), \Delta}{\Gamma \to \forall x\phi(x), \Delta}\,{}_{4} \qquad (\exists\mathrm{r})\ \frac{\Gamma \to \phi(\theta), \exists x\phi(x), \Delta}{\Gamma \to \exists x\phi(x), \Delta}\,{}_{5} \qquad (\forall\mathrm{l})\ \frac{\Gamma, \phi(\theta), \forall x\phi(x) \to \Delta}{\Gamma, \forall x\phi(x) \to \Delta}\,{}_{5} \qquad (\exists\mathrm{l})\ \frac{\Gamma, \phi(f(X_1, \ldots, X_n)) \to \Delta}{\Gamma, \exists x\phi(x) \to \Delta}\,{}_{4}$$

$$(\mathrm{i}\forall)\ \frac{\mathrm{QE}(\forall X, Y\,(\mathsf{if}\ \boldsymbol{s} = \boldsymbol{t}\ \mathsf{then}\ \Phi(X) \to \Psi(X)\ \mathsf{else}\ \Phi(X) \to \Psi(Y)\ \mathsf{fi}))}{\Phi(f(\boldsymbol{s})) \to \Psi(f(\boldsymbol{t}))}\,{}_{6} \qquad (\mathrm{i}\exists)\ \frac{\mathrm{QE}(\exists X \bigwedge_i (\Phi_i \to \Psi_i))}{\Phi_1 \to \Psi_1 \ \ldots \ \Phi_n \to \Psi_n}\,{}_{7}$$

$$([]gen)\ \frac{\phi \to \psi}{\Gamma, [\alpha]\phi \to [\alpha]\psi, \Delta} \qquad (\langle\rangle gen)\ \frac{\phi \to \psi}{\Gamma, \langle\alpha\rangle\phi \to \langle\alpha\rangle\psi, \Delta} \qquad (ind)\ \frac{\phi \to [\alpha]\phi}{\Gamma, \phi \to [\alpha^*]\phi, \Delta} \qquad (con)\ \frac{v > 0 \wedge \varphi(v) \to \langle\alpha\rangle\varphi(v-1)}{\Gamma, \exists v\, \varphi(v) \to \langle\alpha^*\rangle\exists v \leq 0\, \varphi(v), \Delta}\,{}_{8}$$

$$(DI)\ \frac{\chi \to [\forall i : A\ f(\boldsymbol{s})' := \theta]D(\phi)}{\phi \to [\forall i : A\ f(\boldsymbol{s})' = \theta \,\&\, \chi]\phi}\,{}_{9} \qquad (DC)\ \frac{\phi \to [\forall i : A\ f(\boldsymbol{s})' = \theta \,\&\, \chi]\psi \quad \phi \to [\forall i : A\ f(\boldsymbol{s})' = \theta \,\&\, \chi \wedge \psi]\phi}{\phi \to [\forall i : A\ f(\boldsymbol{s})' = \theta \,\&\, \chi]\phi}$$

$$([\cup]\Box)\ \frac{[\alpha]\Box\phi \wedge [\beta]\Box\phi}{[\alpha \cup \beta]\Box\phi}\,{}_{10} \qquad (\langle\cup\rangle\Diamond)\ \frac{\langle\alpha\rangle\Diamond\phi \vee \langle\beta\rangle\Diamond\phi}{\langle\alpha \cup \beta\rangle\Diamond\phi}\,{}_{11} \qquad ([;]\Box)\ \frac{[\alpha]\Box\phi \wedge [\alpha][\beta]\Box\phi}{[\alpha;\beta]\Box\phi}\,{}_{10}$$

$$(\langle;\rangle\Diamond)\ \frac{\langle\alpha\rangle\Diamond\phi \vee \langle\alpha\rangle\langle\beta\rangle\Diamond\phi}{\langle\alpha;\beta\rangle\Diamond\phi}\,{}_{11} \qquad ([?]\Box)\ \frac{\phi}{[?\chi]\Box\phi}\,{}_{10} \qquad (\langle ? \rangle\Diamond)\ \frac{\phi}{\langle ?\chi\rangle\Diamond\phi}\,{}_{11}$$

$$(['\,]\Box)\ \frac{[\forall i : A\ f(\boldsymbol{s})' = \theta \,\&\, \chi]\phi}{[\forall i : A\ f(\boldsymbol{s})' = \theta \,\&\, \chi]\Box\phi}\,{}_{10} \qquad (\langle ' \rangle\Diamond)\ \frac{\langle\forall i : A\ f(\boldsymbol{s})' = \theta \,\&\, \chi\rangle\phi}{\langle\forall i : A\ f(\boldsymbol{s})' = \theta \,\&\, \chi\rangle\Diamond\phi}\,{}_{11}$$

$$([:=]\Box)\ \frac{\phi \wedge [\forall i : A\ f(\boldsymbol{s}) := \theta]\phi}{[\forall i : A\ f(\boldsymbol{s}) := \theta]\Box\phi}\,{}_{10} \qquad (\langle := \rangle\Diamond)\ \frac{\phi \vee \langle\forall i : A\ f(\boldsymbol{s}) := \theta\rangle\phi}{\langle\forall i : A\ f(\boldsymbol{s}) := \theta\rangle\Diamond\phi}\,{}_{11}$$

$$([^{*n}]\Box)\ \frac{[\alpha;\alpha^*]\Box\phi}{[\alpha^*]\Box\phi}\,{}_{10} \qquad (\langle^{*n}\rangle\Diamond)\ \frac{\langle\alpha;\alpha^*\rangle\Diamond\phi}{\langle\alpha^*\rangle\Diamond\phi}\,{}_{11} \qquad ([^*]\Box)\ \frac{[\alpha^*][\alpha]\Box\phi}{[\alpha^*]\Box\phi}\,{}_{10} \qquad (\langle^*\rangle\Diamond)\ \frac{\langle\alpha^*\rangle\langle\alpha\rangle\Diamond\phi}{\langle\alpha^*\rangle\Diamond\phi}\,{}_{11}$$

$$([\cup]\Box_{\mathcal{I}})\ \frac{[c := 0; \alpha \cup \beta \oplus c' = 1; c' = 1]\Box(c \in \mathcal{I} \to \phi)}{[\alpha \cup \beta]\Box_{\mathcal{I}}\,\phi}\,{}_{10\,12} \qquad (\langle\cup\rangle\Diamond_{\mathcal{I}})\ \frac{\langle c := 0; \alpha \cup \beta \oplus c' = 1; c' = 1\rangle\Diamond(c \in \mathcal{I} \wedge \phi)}{\langle\alpha \cup \beta\rangle\Diamond_{\mathcal{I}}\,\phi}\,{}_{11\,12}$$

$$([;]\Box_{\mathcal{I}})\ \frac{[c := 0; \alpha;\beta \oplus c' = 1; c' = 1]\Box(c \in \mathcal{I} \to \phi)}{[\alpha;\beta]\Box_{\mathcal{I}}\,\phi}\,{}_{10\,12} \qquad ([;]\Diamond_{\mathcal{I}})\ \frac{\langle c := 0; \alpha;\beta \oplus c' = 1; c' = 1\rangle\Diamond(c \in \mathcal{I} \wedge \phi)}{\langle\alpha;\beta\rangle\Diamond_{\mathcal{I}}\,\phi}\,{}_{11\,12}$$

$$([?]\Box_{\mathcal{I}})\ \frac{[c := 0; ?\chi \oplus c' = 1; c' = 1]\Box(c \in \mathcal{I} \to \phi)}{[?\chi]\Box_{\mathcal{I}}\,\phi}\,{}_{10\,12} \qquad (\langle ? \rangle\Diamond_{\mathcal{I}})\ \frac{\langle c := 0; ?\chi \oplus c' = 1; c' = 1\rangle\Diamond(c \in \mathcal{I} \wedge \phi)}{\langle ?\chi\rangle\Diamond_{\mathcal{I}}\,\phi}\,{}_{11\,12}$$

$$(['\,]\Box_{\mathcal{I}})\ \frac{[c := 0; \forall i : A\ f(\boldsymbol{s})' = \theta \,\&\, \chi \oplus c' = 1; c' = 1]\Box(c \in \mathcal{I} \to \phi)}{[\forall i : A\ f(\boldsymbol{s})' = \theta \,\&\, \chi]\Box_{\mathcal{I}}\,\phi}\,{}_{10\,12}$$

$$(\langle ' \rangle\Diamond_{\mathcal{I}})\ \frac{\langle c := 0; \forall i : A\ f(\boldsymbol{s})' = \theta \,\&\, \chi \oplus c' = 1; c' = 1\rangle\Diamond(c \in \mathcal{I} \wedge \phi)}{\langle\forall i : A\ f(\boldsymbol{s})' = \theta \,\&\, \chi\rangle\Diamond_{\mathcal{I}}\,\phi}\,{}_{11\,12}$$

$$([:=]\Box_{\mathcal{I}})\ \frac{[c := 0; \forall i : A\ f(\boldsymbol{s}) := \theta \oplus c' = 1; c' = 1]\Box(c \in \mathcal{I} \to \phi)}{[\forall i : A\ f(\boldsymbol{s}) := \theta]\Box_{\mathcal{I}}\,\phi}\,{}_{10\,12}$$

$$(\langle := \rangle\Diamond_{\mathcal{I}})\ \frac{\langle c := 0; \forall i : A\ f(\boldsymbol{s}) := \theta \oplus c' = 1; c' = 1\rangle\Diamond(c \in \mathcal{I} \wedge \phi)}{\langle\forall i : A\ f(\boldsymbol{s}) := \theta\rangle\Diamond_{\mathcal{I}}\,\phi}\,{}_{11\,12}$$

$$([^*]\Box_{\mathcal{I}})\ \frac{[c := 0; \alpha^* \oplus c' = 1; c' = 1]\Box(c \in \mathcal{I} \to \phi)}{[\alpha^*]\Box_{\mathcal{I}}\,\phi}\,{}_{10\,12} \qquad (\langle^*\rangle\Diamond_{\mathcal{I}})\ \frac{\langle c := 0; \alpha^* \oplus c' = 1; c' = 1\rangle\Diamond(c \in \mathcal{I} \wedge \phi)}{\langle\alpha^*\rangle\Diamond_{\mathcal{I}}\,\phi}\,{}_{11\,12}$$

[1] $t, \tilde{t}$ are new variables, $\forall i : A\ \mathcal{S}(t)$ is the quantified assignment $\forall i : A\ f(\boldsymbol{s}) := y_{\boldsymbol{s}}(t)$ with solutions $y_{\boldsymbol{s}}(t)$ of the (injective) differential equations and $f(\boldsymbol{s})$ as initial values. See [22, 23] for the definition of a *injective* quantified assignment or quantified differential equation.

[2] Occurrence $f(\boldsymbol{u})$ in $\phi(f(\boldsymbol{u}))$ is not in scope of a modality (admissible substitution) and we abbreviate assignment $\forall i : A\ f(\boldsymbol{s}) := \theta$ by $\mathcal{A}$, which is assumed to be injective.

[3] $f \neq \Upsilon$ and the quantified assignment $\forall i : A\ f(\boldsymbol{s}) := \theta$ is injective. The same rule applies for $\langle\forall i : A\ f(\boldsymbol{s}) := \theta\rangle$ instead of $[\forall i : A\ f(\boldsymbol{s}) := \theta]$.

[4] f is a new (Skolem) function and $X_1, \ldots, X_n$ are all free logical variables of $\forall x\,\phi(x)$.

[5] θ is an abbreviate term, often a new logical variable.

[6] X, Y are new variables of sort $\mathbb{R}$. QE needs to be applicable in the premises.

[7] Among all open branches, the free (existential) logical variable X of sort $\mathbb{R}$ only occurs in the branch $\Phi_i \to \Psi_i$. QE needs to be defined for the formula in the premises, especially, no Skolem dependencies on X occur.

[8] logical variable v does not occur in α.

[9] The operator D, as defined in [24], is used to computer syntactic total derivations of formulas algebraically.

[10] $\Box$ is the abbreviation for the metric temporal modality $\Box_{[0,+\infty)}$.

[11] $\Diamond$ is the abbreviation for the metric temporal modality $\Diamond_{[0,+\infty)}$.

[12] c is a new variable of sort $\mathbb{R}$ (clock variable).

Figure 1: Rule schemata of the proof calculus for QdMTL

equations with first-order definable solutions. Rules $[:=]$–$\langle :* \rangle$ handle discrete changes for quantified assignments. Axiom *ex* expresses that, for sort $A \neq \mathbb{R}$, there always is a new object n that has not been created yet ($\mathsf{E}(n) = 0$), because domains are infinite. The quantifier rules ∀r–i∃ combine quantifier handling of many-sorted logic based on instantiation with theory reasoning by *quantifier elimination* (QE) for the theory of reals. The global rules $[]gen$, $\langle\rangle gen$ are Gödel generalization rules and *ind* is an induction schema for loops with *inductive invariant* ϕ [13]. Similarly, *con* generalizes Harel's convergence rule [13] to the hybrid case with decreasing variant φ [21]. *DI* and *DC* are rules for quantified differential equations with *quantified differential invariants* [24].

Metric Temporal Rules. The metric temporal rules $[\cup]_{\Box\mathcal{I}}$–$\langle^*\rangle_{\Diamond\mathcal{I}}$ in Fig. 1 for the QdMTL calculus convert metric temporal statements into equivalent non-metric temporal formulas. This conversion is based on the idea of referring metric temporal constraints to the reading on clocks, i.e. variables changing with constant slope 1 (along quantified differential equation $c' = 1$). Clocks are reset by the quantified assignment $c := 0$, which is equivalent to $\forall i : A\ c := 0$ where i does not occur in c, before quantified hybrid programs start executing. They are used to keep track of the progress of time advancing with constant slope 1.

Non-metric Temporal Rules. The non-metric temporal rules $[\cup]_\Box$–$\langle^*\rangle_\Diamond$ in Fig. 1 for the QdMTL calculus successively transform non-metric temporal specifications of quantified hybrid programs into non-temporal Qd$\mathcal{L}$ formulas. The idea underlying this transformation is to decompose quantified hybrid programs and recursively augment intermediate state transitions with appropriate specifications.

Rule $[\cup]_\Box$ decomposes invariants of $\alpha \cup \beta$ (i.e., $[\alpha \cup \beta]\Box\phi$ holds) into an invariant of α (i.e., $[\alpha]\Box\phi$) and an invariant of β (i.e., $[\beta]\Box\phi$)). The Qd$\mathcal{L}$ rule $[\cup]$ can actually be generalized to apply to formulas of the form $[\alpha \cup \beta]\pi$ where π is an arbitrary trace formula, and not just a state formula as in Qd$\mathcal{L}$. Rule $[;]_\Box$ decomposes invariants of $\alpha;\beta$ (i.e., $[\alpha;\beta]\Box\phi$ holds) into an invariant of α (i.e., $[\alpha]\Box\phi$) and an invariant of β that holds when β is started in *any* final state of α (i.e., $[\alpha]([\beta]\Box\phi)$). Its difference with the Qd$\mathcal{L}$ rule $[;]$ thus is that the QdMTL rule $[;]_\Box$ also checks safety invariant ϕ at the symbolic states in between the execution of α and β, and recursively so because of the temporal modality $\Box$. Rule $[:=]_\Box$ expresses that invariants of quantified assignments need to hold before and after the discrete change (similarly for $[?]_\Box$, except that tests do not lead to a state change, so ϕ holding before the test is all there is to it). Rule $[']_\Box$ can directly reduce invariants of continuous evolutions to non-temporal formulas as restrictions of solutions of quantified differential equations are themselves solutions of different duration and thus already included in the continuous evolutions of $\forall i : A\ f(\boldsymbol{s})' = \theta$. The (optional) iteration rule $[^{*n}]_\Box$ can partially unwind loops. It relies on rule $[;]_\Box$. The dual rules $\langle\cup\rangle_\Diamond$, $\langle;\rangle_\Diamond$, $\langle:=\rangle_\Diamond$, $\langle?\rangle_\Diamond$, $\langle'\rangle_\Diamond$, $\langle^{*n}\rangle_\Diamond$ work similarly.

Rules $[^*]_\Box$ and $\langle^*\rangle_\Diamond$ actually define temporal properties of loops inductively. Rule $[^*]_\Box$ expresses that ϕ holds at all times during repetitions of α (i.e., $[\alpha^*]\Box\phi$) iff, after repeating α any number of times, ϕ holds at all times during one execution of α (i.e., $[\alpha^*]([\alpha]\Box\phi)$). Dually, $\langle^*\rangle_\Diamond$ expresses that α holds at some time during repetitions of α (i.e., $\langle\alpha^*\rangle\Diamond\phi$) iff, after some number of repetitions of α, formula ϕ holds at some point during one execution of α (i.e., $\langle\alpha^*\rangle(\langle\alpha\rangle\Diamond\phi)$). In this context, the non-temporal modality $\langle\alpha^*\rangle$ can be thought of as skipping over to the iteration of α during which ϕ actually occurs, as expressed by the nested QdMTL formula $\langle\alpha\rangle\Diamond\phi$. The inductive definition rules $[^*]_\Box$ and $\langle^*\rangle_\Diamond$ completely reduce temporal properties of loops to QdMTL properties of standard non-temporal Qd$\mathcal{L}$ modalities such that standard induction (*ind*) or convergence (*con*) rules, as listed in Fig. 1, can be used for the outer non-temporal modality of the loop. Hence, after applying the inductive loop definition rules $[^*]_\Box$ and $\langle^*\rangle_\Diamond$, the standard Qd$\mathcal{L}$ loop invariant and variant rules can be used for verifying temporal properties of loops without change, except that the postcondition contains temporal modalities.

5.2 Soundness and Completeness

Soundness. The following result shows that verification with the QdMTL calculus always produces correct results about the metric temporal behavior of distributed hybrid systems, i.e., the QdMTL calculus is sound.

Theorem 7 (Soundness of QdMTL)**.** *The* QdMTL *calculus is sound, i.e., every* QdMTL *(state) formula that can be proven is valid.*

Incompleteness of QdMTL. In [22, 23] it has been shown that the verification problem for distributed hybrid systems has *three independent sources* of undecidability. Both the discrete and continuous fragments of Qd$\mathcal{L}$ are subject to Gödel's incompleteness theorem. The fragment with only structural and dimension-changing dynamics is not effective either, because it can encode two-counter machines. Hence, Qd$\mathcal{L}$ cannot be effectively axiomatizable. Since QdMTL is a conservative extension of Qd$\mathcal{L}$, those results lift to QdMTL. Therefore, the discrete, continuous, and structural fragments of QdMTL, even if only containing non-temporal formulas are non-axiomatizable. In particular, QdMTL is non-axiomatizable.

Relative Completeness. The Qd$\mathcal{L}$ calculus has been proved to be complete relative to the *first-order logic of quantified differential equations* (FOQD), i.e., first-order real arithmetic augmented with formulas expressing properties of quantified differential equations, that is, Qd$\mathcal{L}$ formulas of the form $[\forall i : A\ f(\boldsymbol{s})' = \theta\ \&\ \chi]F$ with a first-order formula F [23]. Due to the modular construction of the QdMTL calculus, we can lift the relative completeness result from Qd$\mathcal{L}$ to QdMTL. We essentially show that QdMTL is complete relative to Qd$\mathcal{L}$, which directly implies that QdMTL is even complete relative to FOQD. Again, we restrict our attention to homogeneous combinations of path and trace quantifiers like $[\alpha]\Box_{\mathcal{I}}\,\phi$ or $\langle\alpha\rangle\Diamond_{\mathcal{I}}\,\phi$.

Theorem 8 (Relative Completeness of QdMTL)**.** *The* QdMTL *calculus in Fig. 1 is complete relative to* FOQD*, i.e., every valid* QdMTL *formulas can be derived from* FOQD *tautologies.*

This result shows that metric temporal, non-metric temporal, and non-temporal properties of distributed hybrid systems can be proven to exactly the same extent to which properties of quantified differential equations can be proven. It also gives a formal justification that the QdMTL calculus reduces metric temporal properties to non-temporal Qd$\mathcal{L}$ properties.

6 Conclusions and Future Work

For reasoning about distributed hybrid systems, we have introduced a metric temporal dynamic logic, QdMTL, with modal path quantifiers over traces and metric temporal quantifiers along the traces. It combines the capabilities of quantified differential dynamic logic to reason about possible distributed hybrid system behavior with the power of metric temporal logic in reasoning about the behavior along time intervals of traces. We have presented a proof calculus for verifying metric temporal safety specifications of quantified hybrid programs in QdMTL, which, to the best of our knowledge, is the first verification approach that can handle metric temporal statements about distributed hybrid systems. Our sequent calculus for QdMTL is a completely modular combination of metric temporal, non-metric temporal, and non-temporal reasoning.

Furthermore, We have proven our calculus to be a sound and complete axiomatization relative to quantified differential equations.

We are currently extending a verification tool for distributed hybrid systems, which is an automated theorem prover called KeYmaeraD [26], to cover the full QdMTL calculus. One direction for future work is to extend QdMTL with parametric temporal operators [2] to formulate the *quantified differential parametric metric temporal dynamic logic* (QdPMTL) for specifying and verifying parametric temporal properties of distributed hybrid systems. We would also like to symbolically synthesis parametric safety timing constraints using the QdPMTL calculus. Another direction is to move toward the deductive verification of temporal properties of distributed stochastic hybrid systems.

References

[1] R. Alur, C. Courcoubetis, and D. L. Dill. Model-checking for real-time systems. In *LICS*, pages 414–425. IEEE Computer Society, 1990.

[2] R. Alur, K. Etessami, S. La Torre, and D. Peled. Parametric temporal logic for "model measuring". In J. Wiedermann, P. van Emde Boas, and M. Nielsen, editors, *ICALP*, volume 1644 of *LNCS*, pages 159–168. Springer, 1999.

[3] B. Beckert and S. Schlager. A sequent calculus for first-order dynamic logic with trace modalities. In R. Goré, A. Leitsch, and T. Nipkow, editors, *IJCAR*, volume 4130 of *LNCS*, pages 626–641. Springer, 2001.

[4] J. M. Davoren, V. Coulthard, N. Markey, and T. Moor. Non-deterministic temporal logics for general flow systems. In R. Alur and G. J. Pappas, editors, *HSCC*, volume 2993 of *LNCS*, pages 280–295. Springer, 2004.

[5] J. M. Davoren and A. Nerode. Logics for hybrid systems. *Proceedings of the IEEE*, 88(7):985–1010, July 2000.

[6] A. Deshpande, A. Göllü, and P. Varaiya. SHIFT: A formalism and a programming language for dynamic networks of hybrid automata. In *Hybrid Systems*, pages 113–133, 1996.

[7] G. Dowek, C. Muñoz, and V. A. Carreño. Provably safe coordinated strategy for distributed conflict resolution. In *AIAA Proceedings, AIAA-2005-6047*, pages 278–292, 2005.

[8] E. A. Emerson and E. M. Clarke. Using branching time temporal logic to synthesize synchronization skeletons. *Sci. Comput. Program*, 2(3):241–266, 1982.

[9] E. A. Emerson and J. Y. Halpern. "Sometimes" and "Not Never" revisited: on branching versus linear time temporal logic. *J. ACM*, 33(1):151–178, 1986.

[10] G. Fainekos and A. Girardand G. J. Pappas. Temporal logic verification using simulation. In E. Asarin and P. Bouyer, editors, *FORMATS*, volume 4202 of *LNCS*, pages 171–186. Springer, 2006.

[11] G. Fainekos and G. Pappas. Robustness of temporal logic specications for continuous-time signals. *Theoretical Computer Science*, 410(42):4262–4291, 2009.

[12] T. Flaminio and E. B. P. Tiezzi. On metric temporal lukasiewicz logic. *Electr. Notes Theor. Comput. Sci.*, 246:71–85, 2009.

[13] D. Harel, D. Kozen, and J. Tiuryn. *Dynamic logic.* MIT Press, Cambridge, 2000.

[14] T. A. Henzinger, X. Nicollin, J. Sifakis, and S. Yovine. Symbolic model checking for real-time systems. In *LICS*, pages 394–406. IEEE Computer Society, 1992.

[15] A. Hsu, F. Eskafi, S. Sachs, and P. Varaiya. Design of platoon maneuver protocols for IVHS. Technical Report PATH Research Report UCB-ITS-PRR-91-6, UC Berkeley, 1991.

[16] R. Koymans. Specifying real-time properties with metric temporal logic. *Real-Time Systems*, 2(4):255–299, 1990.

[17] F. Kratz, O. Sokolsky, G. J. Pappas, and I. Lee. R-Charon, a modeling language for reconfigurable hybrid systems. In *HSCC*, pages 392–406, 2006.

[18] O. Maler and D. Nickovic. Monitoring temporal properties of continuous signals. In Y. Lakhnech and S. Yovine, editors, *FORMATS-FTRTFT*, volume 3253 of *LNCS*, pages 152–166. Springer, 2004.

[19] F. Montagna, G. Michele Pinna, and E. B. P. Tiezzi. A cut-free proof system for bounded metric temporal logic over a dense time domain. *Math. Log. Q.*, 46(2):171–182, 2000.

[20] E. Plaku, L. E. Kavraki, and M. Y. Vardi. Falsication of LTL safety properties in hybrid systems. In S. Kowalewski and A. Philippou, editors, *TACAS*, volume 5505 of *LNCS*, pages 368–382. Springer, 2009.

[21] A. Platzer. Differential dynamic logic for hybrid systems. *J. Autom. Reas.*, 41(2):143–189, 2008.

[22] A. Platzer. Quantified differential dynamic logic for distributed hybrid systems. In A. Dawar and H. Veith, editors, *CSL*, volume 6247 of *LNCS*, pages 469–483. Springer, 2010.

[23] A. Platzer. Quantified differential dynamic logic for distributed hybrid systems. Technical Report CMU-CS-10-126, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, May 2010.

[24] A. Platzer. Quantified differential invariants. In E. Frazzoli and R. Grosu, editors, *HSCC*, pages 63–72. ACM, 2011.

[25] V. R. Pratt. Process logic. In *POPL*, pages 93–100, 1979.

[26] D. W. Renshaw, S. M. Loos, and A. Platzer. Distributed theorem proving for distributed hybrid systems. In S. Qin and Z. Qiu, editors, *ICFEM*, volume 6991 of *LNCS*, pages 356–371. Springer, 2011.

[27] S. Sankaranarayanan and G. Fainekos. Falsification of temporal properties of hybrid systems using the cross-entropy method. In T. Dang and I. M. Mitchell, editors, *HSCC*, pages 125–134. ACM, 2012.

Problem Libraries for Non-Classical Logics

– Extended Abstract –

Jens Otten and Thomas Raths

Institut für Informatik, University of Potsdam
August-Bebel-Str. 89, 14482 Potsdam-Babelsberg, Germany
{jeotten|traths}@cs.uni-potsdam.de

1 Introduction

Problem libraries for automated theorem proving (ATP) systems play a crucial role when developing, testing, benchmarking, and evaluating ATP systems and proof search calculi for classical and non-classical logics. During the development of ATP systems problem libraries are necessary for *testing*, providing evidence about the correctness of implemented proof search calculi. Furthermore, new proof search techniques and strategies can be evaluated, improving the efficiency of the implemented algorithms. Theoretical investigations into the time complexity of proof search calculi and their implementations are difficult and usually not carried out. Hence, standardized problem libraries are necessary to measure their performance and put the *evaluation* of ATP systems on a firm basis. The performance evaluation of ATP systems provides evidence about the efficiency of the different underlying proof search calculi, which is important in order to measure and to make progress in the field of ATP. Finally, problem libraries provide a platform for users of ATP systems to submit problems that occur in their applications. This, in turn, allows developers to use these problems to further improve the efficiency of their ATP systems.

For *classical* first-order logic, the first (printed) problem collections started 1976 with 63 problems by McCharen, Overbeek and Wos [6] and was ten years later extended by Pelletier's "Seventy-five Problems for Testing Automatic Theorem Provers" [9]. In 1997, version 2.0.0 of the TPTP library for classical logic included the first 217 problems in first-order (i.e. non-clausal) form [14]. Since then its number has risen significantly and the most recent version 6.0.0 of the TPTP library contains almost 8000 first-order problems (in non-clausal form) divided into 36 different problem domains. The existence of the TPTP library has stimulated the development of efficient ATP systems for classical logic leading to significant performance improvements.

For *non-classical* logics, such as intuitionistic and modal logic, only few problem libraries were developed so far. One of the first ATP systems for first-order *intuitionistic* logic was tested on a small collection of 43 first-order problems [12]. As intuitionistic logic has the same syntax as classical logic, problems used to test classical ATP systems can basically also be used for testing intuitionistic ATP systems. For *modal* logic the situation is more complicated, as it extends classical logic by the unary modal operators $\Box$ and $\Diamond$. The approach of generating random problem formulae for *propositional* modal logic is not practical for *first-order* modal logic. A small collection of problems for first-order modal logic was developed for testing one of the first ATP systems for first-order modal logic [15].

The main purpose of this paper is to make people aware of existing problem libraries for non-classical logic and to discuss ongoing and future work on these libraries. The paper provides an overview of existing problem libraries for some important non-classical logics, namely for first-order intuitionistic and first-order modal logics, and describes possible extensions and ideas for future developments of problem libraries for non-classical logics.

2 Problem Libraries for Intuitionistic and Modal Logics

The ILTP library and the QMLTP library provide a platform for testing and evaluating ATP systems for first-order intuitionistic and first-order modal logics, respectively.

2.1 The ILTP Library for Intuitionistic Logic

The current release v1.1.2 of ILTP library includes a total number of 2754 problems, which are grouped into 24 problem domains [11]. It is available online at `http://www.iltp.de`. Due to the ongoing interest in *propositional* intuitionistic logic, the problem set was split into a first-order part, which contains 2550 problems (2480 first-order problems and 70 propositional problems), and a propositional part, which contains 274 (propositional) problems.

Contents. 2324 problems were taken from the set of "FOF" problems in non-clausal form of the TPTP library for first-order classical logic [14]. These problems are classified into 21 domains, named after their corresponding name in the TPTP library. Among these problem domains are, e.g., AGT (agents), ALG (general algebra), GEO (geometry), KRS (knowledge representation), MGT (management), NLP (natural language processing), NUM (number theory), SET (set theory), SWC (software creation), SWV (software verification), and SYN (syntactic). Additionally, the three domain GEJ, GPJ, and SYJ with a total number of 430 problems were created. These domains contain problems from constructive geometry, group theory, and syntactic intuitionistic problems. In general, each problem file consists of a set of axioms $Ax_1,\dots,Ax_n$ and a conjecture *Conj*. Out of the 2754 problems, 1106 problems contain up to 9 axioms, 1485 problems contain between 10 and 99 axioms, and 163 problems contain between 100 and 922 axioms.

Syntax and Representation. The *syntax* for representing formulae and the naming scheme for the problem files were adopted from the TPTP library [14]. Each problem file is given an unambiguous name of the form `DDD.NNN+V[.SSS].p` consisting of its domain `DDD`, the number `NNN` of the problem, its version number `V`, and an optional parameter `SSS` indicating the size of the instance. Each problem file includes a header with useful information, e.g., the file name, a problem description, references, sources, and the intuitionistic status and rating. The ILTP library contains several *format files* that can be used to translate the problems in the ILTP library into the input syntax of existing ATP systems for intuitionistic logic.

Intuitionistic Status and Rating. The *intuitionistic status* of a problem formula is either `Theorem`, `Non-Theorem`, `Unsolved`, or `Open`. If at least one intuitionistic ATP system has proved or refuted the problem formula $(Ax_1 \wedge \dots \wedge Ax_n) \Rightarrow Conj$, the status is `Theorem` or `Non-Theorem`, respectively. Problems with an `Unsolved` status have not yet been solved by any intuitionistic ATP system, but it is known if they are valid or invalid; for problems with an `Open` status it is not known if they are valid or invalid. The *intuitionistic rating* of a problem indicates the difficulty of that problem with respect to current (intuitionistic) state-of-the-art ATP systems.

2.2 The QMLTP Library for Modal Logics

The current release v1.1 of the QMLTP library includes a total number of 600 problems, which are grouped into 11 problem domains [10]. It is available online at `http://www.iltp.de/qmltp/`. Out of these 600 problems, 580 are unimodal problems and 20 are multimodal problems; 421 problems are first-order problems, 179 are propositional problems.

Contents. The 245 problems in the domains GAL, GLC, GNL, GSE, GSV, and GSY were generated by using Gödel's embedding of intuitionistic logic into the modal logic S4 [3] The original problems were taken from the domains ALG (general algebra), LCL (logic calculi), NLP (natural language processing), SET (set theory), SWV (software verification), and SYN (syntactic) of the TPTP library [14]. The domains APM (applications mixed) and SYM (syntactic modal) contain 255 problems from different applications (planning, querying databases, natural language processing and software verification), syntactic problems from various textbooks, and problems from the TANCS-2000 system competition for modal (propositional) ATP systems [5]. The domains NLP (natural language processing) and SET (set theory) contain a set of 80 classical first-order problems, i.e. problems containing no modal operators, from the corresponding domains of the TPTP library [14]. Furthermore, the domain MML (multimodal logic) contains 20 problems in a multimodal logic syntax from various textbooks and applications. Again, each problem file consists of a set of axioms $Ax_1,\dots,Ax_n$ and a conjecture *Conj*. Out of the 600 problems, 342 problems contain up to 9 axioms, 210 problems contain between 10 and 99 axioms, and 48 problems contain between 100 and 150 axioms.

Syntax and Representation The naming scheme of the problem files is adopted from the TPTP library. For the *syntax* of the modal problems, the syntax of the TPTP library [14] was extended by the two operators `#box:` and `#dia:` representing the modal operators $\Box$ and $\Diamond$, respectively. For multimodal logic, the modal operators $\Box_i$ and $\Diamond_i$ are represented by `#box(i):` and `#dia(i):`, respectively. In this case the command `set_logic` of the TPTP *process instruction language* is used to specify the semantics of the modal operators. E.g., `tpi(1,set_logic,modal([cumulative], [(1,d),(2,s4)]))` specifies that the multimodal operators $\Box_1/\Diamond_1$ and $\Box_2/\Diamond_2$ are interpreted with respect to the cumulative domain variant of the modal logics D and S4, respectively. Each problem file includes a header with useful information, e.g., the file name, a problem description, references, sources, and the modal status and rating. *Format files* are included in the QMLTP library that can be used to convert the problems in the QMLTP library into the input syntax of existing ATP systems for (first-order) modal logic.

Modal Status and Rating. The *modal status* of a problem is either `Theorem`, `Non-Theorem`, or `Unsolved` and is specified with respect to a particular modal logic and *domain condition*. Release v1.1 of the QMLTP library provides modal status and rating information for the modal logics K, D, T, S4, and S5 with constant, cumulative, and varying domain conditions. Term designation is assumed to be rigid, i.e., terms denote the same object in each world, and terms are local, i.e., any ground term denotes an existing object in every world. Even though the modal status and rating information is only provided for the standard modal logics K, D, T, S4, and S5, the usage of the problem library is not restricted to these logics. Furthermore, the *local* logical consequence is used (see also Section 3.1): the status of a problem is `Theorem` if the (modal) formula $(Ax_1 \wedge \dots \wedge Ax_n) \Rightarrow Conj$ was proven valid by at least one (modal) ATP system; if the formula was refuted, i.e. proven invalid, its status is `Non-Theorem`.

3 Ongoing Work and Future Plans

Current and future work include extending the existing problem libraries and developing a common benchmark platform for non-classical logics.

3.1 Extending the ILTP Library and the QMLTP Library

The ongoing work on the ILTP library and the QMLTP library includes increasing the number of problems and providing status and rating information for additional modal logics/semantics.

Adding More Difficult Problems. In order to make meaningful performance evaluations possible, problem libraries need to contain a sufficient amount of challenging, i.e., difficult problems.

Out of the 2550 problems of the first-order part of ILTP library v1.1.2, approximately 1000 problems (40%) are solved, i.e., proved or refuted by current state-of-the-art ATP systems for intuitionistic logic. Hence, the problems in the ILTP library are still sufficiently difficult in order to support the development of more efficient ATP systems. Out of the 274 problems of the propositional part, around 95% are solved by current propositional state-of-the-art ATP systems [4]. Even though the focus of the ILTP library is on first-order problems, more propositional problems will be added in the future, e.g., by generating non-clausal propositional instances of first-order formulae using the first2p tool [1].

Out of the 580 unimodal problems of the QMLTP library v1.1, between 60% (modal K) and 85% (modal S5) are solved by current state-of-the-art ATP systems for modal logic. Additional (syntactic) problems are obtained by applying Gödel's embedding for intuitionistic logic to more difficult problems.

Semantics of Modal Operators and Logical Consequence. The information about the modal status and rating of the problems in the QMLTP library will be extended to cover additional logics, i.e., the modal logics K4, D4, and B. Currently, the only ATP systems that support these logics use an embedding into higher-order logic [1]. Even if the *information* about modal status and rating for these logics is currently not included in the QMLTP library, users can already test and benchmark their ATP systems for these logics by using the existing problem sets in the library.

Currently, all modal status and rating information are based on the *local* logical consequence of modal logic. Let $M \models_w F$ iff M is a (Kripke-)model for the formula F in the world w and let $M \models_w \Gamma$ iff $M \models_w F$ for all $F \in \Gamma$. The *local logical consequence* $\models^L$ is defined as $\Gamma \models^L F$ iff for every model M and every world w: if $M \models_w \Gamma$ then $M \models_w F$ holds. The formulae (i.e. Axioms) in the set Γ are called *local hypotheses* and this kind of notion is often used in the literature. In this case the *deduction theorem* holds, i.e. $\Gamma \cup F' \models^L F$ iff $\Gamma \models^L (F' \Rightarrow F)$ (see also the remarks in the last paragraph of Section 2.2). The *global logical consequence* $\models^G$ is defined as $\Delta \models^G F$ iff for every model M: if $M \models_w \Delta$ for every world w then $M \models_w F$ for every world w holds; the formulae in the set Δ are called *global hypotheses*. Modal status and rating information using the *global* logical consequence will be included in future versions. Until then, users can already test their ATP systems using the global logical consequence (see, e.g.,[2]), but are expected to indicate whenever they diverge from the standard local logical consequence.

Problems from Applications. Future versions of the problem libraries will include more problems from applications. All users who have an application that uses (first-order) intuitionistic or modal logic are asked to get in contact with the authors and submit their problems to the ILTP or QMLTP library.

3.2 A Common Benchmark Platform for Non-Classical Logics

In order to better support the development of ATP systems for non-classical logics, the development of a common benchmark platform for non-classical logics, similar to the StarExec platform, is currently discussed. StarExec [13] provides an infrastructure to researchers to manage benchmark libraries and solvers/ATP systems for some popular logics, such as first-order classical logic and propositional classical logic (SAT). A common benchmark platform for non-classical logics would not only cover intuitionistic and modal logics, but also less popular non-classical logics, such as interval or alternating-time temporal logics, for which the development of separate, independent platforms is not practical (and which are still not widespread enough to include them in the StarExec platform). It would provide a forum for collecting benchmark problems that are currently independently developed, collected and used by developers of non-classical ATP systems, and would make it easier for researchers developing applications to submit their non-classical problems.

4 Conclusion

Non-classical logics, such as intuitionistic and modal logics, have many applications, e.g., in Computer Science, Artificial Intelligence and Philosophy. In contrast to classical logic, the development of ATP systems for non-classical is still in its infancy. This is in particular true for most *first-order* non-classical logics. Problem libraries play an important role when developing, testing, and evaluating ATP systems. They stimulate the development of novel, more efficient calculi and ATP systems. The ILTP library and the QMLTP library are comprehensive and established problem libraries for first-order intuitionistic and first-order modal logics. They already supported the development of some of the fastest ATP systems for first-order intuitionistic and several first-order modal logics [1, 7, 8]. Future versions of these libraries will include more problems as well as status and rating information for further (modal) logics. A common benchmark platform for non-classical logics will support additional non-classical logics.

Acknowledgements. The authors would like to thank Christoph Benzmüller and Geoff Sutcliffe for many helpful discussions and suggestions.

References

[1] C. Benzmüller, J. Otten, T. Raths. Implementing and Evaluating Provers for First-order Modal Logics. In L. De Raedt et al., Eds., *20th European Conference on Artificial Intelligence, ECAI 2012*, pp. 163–168. IOS Press, Amsterdam, 2012.

[2] C. Benzmüller, T. Raths. HOL Based First-Order Modal Logic Provers. In K. McMillian et al., Eds., *LPAR 2013*, LNCS 8312, pp. 127–136. Springer, Heidelberg, 2013.

[3] K. Gödel. An Interpretation of the Intuitionistic Sentential Logic. In J. Hintikka, Ed., *The Philosophy of Mathematics*, pp. 128–129. Oxford University Press, Oxford, 1969.

[4] R. Goré, J. Thomson. An Improved BDD Method for Intuitionistic Propositional Logic: BDDIntKt System Description. In M. P. Bonacina, Ed., *CADE 2013*, LNCS 7898, pp. 275–281. Springer, Heidelberg, 2013.

[5] F. Massacci, F.M. Donini: Design and Results of TANCS-2000 Non-classical (Modal) Systems Comparison. In R. Dyckhoff., Ed., *TABLEAUX 2000*, LNAI 1847, pp. 50–56. Springer, Heidelberg, 2000.

[6] J.D. McCharen, R.A. Overbeek, L.A. Wos. Problems and Experiments for and with Automated Theorem-Proving Programs. *IEEE Transactions on Computers*, C-25(8):773–782, 1976.

[7] J. Otten. leanCoP 2.0 and ileanCoP 1.2: High Performance Lean Theorem Proving in Classical and Intuitionistic Logic. In A. Armando et al., Eds., *IJCAR 2008*, LNCS 5195, pp. 283–291. Springer, Heidelberg, 2008.

[8] J. Otten. MleanCoP: A Connection Prover for First-Order Modal Logic. In S. Demri, D. Kapur, C. Weidenbach, Eds., *IJCAR 2014*, LNAI 8562, pp. 269–276. Springer, Heidelberg, 2014.

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

[10] T. Raths, J. Otten. The QMLTP Problem Library for First-order Modal Logics. In B. Gramlich, D. Miller, U. Sattler, Eds., *IJCAR 2012*, LNAI 7364, pp. 454–461. Springer, Heidelberg, 2012.

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

[12] D. Sahlin, T. Franzen, and S. Haridi. An intuitionistic predicate logic theorem prover. *Journal of Logic and Computation*, 2:619–656, 1992.

[13] A. Stump, G. Sutcliffe, C. Tinelli. Introducing StarExec: a Cross-Community Infrastructure for Logic Solving. In V. Klebanov et al., Eds., *COMPARE 2012*, Workshop Proceedings, p. 2, Manchester/UK, 2012.

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

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

HOL Provers for First-order Modal Logics — Experiments*

Christoph Benzmüller

Department of Mathematics and Computer Science, Freie Universität Berlin, Germany

Abstract

Higher-order automated theorem provers have been employed to automate first-order modal logics. Extending previous work, an experiment has been carried out to evaluate their collaborative and individual performances.

1 Introduction

Higher-order automated theorem provers are well suited as reasoners for a wide range of quantified non-classical logics [1]. The key idea is to exploit classical higher-order logic (HOL) as a universal meta-logic and, for example, to explicitly encode Kripke structures within this meta-logic. Experiments have shown that this approach, which is orthogonal to explicit world labeling techniques in direct theorem provers, is indeed competitive [2]. More recent, but non-exhaustive, experiments have improved and confirmed these results [3]. This short paper significantly further extends the experiments reported in [3]. The paper thus provides useful and relevant information for evaluations of competitor systems. Moreover, some light is shed on the collective and individual performances of the higher-order automated theorem provers LEO-II [4], Satallax [6], Isabelle [9], agsyHOL [8] and Nitpick [5].

2 Experiments

A meta-prover for HOL, called HOL-P has been introduced and evaluated in [3]. This meta-prover exploits the SystemOnTPTP infrastructure [11] and sequentially schedules the HOL reasoners LEO-II, Satallax, Isabelle, agsyHOL and Nitpick running remotely at the SystemOnTPTP cluster at Miami (which provides 2.80GHz computers with 1GB memory). The HOL-P system and the HOL-P constituent provers are evaluated here with respect to the 580 benchmark problems in the QMLTP library [10]. Extending previous experiments [3], these problems are studied for 5 different logics (K, D, T, S4, S5) and for 3 different domain conditions (constant, cumulative, varying). The total sum of considered problem variants is thus 8700. These QMLTP problem variants have been converted into THF syntax with the FMLtoHOL tool [3]. Moreover, the particular configuration of HOL-P has been varied, and different system timeouts and different numbers of constituent provers have been considered. These experiments, which have been conducted over the past four months, have required a substantial amount of time and computing resources on the SystemOnTPTP cluster (I am grateful to Geoff Sutcliffe for providing these resources).[1]

*This work has been supported by the German Research Foundation (DFG) under grants BE 2501/9-1 and BE 2501/11-1.

[1]Important technical remark: QMLTP axioms have been treated as global axioms in the experiments; cf. the definition of local versus global logical consequence in [7].

	K			D			T			S4			S5		
Type	co	cu	va	co	cu	va	co	cu	va	co	cu	va	co	cu	va
THM	192	168	149	206	180	159	260	234	211	298	271	242	345	333	282
CSA	259	284	309	253	270	299	177	190	229	132	146	186	77	77	129
SAT	3	3	3	2	2	2	2	2	2	2	2	2	2	2	2
Σ	454	455	461	461	452	460	439	426	442	432	419	430	424	412	413

Table 1: Performance of HOL-P (with 600s overall timeout, 120s timeout for each constituent prover) for first-order modal logics K, D, T, S4 and S5 with constant domains (co), cumulative domains (cu) and varying domains (va).

	THM	CSA	SAT	Σ	UNK
HOL-P	3530	3017	33	6580	2120
Satallax	3167	752	0	3919	4781
Nitrox	0	3017	33	3050	5650
Isabelle	2955	0	0	2955	5745
LEO-II	2647	284	0	2931	5769
agsyHOL	2784	0	0	2784	5916

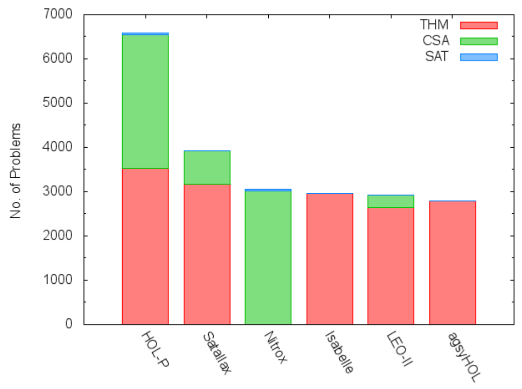

Table 2: Performance of the HOL-P constituent provers (120s timeout each; HOL-P results are wrt 600 seconds timeout).

2.1 First experiment

In this experiment HOL-P was configured to sequentially schedule the provers LEO-II—1.6.2, Satallax—2.7, Isabelle—2013, Nitrox—2013, and agsyHOL—1.0. The timeout for each prover was set to 120 seconds of CPU time. For HOL-P this adds to a total timeout of 600 seconds. The results of this experiment are reported in Table 1 (in this paper THM, CSA, SAT, and UNK stand for theorem, countersatisfiable, satisfiable, and unknown, respectively). The particular setting of the experiment thus coincides with the setting chosen in [3]. However, here HOL-P is evaluated for logics K, T and S5 in addition to logics D and S4. The particular results for the latter two logics very slightly differs from those reported in [3]. We conjecture that these differences are related to SystemOnTPTP issues, which serves as black box in our experiments. In particular, there are no means to control the very detailled execution conditions for each prover run when using this infrastructure. Future work should therefore investigate how the replication precision of experiments conducted via the SystemOnTPTP infrastructure can be further improved.

The individual performances of the HOL-P constituent provers have also been evaluated. Table 2 depicts the cumulative performance of each prover for all 8700 QMLTP problem variants. Remember that each prover was given a timeout of 120 seconds. The cumulative performance of HOL-P is also depicted; however, the comparison is unfair since the underlying timeout of HOL-P is 600 seconds. An alternative comparison is possible with the results reported for HOL-P in Table 3. There HOL-P was run with the same constituent provers but with an overall timeout of just 100 seconds; in this setting HOL-P nevertheless performed better wrt the number of

	K			D			T			S4			S5		
Type	co	cu	va	co	cu	va	co	cu	va	co	cu	va	co	cu	va
THM	186	162	141	201	175	154	252	223	205	289	261	233	345	319	270
CSA	263	275	298	233	245	268	159	180	211	128	140	179	77	74	126
SAT	3	3	3	2	2	2	2	2	2	2	2	2	2	2	2
Σ	452	440	442	436	422	424	413	405	418	419	403	414	424	395	398

Table 3: Performance of HOL-P (100s overall timeout, 20s timeout for each constituent prover) for first-order modal logics K, D, T, S4 and S5 with constant domains (co), cumulative domains (cu) and varying domains (va).

	THM	CSA	SAT	Σ	UNK
HOL-P	3408	2856	33	6297	2403
Satallax	3024	749	0	3773	4927
Nitrox	0	2856	33	2889	5811
LEO-II	2472	231	0	2703	5997
agsyHOL	2644	0	0	2644	6056
Isabelle	2354	0	0	2354	6346

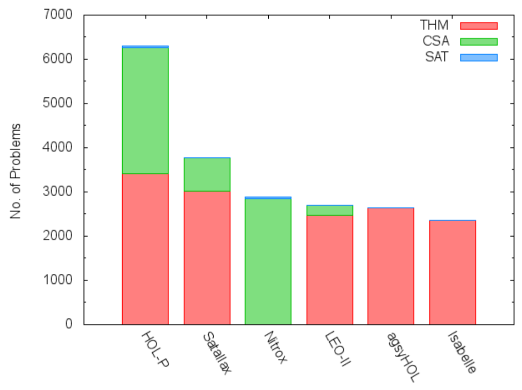

Table 4: Performance of the HOL-P constituent provers (20s timeout each; HOL-P results are wrt 100 seconds timeout).

proved theorems and the overall number of solutions found than any of the individual provers with a 120 seconds timeout. Only with respect to finding countermodels the situation differs, here Nitrox has a slight advantage over HOL-P.

2.2 Second experiment

In this experiment HOL-P was again configured to sequentially schedule the provers LEO-II—1.6.2, Satallax—2.7, Isabelle—2013, Nitrox—2013, and agsyHOL—1.0. However, the timeout for each prover was now set to 20 seconds of CPU time. For HOL-P this adds to a total timeout of 100 seconds. The results of this experiment are reported in Tables 3 and 4. Interestingly, the performance loss with regard to the first experiment is less significant as expected. Even with the short 20 second timeouts for the individual provers, HOL-P remains a competitive prover for first-order modal logics. The individual performances of the HOL-P constituent provers have slightly changed though. In particular Isabelle performs weaker with short timeouts.

2.3 Third experiment

In this experiment HOL-P was configured to sequentially schedule only Satallax—2.7 and Nitrox—2013. These two individual provers performed best in the above experiments. Moreover, they are quite complementary regarding their specialization in proving theorems and finding countermodels. The timeout for each prover was set to 50 seconds of CPU time. For HOL-P this adds to a total timeout of 100 seconds. The results of this experiment are reported in Table 5. The theorem proving performance of HOL-P in this experiment is weaker than in the second experiment. This illustrates the complementary strength of the HOL proveres for proving theorems. However, the performance for finding countermodels has slightly improved now for HOL-P, since Nitrox, which is the only strong countermodel finder currently available for HOL, productively employs its increased reasoning time in the modified setting.

	K			D			T			S4			S5		
Type	co	cu	va	co	cu	va	co	cu	va	co	cu	va	co	cu	va
THM	162	150	132	175	161	141	225	212	190	262	246	219	305	305	258
CSA	266	280	308	251	267	298	176	190	223	132	146	186	77	77	129
SAT	3	3	3	2	2	2	2	2	2	2	2	2	2	2	2
Σ	431	433	443	428	430	441	403	404	415	396	394	407	384	384	389

Table 5: Performance of HOL-P (100s overall timeout, 50s timeout for each constituent prover) for first-order modal logics K, D, T, S4 and S5 with constant domains (co), cumulative domains (cu) and varying domains (va).

3 Summary

The collaborative and individual performances of higher-order automated theorem provers has been evaluated for first-order modal logic problems. The results demonstrate the dominance of the collaborative theorem prover HOL-P over its individual constituent provers. The strongest individual provers in our experiments were Satallax and Nitrox. As our experiments show, Satallax is not subsuming the other provers: assigning more reasoning time to Satallax is less effective than operating with different constituent provers in HOL-P and sharing the available resources. An optimal configuration of HOL-P has not yet been identified and further experiments could try find such a configuration. However, given that the individual HOL provers are subject to frequent revisions and improvements, this optimization problem appears to be a non-trivial, moving target.

References

[1] C. Benzmüller. A top-down approach to combining logics. In *Proc. of the 5th International Conference on Agents and Artificial Intelligence (ICAART)*. SciTePress Digital Library, 2013.

[2] C. Benzmüller, J. Otten, and T. Raths. Implementing and evaluating provers for first-order modal logics. In *Proc. of ECAI 2012*, Montpellier, France, 2012.

[3] C. Benzmüller and Th. Raths. Hol based first-order modal logic provers. In *Proc. of LPAR*, volume 8312 of *LNCS*, pages 127–136. Springer, 2013.

[4] C. Benzmüller, F. Theiss, L. Paulson, and A. Fietzke. LEO-II - a cooperative automatic theorem prover for higher-order logic. In *Proc. of IJCAR 2008*, volume 5195 of *LNCS*, pages 162–170. Springer, 2008.

[5] J.C. Blanchette and T. Nipkow. Nitpick: A counterexample generator for higher-order logic based on a relational model finder. In *Proc. of ITP 2010*, volume 6172 of *LNCS*, pages 131–146. Springer, 2010.

[6] C.E. Brown. Satallax: An automated higher-order prover. In *Proc. of IJCAR 2012*, volume 7364 of *LNCS*, pages 111 – 117. Springer, 2012.

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

[8] F. Lindblad. agsyHol:. `https://github.com/frelindb/agsyHOL`, 2012.

[9] T. Nipkow, L.C. Paulson, and M. Wenzel. *Isabelle/HOL: A Proof Assistant for Higher-Order Logic*. Number 2283 in LNCS. Springer, 2002.

[10] T. Raths and J. Otten. The QMLTP problem library for first-order modal logics. In *Proc. of IJCAR 2012*, volume 7364 of *LNCS*, pages 454–461. Springer, 2012.

[11] G. Sutcliffe. The TPTP problem library and associated infrastructure. *Journal of Automated Reasoning*, 43(4):337–362, 2009.

Proof Support for Common Logic

Till Mossakowski[1], Mihai Codescu[1], Oliver Kutz[1],
Christoph Lange[2], and Michael Gruninger[3]

[1] Institute of Knowledge and Language Engineering, Otto-von-Guericke University of Magdeburg, Germany
[2] University of Bonn and Fraunhofer IAIS, Germany
[3] University of Toronto, Canada

Abstract

We present the theoretical background for an extension of the Heterogeneous Tool Set HETS that enables proof support for Common Logic. This is achieved via logic translations that relate Common Logic and some of its sublogics to already supported logics and automated theorem proving systems. We thus provide the first theorem proving support for Common Logic covering the full language, including the possibility of verifying meta-theoretical relationships between Common Logic theories.

1 Introduction

Common Logic (CL) is an ISO-standardised[1] language based on untyped first-order logic, but extending it in several ways that ease the formulation of complex first-order theories. [21] discusses in detail the motivations and philosophical background of CL, arguing that it not only is a natural formalism for knowledge representation in the context of the Web, but that it also constitutes a natural evolution from the canonical textbook notation and semantics of first-order logic (FOL for short), dispensing with some deeply entrenched views that are reflected in FOL's syntax (and that partly go back to, e.g., Frege's metaphysical views), in particular the segregation of objects, functions, and predicates, fixed arities and signatures, and strict syntactic typing.

Although a substantial number of CL theories have been developed (see Section 2), surprisingly little tool support has been realised so far. In this work, we fill this gap using the Heterogeneous Tool Set HETS [26, 27, 22]). HETS is a general purpose analysis and proof management tool for logical theories. We show that the HETS architecture eases the implementation of a comprehensive tool set for CL, including parsing, theorem proving, checking for consistency, and more.[2]

This paper is organised as follows. In Section 2, we recall CL, and discuss its use cases and hitherto existing tool support. In Section 3, we recall HETS and its theoretical grounding in institution theory. Section 4 contains the core contribution of the paper, namely the integration of CL into HETS, enabling, for the first time, full proof support for CL and its sublogics via various logic translations to already supported logics and automated reasoners. Section 5 presents an example illustrating the achieved CL reasoning support in the context of the COLORE repository. We close in Section 6 with a discussion of the achieved proof support and an outlook to future work.

2 Common Logic

CL is based on untyped first-order logic, but extends first-order logic in two ways: (1) any term can be used as function or predicate, and (2) sequence markers allow for talking about sequences of individuals

[1]Published as "ISO/IEC 24707:2007—Information technology—Common Logic (CL): a framework for a family of logic-based languages" [12]

[2]For a detailed description of HETS, see the HETS user guide [25] (and the special CL version of the guide). HETS is currently available for Linux and Mac OS X from the HETS home page `http://hets.eu`, where you also find the user guides and packages ready for Ubuntu Linux.

directly, and in particular, provide a succinct way for axiomatising polyadic functions and predicates.[3]

A CL signature Σ (called vocabulary in CL terminology) consists of a set of names, with a subset called the set of discourse names, and a set of sequence markers. Discourse names denote first-class objects, which may be individuals, predicates and functions (by Common Logic's "wild west" syntax, any individual can be used in functional and predicate position). Non-discourse names denote second-class objects which can be predicates or functions, but not individuals. Sequence markers denote sequences of (first-class) objects. A Σ-model consists of a set *UR*, the universe of reference (consisting of all first-class and second-class objects), with a non-empty subset $UD \subseteq UR$, the universe of discourse (the first-class objects),[4] and four mappings:

- *int* from names in Σ to *UR*, such that $int(n)$ is in *UD* if and only if n is a discourse name;
- *rel* from *UR* to subsets of $UD^* = \{\langle x_1, \dots, x_n \rangle \mid x_1, \dots, x_n \in UD, n \in \mathbb{N}\}$ (i.e., the set of finite sequences of elements of *UD*);
- *fun* from *UR* to total functions from UD^* into *UD*;
- *seq* from sequence markers in Σ to UD^*.

Σ-sentences are first-order-like sentences formed with the help of Boolean connectives and quantification over names and sequence markers from atoms, which are either equations betwen two terms or predications which consist of a term, called the predicate, and a term sequence, called the argument sequence. A term is either a name or a functional term, where the latter consists of a term, called the operator, and a term sequence. A term sequence is a finite list of terms or sequence markers. It is important that CL distinguishes between a name n and a functional term (n) where n is a 0-ary operator, i.e. it takes no arguments. We will write functional terms and predications in a higher-order like syntax as $(t\ s)$. As an example, consider the formula `(instance instance BinaryPredicate)`, stating that the predicate `instance` is an instance of a binary predicate.

Interpretation of terms and formulae is as in first-order logic, with the difference that a predication $(t\ s)$ is interpreted in a model M as $(rel^M(t^M))(s^M)$ and a function application $(t\ s)$, as $(fun^M(t^M))(s^M)$, while a name n occuring in a formula is interpreted as $int^M(n)$ and a marker m as $seq^M(m)$. A further difference to first-order logic is the presence of sequence terms (namely sequence markers and juxtapositions of terms), which denote sequences in UD^*, with term juxtaposition interpreted by sequence concatenation (or details, see [12]). For example, it is possible to express that the items of a list are distinct as follows (using the sequence marker "..."):

```
(distinct)        // the empty sequence is distinct
(distinct x)    // singleton sequences are distinct
(iff (distinct x y ...) // recursion for length > 1
     (and (not (= x y))       // the first two elements must be different
          (distinct x ...)    // and each of them distinct
          (distinct y ...) )) // to the rest
```

For the rationale and methodology of CL and the possibility to define dialects covering different first-order languages, see [12]. CL also includes modules as a syntactic category, with a semantics that restricts locally the universe of discourse.[5]

[3]Strictly speaking, only the sequence markers go beyond first-order logic, as the other higher-order features of CL can be simulated in FOL.

[4]The CLIF dialect of CL also requires that the natural numbers and the strings over unicode characters are part of the universe of discourse. Even if we use CLIF input syntax, we ignore this requirement here, as it is not a general requirement for CL, and the role of datatypes is currently under discussion for the next revision of the CL standard.

[5]See [29] for technical details of the revised semantics for CL modules, which is also considered in the current revision process of ISO/IEC 24707.

2.1 Uses of Common Logic

COLORE (Common Logic Ontology Repository, available at `http://colore.oor.net`) is an open repository of over 600 CL ontologies. One of the primary applications of COLORE is to support the verification of ontologies for common-sense domains such as time, space, shape, and processes. Verification consists in proving that the ontology is equivalent to a set of core ontologies for mathematical domains such as orderings, incidence structures, graphs, and algebraic structures. COLORE comprises core ontologies that formalise algebraic structures (such as groups, fields, and vector spaces), orderings (such as partial orderings, lattices, betweenness), graphs, and incidence structures in CL, and, based on these, representation theorems for generic ontologies for the above-mentioned common-sense domains.

SUMO (Suggested Upper Merged Ontology) is a large upper ontology, covering a wide range of concepts. It is one candidate for the "standard upper ontology" of IEEE working group 1600.1. While SUMO has originally been formulated in the Knowledge Interchange Format (KIF, a precursor of CL), SUMO-CL is a CL partial variant of SUMO produced by Kojeware (see also [4] for a discussion of higher-order aspects in SUMO). The SUMO-KIF version with all the mid-level and domain ontologies that are shipped with SUMO consists of roughly 32,000 concepts in 150,000 lines of specification.

fUML (Foundational UML) is a subset of the Unified Modelling Language (UML 2) defined by the Object Management Group (OMG). The OMG has specified a "foundational execution semantics" for fUML using CL (see `http://www.omg.org/spec/FUML/`).

PSL (Process Specification Language, ISO standard 18629), developed at the National Institute of Standards and Technology (NIST), is an ontology of processes, their components and their relationships. It is also part of COLORE.

2.2 Tool support for Common Logic

Current software tool support for CL is still ad hoc, and is primarily restricted to parsers and translators between CLIF and the TPTP exchange syntax for first-order logic. The work in [17] proposed an environment for ontology development in CL, named Macleod; although this work includes detailed design documents for the environment, there is as yet no available implementation. Similarly, [34] proposed a system architecture for COLORE that supports services for manipulating CL ontologies, once more merely with basic functionality such as parsing being implemented. There exist some ad-hoc syntax translations of CL to first-order provers, but these seem not to be backed up with a semantic analysis of their correctness and they only support the classical first-order fragment of CL. This means that one cannot reason about a CL theory that uses sequence markers or complex terms t, possibly involving quantified variables, as operators in functional terms or predicates in predications.

3 Institutions and the Heterogeneous Tool Set Hets

Hets [26, 27, 22] is an open source software providing a general framework for formal methods integration and proof management. One can think of Hets as acting like a motherboard where different expansion cards can be plugged in, the expansion cards here being individual logics (with their analysis and proof tools) as well as logic translations. The Hets motherboard already has plugged in a number of expansion cards (e.g., theorem provers like SPASS, Vampire, LEO-II, Isabelle and more, as well as model finders). Hence, a variety of tools is available, without the need to hard-wire each tool to the logic at hand.

Hets consists of logic-specific tools for the parsing and static analysis of basic logical theories written in the different involved logics, as well as a logic-independent parsing and static analysis tool for structured theories and theory relations. The latter of course needs to call the logic-specific tools whenever a basic logical theory is encountered.

3.1 Institutions

The semantic background of Hets is the theory of institutions [14], formalising the notion of a logic. An institution provides notions of signature and signature morphism (formally, this is given by a category **Sig**), and for each signature Σ in **Sig**, a set of sentences $Sen(\Sigma)$, a class of models $Mod(\Sigma)$ and a binary satisfaction relation $\models_\Sigma$ between models and sentences. Furthermore, given a signature morphism $\sigma\colon \Sigma_1 \to \Sigma_2$, an institution provides sentence translation along σ, written $\sigma(\varphi)$, and model reduct against σ, written $M|_\sigma$, in a way that satisfaction remains invariant:

$$M'|_\sigma \models_{\Sigma_1} \varphi \text{ iff } M' \models_{\Sigma_2} \sigma(\varphi)$$

for each $\varphi \in Sen(\Sigma_1)$ and $M' \in Mod(\Sigma_2)$.

3.2 Logics supported by Hets

Based on this foundation, Hets supports a variety of different logics. The following ones are most important for use with CL:

OWL 2 is the Web Ontology Language recommended by the World Wide Web Consortium (W3C, `http://www.w3.org`); see [31]. It is used for knowledge representation on the Semantic Web [6]. Hets supports OWL 2 DL and the provers Fact++ and Pellet.

FOL/TPTP is untyped first-order logic with equality,[6] underlying the interchange language TPTP [36], see `http://www.tptp.org`. Hets offers several automated theorem proving (ATP) systems for TPTP, namely SPASS [37], Vampire [33], Eprover [35], Darwin [2], E-KRHyper [32], and MathServe Broker (which chooses an appropriate ATP upon a classification of the FOL problem) [38].

CFOL is many-sorted first-order logic with so-called sort generation constraints, expressing that each value of a given sort is the interpretation of some term involving certain functions (called constructors). This is equivalent to an induction principle and allows the axiomatisation of lists and other datatypes, using the usual Peano-style axioms (such an axiomatisation is called a *free type*). CFOL is a sublogic of the Common Algebraic Specification Language Casl, see [28, 7]. Proof support for CFOL is available through a simple induction scheme in connection with automated first-order provers like SPASS [20], or via a comorphism to HOL. A connection to the induction prover KIV [1] is under development.

HOL is typed higher-order logic [9]. Hets actually supports several variants of HOL, among them THF0 (the higher-order version of TPTP [5]), with automated provers LEO-II [3], Satallax [10] and an automated interface to Isabelle [30], as well as the logic of Isabelle, with an interactive interface.

Hets supports the input languages of these logics directly. Adding a new logic can be done by writing a number of Haskell data types and functions, providing abstract syntax, parser, static analysis and prover interfaces for the logic. It is also possible to integrate logics (as described in [11]) by specifying them in a logical framework like LF [15].

For expressing meta relations between logical theories, Hets supports the Distributed Ontology Language (DOL). DOL can express relations of theories such as logical consequences, relative interpretations of theories and conservative extensions. DOL is also capable of expressing such relations across theories written in different logics, as well as translations of theories along logic translations [24]. DOL is going to be submitted as response to the Object Management Group's (OMG) Ontology, Model and Specification Integration and Interoperability (OntoIOp) Request For Proposal, see `http://www.ontoiop.org`.

[6] FOL/TPTP is called SoftFOL in the HETS implementation. SoftFOL extends first-order logic with equality with a softly typed logic used by SPASS; however, in this paper we will only use the sublanguage corresponding to FOL.

3.3 Institution Comorphisms

HETS' logic translations are formalised as so-called institution comorphisms [13]. A comorphism from institution I to institution J consists of

- a translation Φ of I-signatures to J-signatures (technically, a functor),
- for each signature Σ, a translation α_Σ of Σ-sentences in I to $\Phi(\Sigma)$-sentences in J, and
- for each signature Σ, a translation β_Σ of $\Phi(\Sigma)$-models in J to Σ-models in I,[7]

such that the following satisfaction condition holds:

$$\beta_\Sigma(M)^I \models \varphi \text{ iff } M \models^J \alpha_\Sigma(\varphi)$$

for each signature Σ, Σ-sentence φ and $\Phi(\Sigma)$-model M. A comorphism is:

- *faithful* if logical consequence is preserved and reflected along the comorphism:

$$\Gamma \models^I \varphi \text{ iff } \alpha(\Gamma) \models^J \alpha(\varphi)$$

- *model-expansive* if each β_Σ is surjective;
- a *subinstitution comorphism* if Φ is an embedding, each α_Σ is injective and each β_Σ is bijective[8];
- an *inclusion comorphism* if Φ and each α_Σ are inclusions, and each β_Σ is the identity.

It is known that each subinstitution comorphism is model-expansive and each model-expansive comorphism is also faithful. Faithfulness means that a proof goal $\Gamma \models^I \varphi$ in I can be solved by a theorem prover for J by just feeding the theorem prover with $\alpha(\Gamma) \models^J \alpha(\varphi)$. Subinstitution comorphism preserve the semantics of more advanced DOL structuring constructs such as renaming and hiding. The notion of *theoroidal* comorphisms provides a generalisation of the notion of a comorphism: Φ may map signatures to theories (where a theory (Σ, Γ) is a signature Σ equipped with a set Γ of Σ-sentences). We need theoroidal comorphisms to axiomatize some properties of CL models in first-order logic.

Our main motivation for the present work is to address the lack of tool support for CL (see Sect. 2.2). We have thus extended HETS with parsers for CLIF and KIF. The Common Logic Interchange Format (CLIF) provides a Lisp-like syntax for Common Logic, while KIF is an older precursor of Common Logic. Moreover, we have implemented a sublogic analysis and proof support for CL, discussed in detail below.

3.4 Sublogics of Common Logic

By equipping Common Logic (CL) as defined in Sect. 2 with signature morphisms, it can be formalised as an institution, see [19]. We define four subinstitutions (sublogics) of CL, all defined through restrictions on the sentences. Moreover, given a sublogic CL.X, we also define a logic CL.X$^\sharp$ which results from CL.X by eliminating the use of the module construct.

CL.**Full:** full Common Logic,

CL.**Seq:** Common Logic with sequence markers, but without impredicative higher-order like features. That is, in each predication and function application $(t\ s)$, t must be a name.

CL.**Imp:** Common Logic with impredicative features, but without sequence markers.

CL.**Fol:** Common Logic without impredicative features and without sequence markers.

COLORE is mostly written in CL.Fol, but some ontologies use CL.Seq or CL.Imp.

[7]Actually, α and β also have to commute with translations along signature morphisms. Technically, they are natural transformations.

[8]An isomorphism if model morphisms are taken into account.

4 Logic translations involving Common Logic

For CL, no dedicated theorem prover is available. Proof support for CL is obtained in HETS using logic translations to a logic supported by some prover, as discussed next.[9]

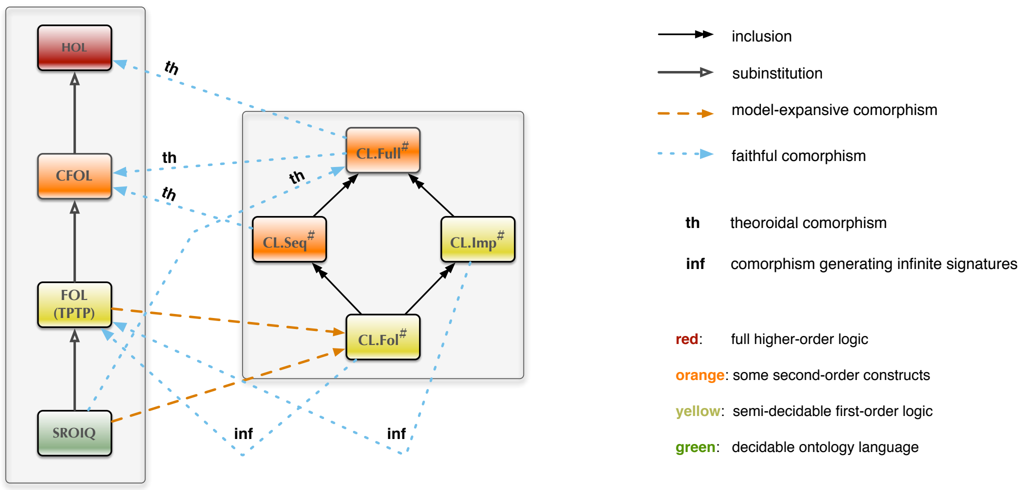
HOL
CFOL
FOL (TPTP)
SROIQ
CL.Full#
CL.Seq#
CL.Imp#
CL.Fol#
th
th
th
th
inf
inf
inclusion
subinstitution
model-expansive comorphism
faithful comorphism
th theoroidal comorphism
inf comorphism generating infinite signatures
red: full higher-order logic
orange: some second-order constructs
yellow: semi-decidable first-order logic
green: decidable ontology language

Figure 1: Graph of logics related to Common Logic that are currently supported by HETS

The logic graph in Fig. 1 is naturally divided into two parts: on the left hand-side, we find classical first- and higher-order logics, on the right hand-side the sublogics of Common Logic. *Within* both parts, we have subinstitution relations: for Common Logic, these are obvious (they are even inclusions), while on the left hand-side, the subinstitutions are a bit more complex. Namely, the subinstitution comorphism from $\mathcal{SROIQ}$ (the logic of OWL 2 [16]) into FOL is the standard translation [18], FOL is an obvious sublogic of CFOL, and the translation from CFOL to HOL expands the generation axioms to explicit Peano-style higher-order induction axioms.

Between the parts of the logic graph, the relations are less tight. Due to the different model theories of classical first- and higher-order logics on the left side and Common Logic on the right, we cannot expect subinstitution comorphisms, but only model-expansive and faithful comorphisms to connect the different parts. We now spell out the comorphisms in more detail. In the following subsections, we define $\mathit{NDNames} = \mathit{Names} \setminus \mathit{DNames}$ as the set of non-discourse names of a CL signature $\Sigma = (\mathit{Names}, \mathit{DNames}, \mathit{Markers})$.

4.1 CL.Fol to FOL

We give a translation of the classical first-order fragment of CL to FOL such that the translated sentences are as similar as possible to the original ones. The intuition behind the translation is to make the two universes (of reference and of discourse) explicit in the translated signature as new sorts, such that the universe of discourse is injectively embedded into the universe of reference. Moreover, we we add function and predicate symbols of any arity for each name, taking also into account the difference between the interpretaton of a name and its interpretation as a constant symbol (see below). This allows us to map the

[9] We also have realised a translation that eliminates the use of CL modules. Since the semantics of CL modules is specific to CL, this elimination of modules is necessary before sending CL theories to a standard first-order prover.

sentences almost identically and to make use of the FOL interpretations of the two sorts and of the function and predicate symbols when defining the CL reduct of a FOL-model of a translated CL signature.

A CL signature $\Sigma = (Names, DNames, \emptyset)$ is translated to a FOL theory $\Phi(\Sigma) = (\Sigma', \Gamma)$ where

- Σ' has two sorts, UR and UD,
- there is a function symbol $inj : UD \rightarrow UR$,
- for each discourse name d there is a constant symbol $[d] : UD$ and for each non-discourse name n there is a constant symbol $[n] : UR$,
- for each $k \in \mathbb{N}$, we have a k-ary function symbol $n : UD^k \rightarrow UD$, capturing the functional interpretation of a name, and a k-ary predicate symbol $n : UD^k$, capturing the relational interpretation of a name.

Γ consists of a sentence asserting injectivity of inj; for each non-discourse name n, the sentence $\forall x : UD.inj(x) \neq [n]$; for each pair of names n_1, n_2, the infinite set of sentences $[n_1] = [n_2] \implies \forall x_1, \ldots, x_k : UD.n_1(x_1, \ldots, x_k) = n_2(x_1, \ldots, x_k)$ where $k \in \mathbb{N}$; and $[n_1] = [n_2] \implies \forall x_1, \ldots, x_k : UD.n_1(x_1, \ldots, x_k) \iff n_2(x_1, \ldots, x_k)$ where $k \in \mathbb{N}$. We have to introduce for each name n both a symbol $[n]$ for its interpretation as a name n and a symbol n for its interpretation as application (n) of a 0-ary function symbol because they can be different in a CL model[10].

Sentences are mapped by induction on their structure and based on the translation of terms. Since we are in the FOL sublogic of Common Logic, we know that on function and predicate positions, only names occur. Therefore, we can define the translation α_Σ of a functional term $(f\ s_1 \ldots s_k)$ as $\alpha_\Sigma((f\ s_1 \ldots s_k)) = f(\alpha_\Sigma(s_1), \ldots, \alpha_\Sigma(s_k))$. Likewise, the translation of a predication $(p\, s_1 \ldots s_k)$ is defined as $\alpha_\Sigma((p\, s_1 \ldots s_k)) = p(\alpha_\Sigma(s_1), \ldots, \alpha_\Sigma(s_k))$. A name n is translated to the constant $[n]$.

For the model reduction component of the comorphism, let Σ be a CL.Fol signature and let N be a $\Phi(\Sigma)$-model. Slightly deviating from our notation for CL, in FOL we will use N_X to denote the interpretation of X in N, where X may be a signature symbol or a term. We will also use the notations $N_{n_f^k}$ (and $N_{n_p^k}$), to make explicit that we refer to the overloaded function (and predicate) symbol n with arity k, for some $k \in \mathbb{N}$. Then $M = \beta_\Sigma(N)$ is defined as follows:

- $UR^M = N_{UR} \uplus \{*\}$,
- $UD^M = N_{inj}(N_{UD})$,
- for any $d \in DNames$, $int^M(d) = N_{inj}(N_{[d]})$ and for any $n \in NDNames$, $int^M(n) = N_{[n]}$.
- for any $x \in UR^M$ and any $s_1, \ldots, s_k \in N_{UD}$, $fun^M(x)(N_{inj}(s_1), \ldots, N_{inj}(s_k)) = N_{inj}(N_{n_f^k}(s_1, \ldots, s_k))$ if there exists a name n such that $int^M(n) = x$[11], and $fun^M(x)(N_{inj}(s_1), \ldots, N_{inj}(s_k)) = *$ otherwise,
- for any $x \in UR^M$ and any $s_1, \ldots, s_k \in N_{UD}$, $(N_{inj}(s_1), \ldots, N_{inj}(s_k)) \in rel^M(x)$ iff $(s_1, \ldots, s_k) \in N_{n_p^k}$ and there exists a name n such that $int^M(n) = x$.

4.2 CL.Imp to FOL

The main idea is that now we add in the translation of a signature function symbols fun and predicate symbols rel taking $k+1$ arguments for every natural number k. There is no need thus to introduce two constants to distinguish between the interpretation of a name and functional intepretation as a constant, like we did for CL.Fol. The drawback is that now rel and fun appear in all translated sentences, but on their first argument any term can occur.

A signature $\Sigma = (Names, DNames, \emptyset)$ is translated to a theory $\Phi(\Sigma) = (\Sigma', \Gamma)$ such that

[10]We could also encode CL.Fol to single-sorted first-order logic and have two predicates UD and UR on the universe instead. While thus the injection inj would not be needed anymore, sentences get more complex, and the definition of the model reduct is not substantially simplified, so we choose not to follow this alternative.

[11]It can happen that n is not unique, but the axioms in Γ always ensure that the particular choice of n is irrelevant.

- Σ' has two sorts *UR* and *UD*,
- there is a function symbol $inj : UD \to UR$,
- for each $d \in DNames$, we have a constant symbol $d : UD$ and for each $n \in NDNames$ we have a constant symbol $n : UR$,
- we have a family of functions $\{fun : UR \times UD^k \to UD\}_{k\in\mathbb{N}}$
- and a family of predicate symbols $\{rel : UR \times UD^k\}_{k\in\mathbb{N}}$
- Γ has the sentence that asserts that *inj* is injective and for each non-discourse name n, the sentence $\forall x : UD.inj(x) \neq n$.

Sentences are translated by induction on their structure, with predications $(t\ s)$ translated as $\alpha_\Sigma((t\ s)) = rel^k(\alpha_\Sigma(t), \alpha_\Sigma(s_1), \ldots, \alpha_\Sigma(s_k))$ and terms $(t\ s)$ translated as $\alpha_\Sigma((t\ s)) = fun^k(\alpha_\Sigma(t), \alpha_\Sigma(s_1), \ldots, \alpha_\Sigma(s_k))$, where s consists of the terms $s_1, \ldots, s_k$[12]. Since there are no sequence markers, the length k of s is always known. Names n are translated identically.

Given a signature Σ, a $\Phi(\Sigma)$-model N reduces to $M = \beta_\Sigma(N)$ as follows:

- $UR^M = N_{UR}$,
- $UD^M = N_{inj}(N_{UD})$,
- for any $d \in DNames$, $int^M(d) = N_{inj}(N_d)$ and for any $n \in NDNames$, $int^M(n) = N_n$,
- for any $x \in UR^M$ and any $s_1, \ldots, s_k \in N_{UD}$, $fun^M(x)(N_{inj}(s_1), \ldots, N_{inj}(s_k)) = N_{inj}(N_{fun^k}(x, s_1, ..., s_k))$
- for any $x \in UR^M$ and any $s_1, \ldots, s_k \in N_{UD}$, $(N_{inj}(s_1), \ldots, N_{inj}(s_k)) \in rel^M(x)$ iff $(x, s_1, ..., s_k) \in N_{rel^k}$.

4.3 Getting finite signatures

We obtain infinite objects by translating CL signatures along the two comorphisms introduced above: infinite theories in the first case and infinite signatures in the second. Of course, a software tool will work only with finite objects. A feature of Common Logic is that signatures are implicitly defined by the symbols used in the sentences of a theory, and reasoning in that theory makes use only of those symbols. We can therefore apply a syntactic transformation α that removes from the signature of a CL theory E all symbols that do not occur in E. For the first comorphism, this means that for each name $n \in Names$, we only introduce a function/predicate symbol of arity k and its corresponding sentences in Γ if the sentences in E contain a term/predication where n takes k arguments. Similarly, for the second comorphism, we keep in the signature only those function symbols fun and predicate symbols *rel* whose arity is given by the terms/predications in E. Since this transformation is only syntactical, we do not need to define a corresponding translation between the class of models of (Σ, E) and $\alpha(\Sigma, E)$. It is however easy to notice that by applying the translation, the logical consequences of (Σ, E) are preserved and reflected by α: a Σ-sentence e is a consequence of E over the signature Σ if and only if it is a consequence of e over the signature $\Phi(\Sigma)$ of $\alpha(\Sigma, E)$ (which also implies that the sentence e itself only uses symbols in $\Phi(\Sigma)$).

4.4 CL.Seq to CFOL

In the presence of markers, we make explicit in the translation of a CL signature a sort of lists of elements of the universe of discourse, axiomatized as a free type to ensure that the interpretation of lists is the expected one. This allows us to distinguish between individuals and sequences. Similarly to the case of the translation of CL.Fol to FOL, we introduce for each name both a constant that gives its interpretation as a name and a function/predicate symbol taking as argument a list. Thus we can distinguish between the name n and its functional application with the empty list as argument.

A signature $\Sigma = (Names, DNames, Markers)$ is translated to a CFOL theory $\Phi(\Sigma) = (\Sigma', \Gamma)$ such that

[12]We made explicit that we use the symbols *fun* and *rel* of arity k.

- $\Phi(\Sigma)$ has three sorts, *UD*, *UR* and *list*,
- there is a function symbol $inj : UD \to UR$,
- on sort *list* we have $nil : list$ and $cons : UD \times list \to list$ and $m : list$ for each $m \in Markers$,
- for each $d \in DNames$ we have a function symbol $d : UD$, for each $n \in NDNames$ we have a function symbol $n : UR$,
- for each $n \in Names$, there is a function symbol $n : list \to UD$ and a predicate symbol $n : list$,
- Γ contains the Peano-style axioms asserting that *list* is a free type over its constructors *nil* and *cons*; moreover, a sentence asserting that *inj* is an injection; for each non-discourse name n, the sentence $\forall x : UD.inj(x) \neq n$; and for each pair of names n_1, n_2, the infinite set of sentences $n_1 = n_2 \implies \forall x_1, \ldots, x_k \in UD.n_1([x_1, \ldots, x_k]) = n_2([x_1, \ldots, x_k])$ where $k \in \mathbb{N}$ and $n_1 = n_2 \implies \forall x_1, \ldots, x_k \in UD.n_1([x_1, \ldots, x_k]) \iff n_2([x_1, \ldots, x_k])$ where $k \in \mathbb{N}$ and $[x_1, \ldots, x_k]$ is a notation for the list with elements $x_1, \ldots, x_k$.

Sentences are mapped by induction on their structure, such that each predication or term $(t\ s)$ is mapped as $\alpha_\Sigma((t\ s)) = t(\alpha_\Sigma(s))$ and each name n is mapped to the constant n. This works because all operators of functional terms and all predicates of predications are names.

Given a $\Phi(\Sigma)$-model N, its reduct $M = \beta_\Sigma(N)$ is defined as follows, using again the notations N_{n_f} and N_{n_p}, to make explicit that we refer to the overloaded function and predicate symbol n.

- $UR^M = N_{UR} \uplus \{*\}$,
- $UD^M = N_{inj}(N_{UD})$,
- for any $d \in DNames$, $int^M(d) = N_{inj}(N_d)$ and for any $n \in NDNames$, $int^M(n) = N_n$,
- for any $x \in UR^M$ and any $s \in (N_{UD})^*$, $fun^M(x)(N_{inj}(s)) = N_{inj}(N_{n_f}(s))$, if there exists a name n such that $int^M(n) = x$ and $fun^M(x)(N_{inj}(s)) = *$ otherwise,
- for any $x \in UR^M$ and any $s \in (N_{UD})^*$, $N_{inj}(s) \in rel^M(x)$ iff $s \in N_{n_p}$ and there exists a name n such that $int^M(n) = x$,
- for any $m \in Markers$, $seq^M(m) = N_{inj}(N_m)$, that is, N_{inj} is applied each element of the list N_m to obtain a new list with elements in UD^M.

4.5 CL.Full to CFOL

Finally, the translation of the full CL to CFOL combines the translations CL.Seq to CFOL and CL.Imp to FOL.

A signature $\Sigma = (Names, DNames, Markers)$ is translated to a CFOL theory $\Phi(\Sigma) = (\Sigma', \Gamma)$ such that

- Σ' has three sorts, *UR*, *UD* and *list*,
- there is a function $inj : UD \to UR$,
- on *UD* we have a constant symbol d for each discourse name $d \in DNames$,
- on *UR* we have a constant symbol n for each non-discourse name $n \in NDNames$,
- on sort *list* we have $nil : list$, $cons : UD \times list \to list$ and $m : list$ for each $m \in Markers$,
- there is a function symbol $fun : UR \times list \to UD$ and a predicate symbol $rel : UR \times list$,
- Γ consists of the Peano-style axioms asserting that *list* is a free type over its constructors *nil* and *cons*, the sentence asserting that *inj* is injective and the sentences asserting for each non-discourse name n that n is not in the image of *UD* along *inj*, i.e. $\forall x : UD.inj(x) \neq n$.

Sentences are translated by induction on their structure, with predications $(t\ s)$ translated to predications $rel(\alpha_\Sigma(t), \alpha_\Sigma(s))$, terms $(t\ s)$ translated to terms $fun(\alpha_\Sigma(t), \alpha_\Sigma(s))$ and names n translated to the corresponding constants n in Σ'.

Given a (Σ', Γ)-model N, its reduct $M = \beta_\Sigma(N)$ is defined as follows:

- $UR^M = N_{UR}$,
- $UD^M = N_{inj}(N_{UD})$, that is, the image of N_{UD} along the injection N_{inj},
- for any $d \in DNames$, $int^M(n) = inj(N_d)$, and for any $n \in NDNames$, $int^M(n) = N_n$. The sentences in Γ ensure that int maps a name x to an element of UD^M iff x is a discourse name,
- for any $x \in UR^M$ and any $s \in (N_{UD})^*, fun^M(x)(N_{inj}(s)) = N_{inj}(N_{fun}(x,s))$,
- for any $x \in UR^M$ and any $s \in (N_{UD})^*$, $N_{inj}(s) \in rel^M(x)$ iff $(x,s) \in N_{rel}$,
- for any $m \in Markers$, $seq^M(m) = N_{inj}(N_m)$.

4.6 Faithful comorphisms

We now come to proving faithfulness of the comorphisms, which ensures we can borrow proofs. For CL.Fol to FOL and CL.*Seq* to CFOL the arguments are similar, so we only present the proof of faithfulness for the second one.

Theorem 1. *The comorphism* CL.*Seq to CFOL is faithful.*

Proof. It suffices to prove that for any CL-signature Σ, if $\alpha_\Sigma(\Gamma) \models^{CFOL}_{\Phi(\Sigma)} \alpha_\Sigma(e)$, then $\Gamma \models^{\mathsf{CL}}_\Sigma e$.

We define a mapping $\delta_\Sigma : \mathbf{Mod}^{\mathsf{CL}}(\Sigma) \to \mathbf{Mod}^{CFOL}(\Phi(\Sigma))$ such that $M \models e$ iff $\delta_\Sigma(M) \models \alpha(e)$ where M is a Σ-model and e is a Σ-sentence.

We denote $N = \delta_\Sigma(M)$. N is defined as follows:

- $N_{UR} = UR^M$,
- $N_{UD} = UD^M$,
- N_{inj} is the inclusion of UD^M in UR^M,
- the interpretation of the sort *list* and of the operations *cons*, *nil* and $++$ is the expected one,
- $N_m = seq^M(m)$ for each $m \in Markers$,
- $N_n = int^M(n)$ for each $n \in Names$
- for each x in *Names* and any $s \in UD^*$, $N_x(s) = fun^M(int^M(x))(s)$,
- for each x in *Names* and any $s \in UD^*$, $s \in N_x$ iff $s \in rel^M(int^M(x))$.

It is easy to see that N satisfies the sentences in $\Phi(\Sigma)$ and $M \models e$ iff $N \models \alpha(e)$ for each Σ-sentence e.

Assume $\alpha_\Sigma(\Gamma) \models^{CFOL}_{\Phi(\Sigma)} \alpha_\Sigma(e)$. In order to show $\Gamma \models^{\mathsf{CL}}_\Sigma e$, let M be a Σ-model such that $M \models^{\mathsf{CL}}_\Sigma \Gamma$. Then $\delta_\Sigma(M) \models^{CFOL}_{\Phi(\Sigma)} \alpha_\Sigma(\Gamma)$. Hence $\delta_\Sigma(M) \models^{CFOL}_{\Phi(\Sigma)} \alpha_\Sigma(e)$, and thus $M \models^{\mathsf{CL}}_\Sigma e$. $\square$

For the other two comorphisms we can prove a stronger result, namely model-expansiveness. The proof proceeds in a similar way as above.

Theorem 2. *The comorphisms* CL.*Imp to CFOL and* CL.*Full to CFOL are model-expansive.*

4.7 FOL to CL.Fol

This comorphism maps classical FOL to CL.Fol.

A FOL signature is translated to CL.Fol by turning all constants into discourse names, and all other function symbols and all predicate symbols into non-discourse names. A FOL sentence is translated to CL.Fol by a straightforward recursion, the base being translations of predications:

$$\alpha_\Sigma(P(t_1,\ldots,t_n)) = (P\ \alpha_\Sigma(t_1)\ \ldots\ \alpha_\Sigma(t_n))$$

Within terms, function applications are translated similarly:

$$\alpha_\Sigma(f(t_1,\ldots,t_n)) = (f\ \alpha_\Sigma(t_1)\ \ldots\ \alpha_\Sigma(t_n))$$

A CL.Fol model is translated to a FOL model by using the universe of discourse as FOL universe. The interpretation of constants is directly given by the interpretation of the corresponding names in CL.Fol. The interpretation of a predicate symbol P is given by using $rel^M(int^M(P))$ and restricting to the arity of P; similarly for function symbols (using fun^M). Both the satisfaction condition and model-expansiveness of the comorphism are straightforward.

4.8 FOL to CFOL

A (single-sorted) FOL signature is mapped to (many-sorted) CFOL by introducing a sort s, and letting all function and predicate symbols be typed by a list of s's, the length of the list corresponding to the arity. The rest is straightforward.

4.9 CFOL to HOL

A CFOL signature is translated to a HOL signature by mapping all sorts to type constants, and all function and predicate symbols to constants of the respective higher-order type. Translation of sentences and models is then straightforward, except for sort generation constraints. The latter can be translated to second-order induction axioms, see [23].

4.10 SROIQ to FOL

This comorphism extends the well-known standard translation for description logics [18, 8] to the additional expressive means of $\mathcal{SROIQ}$. Individuals are translated to constants, concepts to unary predicates and roles to binary predicates. Concepts are translated to formulas with one free variable. FOL-models are translated to $\mathcal{SROIQ}$-models by keeping the universe of discourse, and using the interpretations of constants, unary and binary predicates for interpreting individuals, concepts and roles, respectively.

4.11 SROIQ to CL.Fol

This comorphism can be obtained as the composition SROIQ $\rightarrow$ FOL $\rightarrow$ CL.Fol.

4.12 SROIQ to CL.Full

This comorphism internalises all notions (like Boolean operators on concepts, inverses of roles, etc.) used in $\mathcal{SROIQ}$ using the higher-order like features of CL. Then the translation of sentences becomes trivial.

Due to the theory needed for the axiomatisation of these $\mathcal{SROIQ}$ notions, the comorphism is theoroidal. [13]

A signature is translated by mapping all individuals, concepts and roles to discourse names, and augmenting this with a CL.Imp theory that internalises all operations on concepts that are possible in $\mathcal{SROIQ}$, e.g.

```
(forall (c x) (iff ((OWLnot c) x) (not (c x))))                  // complement
(forall (r c x) (iff ((OWLsome r c) x)           // existential quantification
                     (exists (y) (and (r x y) (c y)))))
(forall (r x) (iff ((OWLself r) x) (r x x)))                // self-restriction
(forall (c d) (iff ((OWLsubsumes c d)          // class subsumption (subclass)
                   (forall (x) (if (c x) (d x))))))
(forall (r) (iff ((OWLirr r)                                    // irreflexivity
                 (forall (x y) (not (r x x))))))

(forall (p) (holds-all p))
(forall (p x ...) (iff (holds-all p x ...)
                       ((and (p x) (holds-all p ...)))))
((same-length))
(forall (x ...) (not ((same-length) x ...)))
(forall (x ...) (not ((same-length x ...))))
(forall (x y ...a ...b) (iff ((same-length x ...a) y ...b)
                             ((same-length ...a) ...b)))

(forall (r c x y) (iff ((restrict r c x) y)
                       (and (r x y) (c y))))
(forall (r c ...) (iff ((OWLmin r c ...) x)             // minimum cardinality
    (exists (...a) (and ((same-length ...) ...a)
                        (holds-all (restrict r c x) ...a)
                        ((distinct ...a))))))

(forall x (not ((OWLnominal) x)))      //  enumeration of individuals (nominals)
(forall ... x y (iff ((OWLnominal y ...) x)
                     (or (= x y) ((OWLnominal ...) x))))

(forall (r x y) (iff ((OWLcomp r) x y) (r x y)))              // property chain
(forall (r ... x y)                                        // (role composition)
   (iff ((OWLcomp r ...) x y)
        (exists (z) (and (r x z) ((OWLcomp ...) z y)))))
```

Note the use of sequence markers for handling lists of variable length. Functions cannot take two sequence markers as argument unless they are written in curried form (otherwise, the two sequences will be concatenated into one argument). Therefore, functions like `same-length` use a curried form; that is, two separate function applications are used for the two arguments, as in `((f x) y)`.

Translation of concepts sentences is then straightforward, e.g.

- $\alpha(\neg C) =$ (`OWLnot` $\alpha(C)$)
- $\alpha(\exists R.C) =$ (`OWLsome` $\alpha(R)$ $\alpha(C)$)

[13]Translations of SROIQ to CL.Full have been introduced in `http://philebus.tamu.edu/cmenzel/Papers/AxiomaticSemantics.pdf` and `http://www.ihmc.us/users/phayes/CL/SW2SCL.html`. Our translation is simpler as in the second reference due to the use of a rich background theory, as in the first reference. Comparing to the latter, we use standard lists, as provided by CL sequences, in contrast to a weak axiomatization of lists in CL that admits non-standard lists as models as well.

- $\alpha(\geq nR.C) = ($`OWLmin` $\alpha(R)\ \alpha(C)\ (a\ldots a))$

Here, the sequence `(a ...a)` repeats an arbitrary name a n times. This is used as a coding of the natural number n. The function `same-length` above is then used for quantifying over all sequences of length n. The term `((same-length c ...)  ...a)` above tests whether `...a` has length $n+1$, where n is encoded by `...`. Note that the value of `c` is irrelevant here, it is just used for increasing the length of the sequence `...` by one.

Models are translated by keeping the universe of discourse, and the interpretation of individuals, concepts and roles are given by the interpretation of the respective names.

4.13 Module elimination

Finally, Fig. 2 shows the square of CL.Full and its sublogics CL.Imp, CL.Seq, and CL.Fol and the square of logics obtained by eliminating the module construct from the languages (denoted by CL.Full# CL.Imp#, CL.Seq#, and CL.Fol#):

- the obtained cube relates CL (and its sublogics) with the respective restrictions;
- logics without a module construct are obtained (essentially) as a re-writing using quantifier restrictions;
- while the restrictions are obviously inclusions, the reverse translations are obtained as subinstitutions.

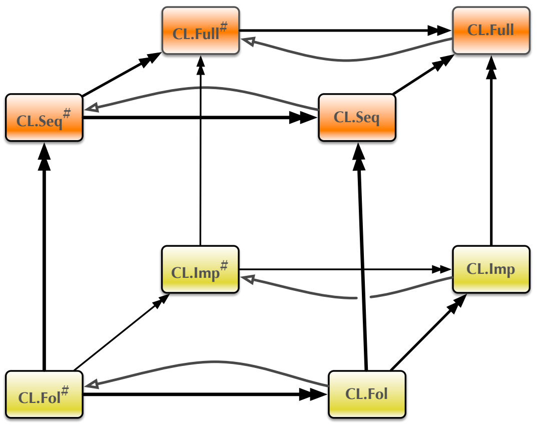

Figure 2: Elimination of the module construct in CL

5 Case Study: Verifying Interpretations in COLORE

We have used Hets for verifying a number of consequences and interpretations of COLORE theories, and for checking their consistency.

The listing below shows a typical example of a COLORE interpretation.[14] Linearly ordered time intervals that begin and end with an instant (`ins:owltime_le`) are interpreted as lines that are incident with linearly ordered points in a special geometry (union of `ord:linear_ordering` and `bi:complete_graphical`), given an appropriate mapping, which is defined as another ontology (`owltime_interval_reduction`). We state that the source ontology can be interpreted in terms of the union of the target ontologies and the mapping ontology in a model-theoretically conservative way, and that the mapping ontology merely extends the target ontologies with definitions.

```
%prefix( %% declaration of prefixes for abbreviating long network identifiers
 %% this distributed ontology (further technical stuff abbreviated with ...)
 :    <http://colore.googlecode.com/.../complex/owltime/owltime_interval/mappings/owltime_le.dol#>
 %% Namespaces of other COLORE ontologies used here:
 bi:  <http://colore.../core/bipartite_incidence/>   ins: <http://.../complex/owltime/owltime_instant/>
 int: <http://.../complex/owltime/owltime_interval/> ord: <http://.../complex/orderings/> )%

%% The following ontologies are in the logic Common Logic
logic CommonLogic

%% The ontology of linearly ordered time intervals that begin and end with an instant extends ...
%% ... linearly ordered time intervals with intervals that begin and end with an instant.
ontology int:owltime_le = int:owltime_linear then int:owltime_e
```

[14]This is an excerpt from `https://colore.googlecode.com/svn/trunk/ontologies/complex/owltime/owltime_interval/mappings/owltime_le.dol` in the COLORE repository. The individual ontologies are actually stored in separate files, but this listing includes them for conciseness, thus demonstrating DOL's ability to maintain different ontologies within one file.

```
%% The ontology of linearly ordered time intervals extends intervals with linearly ordered instants.
ontology int:owltime_linear = int:owltime_interval then ins:owltime_instant_l

%% (int:owltime_e, int:owltime_interval, ins:owltime_instant_l and their imports not shown here)

ontology ord:linear_ordering =        %% Here we use Common Logic’s import construct and add one axiom:
  (cl-imports ord:partial_ordering) (forall (x y) (or (leq x y) (leq y x) (= x y)))

ontology ord:linear_ordering =
  (cl-imports ord:partial_ordering) (forall (x y) (or (leq x y) (leq y x) (= x y))))
  %% (ord:partial_ordering and its imports not shown here)

ontology bi:complete_graphical = bi:grapical_incidence                                   %% see below
  then bi:line_existence                          %% "There is a line through any two distinct points."

ontology bi:graphical_incidence = bi:partial_bipartite    %% "Every line has a point incident with it."
  then bi:double_points          %% a geometry in which at most two points can be incident with a line
  %% The incidence relation (further imports not shown here) is reflexive and symmetric.

ontology int:mappings/owltime_interval_reduction =
  (forall (x) (iff (Instant x) (point x)))              // mapping time instants to geometrical points
  (forall (x) (iff (Interval x) (line x)))              // mapping time intervals to geometrical lines
  // skipping some axioms
  (forall (x y) (iff (begins x y)                                 // x begins with y iff x is a point ...
                   (or (and (point x) (line y) (in x y) // ... that is ≤ any other point incident ...
                            (forall (z) (if (and (point z) (in z y)) (leq x z))))
                       (and (point x) (= x y)))))// ... to the line y, or x and y are the same point.

interpretation geometry_of_time %mcons :  %% Interpretation of linearly ordered time intervals that ...
  int:owltime_le      %% ... begin and end with an instant as lines that are incident with linearly ...
  to { ord:linear_ordering and bi:complete_graphical   %% ... ordered points in a special geometry, ...
     %% ... given an appropriate mapping ontology (defined above), plus the following explicit mapping:
       and %def int:mappings/owltime_interval_reduction } = ProperInterval |-> Interval end
```

In the case of our example, some proof goals introduced when verifying correctness of the interpretation `geometry_of_time` actually end up *disproved*. In the concrete case of the goal `ins:owltime_instants` (corresponding to the theory of partially ordered time instants), Hets finds a countermodel where the predicate `before` is neither irreflexive nor asymmetric, thus violating the strict order axioms one would expect them to satisfy. In the source theory, `before` is axiomatised as intended, whereas the target theory does not do anything besides defining it – via the mapping ontology – equivalent to a predicate `lt`, about which nothing is stated at all. This suggests that an axiomatisation of `lt` is missing, and in fact the problem can be fixed by adding to the target of the interpretation the ontology `ord:orderings_def`, which defines the “less than” operator.

6 Discussion and Outlook

We have established the first full theorem proving support for Common Logic as well as the possibility of verifying meta-theoretical relationships between Common Logic theories via an integration into the HETS system, primarily exploiting the power of logic translation. Using the translations in Sect. 4, we provide proof support for the various sublanguages of Common Logic. Some sublanguages can be translated into classical first-order logic, such that automated theorem provers like SPASS, Vampire or Darwin can be used. Via faithfulness of the translations, soundness and completeness of this proof support for Common Logic is inherited from that for the provers w.r.t. FOL. Other sublanguages need provers with induction capabilities, or higher-order provers like Leo-II or Isabelle. This is still sound, but only complete w.r.t. Henkin semantics for HOL, i.e. a semantics allowing for models with non-standard sequences. By Gödel’s incompleteness theorem, we cannot expect more.

All these translations and provers are integrated within HETS. As CL is a popular language within ontology communities interested in greater expressive power than provided by the decidable OWL DL language, this tool support for CL is a substantial step towards supporting more ambitious ontology engineering efforts. As a first evaluation of the provided CL reasoning support, we have used HETS for verifying consequences of and interpretations between COLORE theories, and for checking their consistency. During this process, numerous errors in COLORE have been found and corrected (including parsing errors, wrong symbol mappings, missing Boolean operators, and inconsistencies). The sublogic analysis for Common Logic provided by HETS was of particular importance here, because automation and efficiency of proofs greatly varies among the sublogics. Most proof goals in the CL.Fol theories of COLORE could be proved using SPASS, while for COLORE's graph theories involving recursive use of sequence markers, the interactive theorem prover Isabelle needed to be used.

Related work includes the bridges of provers like Isabelle and Ωmega to automated theorem proving (ATP) systems like SPASS. HETS also provides bridges from CL (and other logics) to ATP systems. However, while other systems have the bridges built-in in a hard-coded way, HETS realises bridges as institution comorphisms and supports them as first-class citizens. Hence, HETS offers a greater flexibility by letting the user chose among different bridges (or even develop new ones). We have exploited this flexibility by providing different bridges, depending on the sublogic of CL in which a given theory is formulated.

Future work should analyse non-recursive uses of sequence markers (as they occur in theories that are generic over the arity of certain predicates and functions) more carefully and provide automated first-order proof support for these. We also plan to integrate our work into the web ontology repository engine `ontohub.org` and exploit more explicitly the structuring mechanisms of DOL in connection with CL.

Acknowledgements. Work on this paper was supported by the DFG-funded Research Center on Spatial Cognition (SFB/TR 8). We thank Fabian Neuhaus for his detailed comments on several versions of this paper, and to Christian Maeder, Sören Schulze, Eugen Kuksa for contributing to the implementation of support of CL in HETS. We gratefully acknowledge the financial support of the Future and Emerging Technologies (FET) programme within the Seventh Framework Programme for Research of the European Commission, under FET-Open Grant number: 611553, project COINVENT.

References

[1] Michael Balser, Wolfgang Reif, Gerhard Schellhorn, and Kurt Stenzel. Kiv 3.0 for provably correct systems. In Dieter Hutter, Werner Stephan, Paolo Traverso, and Markus Ullmann, editors, *FM-Trends*, volume 1641 of *Lecture Notes in Computer Science*, pages 330–337. Springer, 1998.

[2] Peter Baumgartner, Alexander Fuchs, and Cesare Tinelli. Implementing the Model Evolution Calculus. *Special Issue of the International Journal of Artificial Intelligence Tools (IJAIT)*, 15(1), 2005.

[3] C. Benzmüller, L. C. Paulson, F. Theiss, and A. Fietzke. LEO-II - A Cooperative Automatic Theorem Prover for Classical Higher-Order Logic (System Description). In Alessandro Armando, Peter Baumgartner, and Gilles Dowek, editors, *IJCAR*, volume 5195 of *LNCS*, pages 162–170. Springer, 2008.

[4] Christoph Benzmüller and Adam Pease. Higher-order aspects and context in SUMO. *Journal of Web Semantics (Special Issue on Reasoning with context in the Semantic Web)*, 12-13:104–117, 2012.

[5] Christoph Benzmüller, Florian Rabe, and Geoff Sutcliffe. THF0 – the core of the TPTP language for higher-order logic. In Alessandro Armando, Peter Baumgartner, and Gilles Dowek, editors, *IJCAR 2008*, volume 5195 of *Lecture Notes in Computer Science*, pages 491–506. Springer, 2008.

[6] T. Berners-Lee, J. Hendler, and O. Lassila. The Semantic Web. *Scientific American*, May 2001.

[7] M. Bidoit and P. D. Mosses. CASL *User Manual*, volume 2900 of *LNCS*. Springer, 2004. `www.cofi.info`.

[8] A. Borgida. On the relative expressiveness of description logics and predicate logics. *Artificial Intelligence*, 82:353–367, 1996. Research Note.

[9] T. Borzyszkowski. Higher-order logic and theorem proving for structured specifications. In D. Bert, C. Choppy, and P. D. Mosses, editors, *WADT*, volume 1827 of *LNCS*, pages 401–418. Springer, 1999.

[10] Chad E. Brown. Satallax: An automatic higher-order prover. In Bernhard Gramlich, Dale Miller, and Uli Sattler, editors, *IJCAR*, volume 7364 of *LNCS*, pages 111–117. Springer, 2012.

[11] Mihai Codescu, Fulya Horozal, Michael Kohlhase, Till Mossakowski, Florian Rabe, and Kristina Sojakova. Towards logical frameworks in the heterogeneous tool set hets. In Till Mossakowski and Hans-Jörg Kreowski, editors, *WADT 2010*, volume 7137 of *Lecture Notes in Computer Science*, pages 139–159. Springer, 2012.

[12] Information technology — Common Logic (CL): a framework for a family of logic-based languages. Technical Report 24707:2007, ISO/IEC, 2007. `http://iso-commonlogic.org`.

[13] J. Goguen and G. Roşu. Institution morphisms. *Formal aspects of computing*, 13:274–307, 2002.

[14] J. A. Goguen and R. M. Burstall. Institutions: Abstract model theory for specification and programming. *Journal of the Association for Computing Machinery*, 39:95–146, 1992.

[15] R. Harper, F. Honsell, and G. Plotkin. A framework for defining logics. *Journal of the Association for Computing Machinery*, 40(1):143–184, 1993.

[16] I. Horrocks, O. Kutz, and U. Sattler. The Even More Irresistible $\mathcal{SROIQ}$. In *Proc. of the 10th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR2006)*, pages 57–67. AAAI Press, June 2006.

[17] Megan Katsumi. A methodology for the development and verification of expressive ontologies. M. Sc. Thesis, Department of Mechanical and Industrial Engineering, University of Toronto, 2011.

[18] Yevgeny Kazakov. RIQ and SROIQ Are Harder than SHOIQ. In Gerhard Brewka and Jérôme Lang, editors, *KR*, pages 274–284. AAAI Press, 2008.

[19] O. Kutz, T. Mossakowski, and D. Lücke. Carnap, Goguen, and the Hyperontologies: Logical Pluralism and Heterogeneous Structuring in Ontology Design. *Logica Universalis*, 4(2):255–333, 2010.

[20] Klaus Lüttich and Till Mossakowski. Reasoning Support for CASL with Automated Theorem Proving Systems. In J. Fiadeiro, editor, *WADT 2006*, number 4409 in LNCS, pages 74–91. Springer, 2007.

[21] Christopher Menzel. Knowledge representation, the World Wide Web, and the evolution of logic. *Synthese*, 182:269–295, 2011.

[22] T. Mossakowski. Heterogeneous specification and the heterogeneous tool set. Habilitation thesis, University of Bremen, 2005.

[23] Till Mossakowski. Relating CASL with other specification languages: the institution level. *Theoretical Computer Science*, 286:367–475, 2002.

[24] Till Mossakowski, Christoph Lange, and Oliver Kutz. Three Semantics for the Core of the Distributed Ontology Language. In Maureen Donnelly and Giancarlo Guizzardi, editors, *FOIS 2012*, volume 239 of *Frontiers in Artificial Intelligence and Applications*, pages 337–352. IOS Press, 2012. FOIS Best Paper Award.

[25] Till Mossakowski, Christian Maeder, and Mihai Codescu. Hets user guide, 2011. See `hets.eu`.

[26] Till Mossakowski, Christian Maeder, and Klaus Lüttich. The Heterogeneous Tool Set. In Orna Grumberg and Michael Huth, editors, *TACAS 2007*, volume 4424 of *LNCS*, pages 519–522. Springer, 2007.

[27] Till Mossakowski, Christian Maeder, and Klaus Lüttich. The Heterogeneous Tool Set. In Bernhard Beckert, editor, *VERIFY 2007*, volume 259 of *CEUR Workshop Proceedings*. 2007.

[28] Peter D. Mosses, editor. *CASL Reference Manual*, volume 2960 of *Lecture Notes in Computer Science*. Springer, 2004. Free online version available at `http://www.cofi.info`.

[29] Fabian Neuhaus and Pat Hayes. Common logic and the Horatio problem. *Applied Ontology*, 7(2):211–231, 2012.

[30] T. Nipkow, L. C. Paulson, and M. Wenzel. *Isabelle/HOL — A Proof Assistant for Higher-Order Logic*. Springer Verlag, 2002.

[31] OWL Working Group. OWL 2 web ontology language: Document overview. W3C recommendation, World Wide Web Consortium (W3C), October 2009.

[32] Björn Pelzer and Christoph Wernhard. System description: E-KRHyper. In Frank Pfenning, editor, *CADE*, volume 4603 of *Lecture Notes in Computer Science*, pages 508–513. Springer, 2007.

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

[34] C. Ross. COLORE system architecture. Available at `http://ontolog.cim3.net/cgi-bin/wiki.pl?OpenOntologyRepository_Architecture/From_COLORE`.

[35] S. Schulz. E – A Brainiac Theorem Prover. *Journal of AI Communications*, 15(2/3):111–126, 2002.

[36] Geoff Sutcliffe. The TPTP world – infrastructure for automated reasoning. In Edmund M. Clarke and Andrei Voronkov, editors, *LPAR (Dakar)*, volume 6355 of *LNCS*, pages 1–12. Springer, 2010.

[37] C. Weidenbach, U. Brahm, T. Hillenbrand, E. Keen, C. Theobalt, and D. Topic. SPASS version 2.0. In A. Voronkov, editor, *Automated Deduction – CADE-18*, LNCS 2392, pages 275–279, 2002.

[38] Jürgen Zimmer and Serge Autexier. The MathServe System for Semantic Web Reasoning Services. In U. Furbach and N. Shankar, editors, *3rd IJCAR*, LNCS 4130. Springer, 2006.

Embedding of Quantified Higher-Order Nominal Modal Logic into Classical Higher-Order Logic*

Max Wisniewski[1] and Alexander Steen[2]

[1] Freie Universität Berlin, Germany
max.wisniewski@fu-berlin.de
[2] Freie Universität Berlin, Germany
a.steen@fu-berlin.de

Abstract

In this paper, we present an embedding of higher-order nominal modal logic into classical higher-order logic and study its automation. There exists no automated theorem prover for first-order or higher-order nominal logic at the moment, hence, this is the first automation for this kind of logic. In our work, we focus on nominal tense logic and have successfully proven some first theorems.

1 Introduction

In writing ATP systems for non-classical logics, it is a common approach to develop special calculi and prover for each logic. Apart from the implementation work, this imposes the need to prove completeness and soundness for each calculus individually. A different approach is to embed a logic in higher-order logic (HOL), which had recently been a notable success (see, e.g., [1]). A major benefit is that there is no need to build new ATPs, but only to write a translation from one calculus to another.

In this paper we modified an embedding based approach for ordinary modal logic (see [2, 1]) and employed it for the embedding of higher-order nominal modal logic (HONL). Surprisingly, in our translation we observed a problem with the valuation of nominals. We propose two solutions to this problem. Lastly, we implemented the embedding for a special kind of HONL, the higher-order nominal tense logic (HONTL), a higher-order version of the nominal tense logic used by Blackburn [9].

As a novelty, our approach (up to our knowledge) implements the first reasoner for higher-order nominal logic. Already existing ATPs, such as hylotab [4], htab [7] and spartacus [5] are restricted to propositional nominal logic.

In section 2 we briefly present the HONL syntax and semantics including HONTL as a special case. In section 3, we survey the embedding of ordinary modal logic into HOL to encourage our own embedding. Finally, we present the embedding of HONL and some of our tests using the Isabelle/HOL proof assistant [8].

2 Higher-Order Nominal Logic

Nominal (modal) logic, often referred to as *hybrid logic*, is a general term for extensions of ordinary modal logics. The nominal logic considered here, introduces a new sort of constant symbols, the so-called *nominals*, that are only true at one possible world and false at every other. This logic is often denoted $\mathcal{H}$ and is the simplest of the nominal extensions [3].

*This work has been supported by the German National Research Foundation (DFG) under grant BE 2501/11-1 (LEO-III).

Early forms of nominal logic originated from Arthur Prior's research on tense logics [10] and were further developed by Robert Bull [3]. Since then, hybrid logics have intensively been studied by several others, including Valentin Goranko [11] and Patrick Blackburn [3, 9].

In contrast to most hybrid logic literature, we consider a simple extension to higher-order modal logics (HOML), rather than to propositional variants. The resulting logic is denoted HONL (for *higher-order nominal logic*). We closely follow the notation for higher-order logics used in [1], where also a brief introduction can be found.

Definition 1 Let I be some index set and T the set of simple types, freely generated from $\{o, \mu\}$ (where o is the type of Booleans and μ the type of individuals) and the right-associative function type constructor $\rightarrow$. The grammar for HONL is given by ($\alpha, \beta \in T, i \in I$):

$$\begin{aligned} s, t ::= \quad & p_\alpha \mid n_o \mid X_\alpha \mid (\lambda X_\alpha . s_\beta)_{\alpha \rightarrow \beta} \mid (s_{\alpha \rightarrow \beta} t_\alpha)_\beta \mid (\neg_{o \rightarrow o} s_o)_o \mid \\ & ((\vee_{o \rightarrow o \rightarrow o} s_o) t_o)_o \mid \forall_{(\alpha \rightarrow o) \rightarrow o} (\lambda X_\alpha . s_o))_o \mid (\Box^i_{o \rightarrow o} s_o)_o \end{aligned}$$

The symbols p_α, X_α and n_o denote constant symbols, variables, and nominals respectively. Further operators such as $\wedge, \rightarrow, \leftrightarrow, \ldots$ can be defined in the usual way. The terms of type o are called *formulas*.

The semantics can be adopted in large parts from ordinary higher-order modal logics, except that we restrict the interpretation of nominals:

A *model* M for HONL is a pair $M = \langle W, \{R_i\}_{i \in I}, \mathcal{D}, \{\mathcal{I}_w\}_{w \in W} \rangle$ where W is the (non-empty) set of possible worlds with each R_i being a accessibility relation between them. $\mathcal{D}$ is a collection of sets $\mathcal{D}_\alpha$, for each $\alpha \in T$, with $\mathcal{D}_o = \{T, F\}$ (for truth and falsehood) and $\mathcal{D}_{\alpha \rightarrow \beta}$ a set of functions from $\mathcal{D}_\alpha$ to $\mathcal{D}_\beta$. Each $\mathcal{I}_w$ is an interpretation mapping each p_α to its denotation in $\mathcal{D}_\alpha$ (depending on the world w it is interpreted in). As a special case, every nominal n_o is assigned a value in $\{T, F\}$, such that each n_o is mapped to T by exactly one $\mathcal{I}_w$ and to F by every other interpretation.

A valuation g assigns each variable X_α to an object in $\mathcal{D}_\alpha$. An X-variant of g is a valuation $g[a \,/\, X_\alpha]$, that maps each symbol $Y_\alpha \neq X_\alpha$ to $g(Y_\alpha) \in \mathcal{D}_\alpha$, except that X_α is mapped to $a \in \mathcal{D}_\alpha$. We assume the usual β- and η-reduction with the associated β- and $\beta\eta$-normal form.

Definition 2 The value of s_o in world w and model M, with valuation g, is denoted $\|s_o\|^{M,g,w}$ and defined by:

1. $\|p_\alpha\|^{M,g,w} = \mathcal{I}_w(p_\alpha)$ and $\|n_o\|^{M,g,w} = \mathcal{I}_w(n_o)$
2. $\|X_\alpha\|^{M,g,w} = g(X_\alpha)$
3. $\|(s_{\alpha \rightarrow \beta} t_\alpha)\|^{M,g,w} = \|s_{\alpha_\beta}\|^{M,g,w}(\|t_\alpha\|^{M,g,w})$
4. $\|\lambda X_\alpha . s_\beta\|^{M,g,w} = f \in D_{\alpha \rightarrow \beta}$ s.t. $\forall d \in D_\alpha : \; f(d) = \|s_\beta\|^{M,g[d \,/\, X_\alpha],w}$
5. $\|(\neg s_o)\|^{M,g,w} = T$ iff $\|s_o\|^{M,g,w} = F$
6. $\|\forall_{(\alpha \rightarrow o) \rightarrow o} (\lambda X_\alpha . s_o)\|^{M,g,w} = T \Leftrightarrow \forall d \in D_\alpha : \; \|s_o\|^{M,g[d \,/\, X_\alpha],w} = T$
7. $\|\Box^i_{o \rightarrow o} s_o\|^{M,g,w} = T \Leftrightarrow \forall v \in W : \; R_i \, w \, v \Rightarrow \|s_o\|^{M,g,v}$

We use the same definition for validity (in models) as given in [1].

The constructed language HONL corresponds to the nominal extension of modal logic K. As usual, we can enrich the conditions on frames by introducing properties on the accessibility relations (e.g. transitivity or symmetry).

The extension with nominals allows us to express formulas that explicitly talk about the world they are evaluated in. As an example, consider the formula "If I am at world w, P_o must hold". This formula cannot be expressed in modal logic – but if we have a nominal w_o (that is appropriately mapped), we can express it simply as $w_o \rightarrow P_o$. Furthermore, with the help of nominals, we can express certain frame conditions that cannot be formulated in ordinary modal logic, e.g. irreflexity or antisymmetry [9]. Thus, without any further extensions, HONL is already more expressive than HOML.

Often, a logic with special modal operator L (called *shifter*) with S5 context is considered [9]. With this operator, formulas like $L(1941_o \rightarrow \text{(Zuse completed Z3)}_o)$ can easily be formulated. Hence, L shifts to the world denoted by 1941_o and checks whether the given formula is true in exactly this world.

Nominal Tense Logic Higher-Order Nominal Tense Logic (HONTL), the higher-order variant of *Tense logic* K_t [6], can be interpreted as a restricted version of HONL. Here, we introduce two modal operators G and H (for $G = \Box_1, G = \Box_2$), as *goes on* and *has been*, with the dual operators $F = \neg G \neg$ and $P = \neg H \neg$ as *in the future* and *in the past* respectively.

Together with the two axioms $p \rightarrow HFp$ and $p \rightarrow GPp$, HONTL is already completely characterized.

3 Embedding

Ordinary Modal Logic. The embedding of HOML in HOL was described by Benzmüller and Woltzenlogel Paleo [1]. The main idea is to lift the truth (of type o) of a formula to be dependent on the world it is evaluated in (to a type $\sigma := \iota \rightarrow o$) and to introduce possible worlds as dedicated objects of a new type ι.

Definition 3. Let $\lceil . \rceil$ be the function that type-raises HOML terms. For each accessibility relation i a new constant symbol $r^i_{\iota\rightarrow\iota\rightarrow o}$ is introduced. The lifting translations of the connectives are as follows

$$\begin{aligned}
\lceil \neg \rceil_{\sigma\rightarrow\sigma} &= \lambda s_\sigma . \lambda W_\iota . \neg(s\, W) \\
\lceil \vee \rceil_{\sigma\rightarrow\sigma\rightarrow\sigma} &= \lambda s_\sigma . \lambda t_\sigma . \lambda W_\iota . (s\, W) \vee (t\, W) \\
\lceil \forall \rceil_{(\alpha\rightarrow\sigma)\rightarrow\sigma} &= \lambda s_{\alpha\rightarrow\sigma} . \lambda W_\iota . \forall(\lambda X_\alpha . s\, X\, W) \\
\lceil \Box^i \rceil_{\sigma\rightarrow\sigma} &= \lambda s_\sigma . \lambda W_\iota . \forall V_\iota . \neg(r^i_{\iota\rightarrow\iota\rightarrow o}\, W\, V) \vee s\, V
\end{aligned}$$

As it can be seen above, the embedding makes the possible worlds explicit in the HOL definitions. The predicate **valid** $= \lambda s_\sigma . \forall(\lambda W_\iota . s\, W)$ realizes the validity of a formula s_σ in HOL. As abbreviation, the notion $[s_\sigma]$ is used. The embedding of HOML is sound and complete, i.e. $\models_{HOML} s_o \Leftrightarrow \models_{HOL} [\lceil s_o \rceil]$, as shown in [1].

HONL Embedding. In order to expand the embedding of HOML to HONL, we only have to address the valuation of nominals. The only change in the semantics is that the valuation of nominals maps to singleton sets of worlds, i.e. is a function from nominals to worlds.

Where in the HONL syntax nominals are objects of type o, their lifted equivalents are assigned a new distinct type η. We hereby emphasize that the truth-value of a nominal n_η depends on the world it is evaluated in. The lifting of these objects is given by $\lceil n_o \rceil = \langle n_\eta \rangle = \lambda W_\iota . \lambda V_{\eta\rightarrow\iota} . (W = V n_\eta)$. For the valuation used inside the lifting definition we studied two approaches:

1. We choose the valuation $V_{\eta\to\iota}$ to be a globally fixed function, called $worldAt_{\eta\to\iota}$. We use the above approach, but replace the lifting immediately to $\langle n_\eta \rangle = \lambda W_\iota.(W = worldAt\, n_\eta)$.
2. As for worlds, we pass a function $V_{\eta\to\iota}$ through the formula. Each term is further lifted to take not only worlds, but nominal valuation functions as well. The validity predicate is then adjusted to be
$$\textbf{valid} = \lambda s_{\iota\to(\eta\to\iota)\to o}.\forall W_\iota.\forall V_{\eta\to\iota}.s\, W\, V$$

Both approaches abstract the valuation of nominals out of the formula, by introducing the valuation as part of the embedding, hence, reducing it to HOML.

The second approach clearly preserves the semantics, for the quantification over all valuations is done explicitly. The first approach does only work if we (a) work with a fixed valuation or (b) quantify over all nominals. For (b), to preserve the semantics, we have to require surjectivity for *worldAt*. The main idea behind this interpretation is that a quantification over all valuations will ultimately lead a quantified nominal to be true at each world at least once. Hence, we do not lose generality.

4 Tests

We used Isabelle/HOL [8] to implement the embedding described in Section 3. As for the theoretical part, we modified the Isabelle/HOL embedding of Benzmüller and Woltzenlogel Paleo[1] for nominal logic. In this section, we present some tests of our embedding[2].
The first test was to check some frame-condition correspondences (see [9]) that can now be expressed in our approach. Figure 1 shows the formulation of the respective theorems for irreflexivity and antisymmetry correspondences (the operators with a `t`-prefix, e.g. `t→`, are the representation of the associated type-lifted operators and `t<` is the accessibility relation).

```
lemma L1a:
  assumes "∀i.[<i> t→ t¬ (F <i>)]"
  shows "∀x.¬(x t< x)"

lemma L2a:
  assumes "∀i.[<i> t→ t¬ (F (F <i>))]"
  shows "∀x.∀y. x t< y ⟶ ¬(y t< x)"
```

```
lemma L1b:
  assumes "∀x.¬(x t< x)"
  shows "∀i.[<i> t→ t¬ (F <i>)]"

lemma L2b:
  assumes "∀x.∀y. x t< y ⟶ ¬(y t< x)"
  shows "∀i.[<i> t→ t¬ (F (F <i>))]"
```

Figure 1: Isabelle formulation of irreflexity (left), antisymmetry (right)

These correspondences were proven in both possible interpretations for the valuation of nominals, except for one lemma. Interestingly, this one was able to be proven by metis itself (with hints) but several other ATPs via sledgehammer gave up. Surprisingly the direction `L*b` is faster in the quantified approach, but without hints it is impossible to proof the `L*a` direction. The time for proving these lemmas was mostly in the area of 10ms with the fixed valuation (variant (1)) and took up to 840ms (right direct correspondence). The second valuation method took 3-4ms in `L*b` direction but around 40ms for the `L*a` direction with hints.

Next, we tested the framework on a simple example with quantification. We introduced an individual W(ashington) and stated that this person was not president at the Independence

[1]The original Isabelle embedding by Benzmüller and Woltzenlogel Paleo can be found at `https://github.com/FormalTheology/GoedelGod/tree/master/Formalizations/Isabelle`

[2]Our embedding can be found at `http://mwisnie.userpage.fu-berlin.de/logic/honl-embedding/nominal-embedding.tar`

```
consts indDayTime :: "ι"
   federalHallTime :: "ι"
   independenceDay :: "η"
   federalHall :: "η"
   Pr :: "μ ⇒ σ"
  W :: "μ ⇒ σ"

consts w :: "μ"
axiomatization where
   washington: "[W w]" and
   w1: "¬((Pr w)indDayTime)" and
   w2: "(Pr w)federalHallTime"
```

```
lemma WwillAlwaysHaveBeenPresident:
   shows "[L(<federalHall> t→ Pr w)]"

lemma WwasPresident:
   shows "[∃(λx.L(<federalHall> t→(Pr x)))]"

lemma WwasnotPresidentBefore:
   shows "[∃(λx.L(<federalHall>t→P(t¬(Pr x))))]"
```

Figure 2: Isabelle example problem: When was Washington president?

Day, but after his inauguration he was, of course, president. The problems in Figure 2 were successfully proven automatically. In the proof context we keep the valuation fixed by the defined axioms. Therefore, we chose the first approach and formulated the valuation as a globally fixed function.

5 Conclusion

We were able to adopt the embedding of HOML to HONL and thus use existing higher-order ATP to reason about nominal logic. The first tests were promising: Basic correspondences and tests were automatically proven. To give a full evaluation of this attempt, more experiments have to be carried out. Especially the pros and cons of both presented approaches and possible other options have to be evaluated.

References

[1] C. Benzmüller and B. Woltzenlogel Paleo. Automating Gödel's ontological proof of god's existence with higher-order automated theorem prover. In *21st European Conference on Artificial Intelligence (ECAI)*, 2014. (available at http://page.mi.fu-berlin.de/cbenzmueller/papers/C40.pdf).

[2] C. Benzmüller and L. Paulson. Quantified multimodal logics in simple type theory. *Logica Universalis (Special Issue on Multimodal Logics)*, 7(1):7–20, 2013.

[3] P. Blackburn, J.F.A.K. van Benthem, and F. Wolter. *Handbook of Modal Logic.* Studies in Logic and Practical Reasoning. Elsevier Science, 2006.

[4] J. V. Eijck. Hylotabtableau-based theorem proving for hybrid logics. Technical report, 2002.

[5] D. Götzmann et al. Spartacus: A tableau prover for hybrid logic. *Electronic Notes in Theoretical Computer Science*, 262(0):127 – 139, 2010.

[6] A. Galton. Temporal logic. In Edward N. Zalta, editor, *The Stanford Encyclopedia of Philosophy.* Fall 2008 edition, 2008.

[7] G. Hoffmann and C. Areces. Htab: a terminating tableaux system for hybrid logic. *Electr. Notes Theor. Comput. Sci.*, 231:3–19, 2009.

[8] T. Nipkow, L. C. Paulson, and M. Wenzel. *Isabelle/HOL — A Proof Assistant for Higher-Order Logic*, volume 2283 of *LNCS.* Springer, 2002.

[9] P.Blackburn. Nominal tense logic. *Notre Dame Journal of Formal Logic*, 34(1):56–83, 1993.

[10] A.N. Prior. *Papers on time and tense.* Oxford books. Clarendon P., 1968.

[11] V.Goranko. Temporal logic with reference pointers. In *Proceedings of the 1st International Conference on Temporal Logic, volume 827 of LNAI*, pages 133–148. Springer.

Dialogues for proof search

Jesse Alama[1*]

Theory and Logic Group
Technical University of Vienna
Vienna, Austria
`alama@logic.at`

Abstract

Dialogue games are a two-player semantics for a variety of logics, including intuitionistic and classical logic. Dialogues can be viewed as a kind of analytic calculus not unlike tableaux. Although dialogues can serve as an alternative characterization of intuitionistic logic, it is less clear to what extent dialogues can be practically used as a search procedure. We announce Kuno, an automated theorem prover for intuitionistic first-order logic based on dialogue games.

1 Introduction

Dialogue games ("dialogues" for short) are a game-theoretic semantics for intuitionistic logic. The game starts with a logical formula ϕ asserted by the Proponent (**P**), who takes the stance that the formula is valid, against the Opponent (**O**) who disputes this. The players take turns and move according to certain rules. As with other game-based semantics for logics, the focus is less at the level of a particular play of the game (sequence of moves that follow the rules) and more at the level of strategy (a way of playing the game to ensure a certain outcome). The main result about dialogue games that is of interest for us here is [3, 4]:

Theorem 1. *A formula ϕ is intuitionistically valid iff there exists a winning strategy for ϕ.*

(A winning strategy for ϕ is a way of playing a dialogue game such that, for every move that **O** can make, **P** can respond in such a way that a win [for **P**] is ensured.)

The dialogical approach to logic ("dialogical logic", for short) is found mainly in philosophical discussions about logic and semantics. Dialogues differ from other game-based approaches to logic, such as Hintikka-style games. There, for instance, players have certain roles (e.g., "Abelard" and "Eloise") that can switch as the game proceeds; in dialogue games, the two players **P** and **O** play the same "role" throughout the game (they do not swap sides).

The question in focus in the present paper is: *Can dialogue games be profitably understood as a proof search calculus?* Although dialogues can be used to characterize intuitionistic logic (and a handful of other logics, as well), it is less clear whether dialogues can be a practical basis for proof search. By viewing dialogues in this way, we put new "pressure" on the basic notions of the field. We also hope to reap fresh insights into the foundations of dialogical logic by seeing how they respond to the pressures, so to speak, of everyday proof search.

The fruit of our efforts is Kuno, an automated theorem prover for first-order intuitionistic logic that is based on dialogue games. Section 2 discusses the game calculus. We shall see that dialogue games are essentially a kind of analytic calculus with similarities to tableaux. Hypersequent calculi for various intermediate logics can also be seen from a dialogical perspective [6, 5]. Section 3 briefly discusses the implementation and operation of Kuno. Section 4 evaluates Kuno

*Supported by FWF grant P25417-G15 (LOGFRADIG).

Assertion	**Attack**	**Response**
$\phi \wedge \psi$	$\wedge_L$	ϕ
	$\wedge_R$	ψ
$\phi \vee \psi$	$?$	ϕ or ψ
$\phi \to \psi$	ϕ	ψ
$\neg\phi$	ϕ	—
$\forall x \phi$	$?_t$	$\phi[t/x]$
$\exists x \phi$	$?$	$\phi[t/x]$

Table 1: Particle rules for dialogue games

on a part of the Intuitionistic Logic Theorem Proving Library (ILTP), a collection of theorem proving problems [11, 12]. Section 5 concludes with further problems to be solved.

`Kuno` is available online at `http://github.com/jessealama/dialogues`.

2 Dialogue games for intuitionistic first-order logic

For a thorough introduction to dialogical logic, see [7] or [10]. We work in a first-order language built from $\forall$, $\exists$, $\neg$, $\vee$, $\wedge$, and $\to$. It is assumed that equality is not present, nor $\bot$, nor $\top$. (All of these restriction can be lifted by suitable rewritings, but for the sake of simplicity we give the traditional formulation of dialogues.) Dialogue games involve not just formulas, but also so-called *symbolic attacks* ? (akin to asking "which?"), $\wedge_L$, and $\wedge_R$ (akin to prompting the other player "defend the left-hand side" or "defend the right-hand side"). Together formulas and symbolic attacks are called *statements*; they are what is asserted by **P** and **O** in a dialogue game.

The rules governing dialogues are divided into two types. *Particle rules* can be seen as specifying the meaning of connectives in a local fashion and say how formulas can be attacked and defended depending on their main connective. By contrast, *structural rules* operate globally and define what sequences of attacks and defenses count as dialogues, thus giving a kind of global meaning to the connectives.

Dialogue games start with the Proponent (**P**) making the initial assertion. Play alternates between **P** and the Opponent **O**. Every move is either an attack on something previously asserted or a defense against an attack. The standard particle rules are given in Table 1. According to the first row, there are two possible attacks against a conjunction: The attacker specifies whether the left or the right conjunct is to be defended, and the defender then continues the game by asserting the specified conjunct. The second row says that there is one attack against a disjunction; the defender then chooses which disjunct to assert. The interpretation of the third row is straightforward. The fourth row says that there is no way to defend against the attack against a negation; the only appropriate "defense" against an attack on a negation $\neg\phi$ is to continue the game with the new information ϕ. The particle rule for $\forall$ says that the challenger picks an instance (a term) and it is up to the original claimant to defend the instance of the universal generalization. For $\exists$, the challenge is simply: "which one?" The way to proceed is to pick an instance of the existential generalization.

The structural rules are:

- **P** may assert an atomic formula only if **O** has asserted it earlier.

- Only the most recent open attack may be defended. (An attack is open if there there is no defense against it.)
- Attacks may be defended only once.
- **P**'s assertions may be attacked at most once.

The game ends if no possible move can be made. If **O** cannot move, then it is said that **P** has won the game; if **P** cannot move, then **O** has won the game. (It is possible, even at the propositional level, for games to go on infinitely, with neither player winning.)

In addition to these standard structural rules, another rule is often considered in connection with dialogues:

E **O** must immediately respond to **P**'s moves.

For brevity, by "the E rules" we mean the structural rules together with the E rule. The standard name in the dialogue game literature for the structural rules presented here is "D". Evidently, when the so-called E rule is present, **O** is rather tightly constrained. A consequence of the E rule being present is that whenever **P** defends against an attack, **O** must immediately attack **P**'s move.

Theorem 2 (Felscher)**.** *There exists a winning strategy for ϕ iff there exists a winning strategy for ϕ that adheres to the E rules.*

(Note that our assumption that E is present is helpful for proof search considerations. It is another matter to philosophically justify the inclusion of E. We are relying on the fact that dialogue validity is the same with or without E, a result proved by Felscher.)

3 Implementation

Kuno is a Common Lisp (CL) program. One can run Kuno within a CL Read-Eval-Print loop (REPL), or from the commandline by first compiling the CL sources. At the moment, the only tested CL implementation is SBCL (Steel Bank Common Lisp), a major open-source CL implementation (`http://www.sbcl.org`).

The name "Kuno" is a tribute to Kuno Lorenz, one of the foundational figures of the field of dialogue games [8].

Kuno is based on a previous program that was designed to support a kind of interactive proof search, with a web-based frontend, is used [1].

4 Evaluation on ILTP

We consider first the propositional part of the ILTP (version 1.1.2, available at `http://www.cs.uni-potsdam.de/ti/iltp/formulae.html`). Table 2 contains the result of working with propositional problems in the LCL (devoted to Logic Calculi) and SYN (Syntactic) sections of the ILTP library. We developed strategies to a depth limit of 30 (that is, if a strategy ever exceeded depth 30, it was discarded from the search even if potentially it could be completed to a winning strategy). "Depth" means that Kuno did terminate, but the best it can say is that there is no strategy below the given depth limit. "Timeout" means that computation had to be halted by a time limit. "Crash" means that the system ran out of memory.

Problem	Intended SZS Status	Computed SZS Status agrees?	Reason
LCL181+1	Non-Theorem	-	depth
LCL230+1	Non-Theorem	-	depth
SYN001+1	Non-Theorem	-	depth
SYN007+1.014	Non-Theorem	-	crash
SYN040+1	Non-Theorem	-	timeout
SYN041+1	Theorem	+	
SYN044+1	Theorem	+	
SYN045+1	Theorem	+	
SYN046+1	Non-Theorem	-	depth
SYN047+1	Non-Theorem	-	timeout
SYN387+1	Non-Theorem	-	depth
SYN388+1	Non-Theorem	-	depth
SYN389+1	Non-Theorem	-	depth
SYN390+1	Theorem	+	
SYN391+1	Theorem	+	
SYN392+1	Non-Theorem	-	timeout
SYN393+1	Non-Theorem	-	timeout
SYN416+1	Non-Theorem	-	depth
SYN915+1	Theorem	+	
SYN916+1	Non-Theorem	+	
SYN977+1	Non-Theorem	-	depth
SYN978+1	Theorem	+	

Table 2: An evaluation of the E ruleset on several problems from the ILTP library (propositional part)

The initial experiment was useful not only for identifying bugs, but for gaining additional insight into dialogues as a decision procedure for propositional intuitionistic logic. In the cases where a result is indeed a theorem, we were not especially surprised, in view of previous experience with the architecture underlying Kuno, which was focused on working with known theorems. It was the case of non-theorems where we were more interested. We found that an important obstacle that prevents the program from terminating with useful information is the possibility of endless repetition or duplication by one of the players. In the case where we are dealing with a formula ϕ that is not intuitionistically valid, we find that there are two possibilities:

- **O** is able to repeat moves ad infinitum.
- The dialogue search tree (a complete development of all possible dialogues whatsoever starting from an initial formula) is finite, but it contains no winning strategy.

The second case is generally detected by Kuno; the first case is more interesting. Motivated by such concerns, we are led to consider an additional "no repetition" constraint.

No-Repeats Neither player can repeat a move.

We are at the moment unable to prove this constraint preserves completeness in the context of intuitionistic logic (though an analogous restriction preserves completeness in the case of classical logic [2]).

Happily, when working with the E ruleset with the No-Repeats constraint, we are able to solve all of the unsolved problems of Table 2 except SYN007+1.014, SYN047+1, and SYN393+1. Looking into more detail on the failure to come a decision on these three problems, we find two sources of difficulty:

- SYN007.014 is perhaps inherently quite difficult because it involves many atoms with many equivalences. Since $\leftrightarrow$ is treated as an abbreviation, the resulting formula is very large.
- **P**'s attacks are sometimes premature: for some formulas a solution can be found more quickly if an attack is delayed.

In light of the second observation, we considered an additional restriction on search:

Prefer-Defense If **P** can defend, then he does defend. (If multiple defenses are available, the choice is arbitrary.)

With this constraint, all the SYN problems of the ILTP library (except SYN007.014) are solvable, each in less than a minute (and most within several seconds).

Evaluating Kuno on properly first-order problems makes clear some difficulties with the naive depth-first approach currently implemented. Kuno works directly with the notion of winning strategy, rather than via tableaux or (hyper)sequents. Whether this approach can be carried through for genuinely difficult problems (e.g., those that are known to be classical theorems but which remain open in the ILTP) remains to be seen. We can report, though, that working at the first-order level reveals challenges not revealed with the propositional SYN problems. Namely, **Prefer-Defense** constraint cannot be rigidly applied; doing so leads to incompleteness. A simple example (a modification of SYJ001+1.002) illustrates the difficulty:

$$(\exists y \forall x (p(x) \wedge q(y))) \rightarrow (\forall x \exists y (p(x) \wedge q(y))).$$

When playing a dialogue game for this formula, **O** begins by asserting the existential in the antecedent. It is essential that **P** attacks this formula even though he could defend choose to defend against the initial attack by asserting the consequent. The difficulty, intuitively, is that if **P** begins by defending, **O** can pin him down by attacking a formula that, from a constructive point of view, is weaker than what was initially given to **P**. To solve this, one apparently has to relax the constraint imposed by **Prefer-Defense**.

5 Conclusion and future work

Kuno currently works in the equality-free fragment of intuitionistic logic. For many first-order theorem proving problems of interest, this is a rather serious restriction that ought to be remedied. At the moment, if a problem has equality in it at all, Kuno returns the SZS status `Inappropriate` to signal that it cannot deal with the problem. A fairly easy remedy for such a gap would be to be preprocess (using, e.g., TPTP4X) any problem involving equality so that appropriate equality axioms are present. Similarly, problems containing $\bot$ and $\top$ are rejected as inappropriate. One needs to extend the usual dialogue rules to account for these distinguished atoms.

By viewing dialogue games as an infrastructure for proof search, one quickly (and unsurprisingly) encounters the issue of redundancy. Various strategies can in general be developed starting from an initial formula, but they can differ from one another immaterially. The difficulty is, of course, to precisely specify what it means for two differences to be immaterial. By contrast with resolution calculi, the notion of redundancy seems to be underdeveloped within the dialogue game framework. Inspiration and ideas may come from other analytic search methods for intuitionistic logic, and perhaps even from game theory in general.

Moving beyond intuitionistic logic, Kuno could be extended to parallel dialogue games, which are known to be adequate for various intermediate logics [5, 6].

Intuitively, when the E rule is present, **O** is more constrained than when E is absent. One intuitively expects, then, that the E rules are favorable when the problem is to search for a winning strategy, that is, to determine intuitionistic validity. When E is absent (that is, when playing according to the D rules), **O** has more options (at least sometimes, in general), so **P** intuitively faces a greater risk of losing (**O** might be able to pursue a more hostile "line of reasoning" against **P**). In this spirit, one could conduct further experiments that focus on non-theorems, rather than trying to verify theoremhood. Such work could contribute to a better understanding of when dialogues go wrong [9].

References

[1] Jesse Alama and Sara L. Uckelman. Playing Lorenzen dialogue games on the web. In Andrei Voronkov, Geoff Sutcliffe, Matthias Baaz, and Christian G. Fermüller, editors, *LPAR short papers(Yogyakarta)*, volume 13 of *EPiC Series*, pages 7–12. EasyChair, 2010.

[2] Nicolas Clerbout. First-order dialogical games and tableaux. *Journal of Philosophical Logic*, pages 1–17, 2013.

[3] Walter Felscher. Dialogues, strategies, and intuitionistic provability. *Annals of Pure and Applied Logic*, 28:217–254, 1985.

[4] Walter Felscher. Dialogues as a foundation for intuitionistic logics. In D. M. Gabbay and F. Guenther, editors, *Handbook of Philosophical Logic*, volume III, pages 341–372. Reidel, 1986.

[5] Christian G. Fermüller. Parallel dialogue games and hypersequents for intermediate logics. In Marta Cialdea Mayer and Fiora Pirri, editors, *TABLEAUX*, volume 2796 of *Lecture Notes in Computer Science*, pages 48–64. Springer, 2003.

[6] Christian G. Fermüller and Agata Ciabattoni. From intuitionistic logic to Gödel-Dummett logic via parallel dialogue games. In *ISMVL*, pages 188–. IEEE Computer Society, 2003.

[7] Laurent Keiff. Dialogical logic. In Edward N. Zalta, editor, *The Stanford Encyclopedia of Philosophy*. Summer 2011 edition, 2011.

[8] Kuno Lorenz. Basic objectives of dialogical logic in historical perspective. *Synthese*, 127:255–263, 2001.

[9] Sara Negri. Proofs and countermodels in non-classical logics. *Logica Universalis*, 8(1):25–60, 2014.

[10] Shahid Rahman and Laurent Keiff. On how to be a dialogician. In Daniel Vanderveken, editor, *Logic, Though, and Action*, volume 2, pages 359–408. Springer, 2005.

[11] Thomas Raths, Jens Otten, and Christoph Kreitz. The ILTP library: Benchmarking automated theorem provers for intuitionistic logic. In Bernhard Beckert, editor, *Automated Reasoning with Analytic Tableaux and Related Methods*, volume 3702 of *Lecture Notes in Computer Science*, pages 333–337. Springer, 2005.

[12] Thomas Raths, Jens Otten, and Christoph Kreitz. The ILTP Problem Library for intuitionistic logic. *Journal of Automated Reasoning*, 8(1–3):261–271, 2007.

Theorem Proving for Logic with Partial Functions by Reduction to Kleene Logic [*]

Hans de Nivelle

Instytut Informatyki Uniwersytetu Wrocławskiego
Wybrzeże Joliot-Curie 15, 50-383, Wrocław, Poland
nivelle@ii.uni.wroc.pl

Abstract

We develop a theorem proving strategy for Partial Classical Logic (PCL) that is based on geometric logic. The strategy first translates PCL theories into sets of Kleene formulas. After that, the Kleene formulas are translated into 3-valued geometric logic. The resulting formulas can be refuted by an adaptation of geometric resolution.

1 Introduction

Partial Classical Logic (PCL) was introduced by the author in [6] with the aim of being able to express rich type systems and partial functions. In order to allow type systems of arbitrary complexity, one can use *relativization.* This means that **(1)** ordinary formulas are used for expressing type conditions: $+\colon N \times N \rightarrow N$ can be expressed by $\forall n_1, n_2\ N(n_1) \wedge N(n_2) \rightarrow N(n_1 + n_2)$. **(2)** Universal quantifiers use implication to include type conditions: $\forall n\colon N\ n \geq 0$ becomes $\forall n\ N(n) \rightarrow n \geq 0$, and **(3)** existential quantifiers use conjunction to include type conditions: $\exists n\colon N\ n \geq 0$ becomes $\exists n\ N(n) \wedge n \geq 0$. The advantage of relativization is the fact that it is flexible: Every property that can be expressed in logic can be used as type condition. The disadvantage is the fact that the type system looses its possibility to pre-check formulas, and to discard meaningless formulas. Using relativization, an ill-typed formula will become either true or false. As a theorem, the formula becomes either too hard to too easy to prove. As an assumption, it will be either too strong or too weak. The dangerous cases, which are an easy theorem, and a strong assumption, may go undetected for long time.

In order to model undefinedness, it has been proposed to use three-valued Kleene logic. In Kleene logic, the set of Booleans is extended with a third truth-value $\mathbf{u}$, which denotes *unknown.* Ill-typed formulas are treated as underspecified. The truth values are assumed to be ordered by $\mathbf{f} < \mathbf{u} < \mathbf{t}$. Disjunction $\oplus$ is defined by picking the rightmost truth-value from its arguments. Conjunction $\otimes$ picks the leftmost truth-value from its arguments. One has $\mathbf{u} \oplus \mathbf{t} = \mathbf{t}$, which reflects the fact that $\mathbf{u}$ can be interpreted as underspecified. If one of the arguments of a disjunction is $\mathbf{t}$, then the other argument does not matter. Kleene logic has been proposed for modeling partial functions in [9, 11, 10, 13] and [5].

In our view, Kleene logic does not capture the way in which types are intuitively used: Kleene logic is designed in such a way that one can derive as much as possible from underspecified formulas. The fact that $\oplus$ ignores $\mathbf{u}$ when the other argument is $\mathbf{t}$, (the same happens when one of the arguments of $\otimes$ is $\mathbf{f}$) reflects this. As a consequence, it is possible to assume an ill-typed formula. The formula will be assumed true, and it will be possible to derive consequences

[*]This work was supported by the Polish National Science Center (Narodowe Centrum Nauki) through grant number DEC-2011/03/B/ST6/00346

from it. If the formula is ill-typed, its type correctness is assumed together with the truth of the formula, so that an assumed formula may implicitly declare symbols.

PCL in contrast has *type strictness*: Nothing can be done with a formula when it is ill-typed at the point where it is used. In order to ensure this, contexts in PCL are ordered, and implication and conjunctions are both separated into two operators: the strict operators $\rightarrow$ and $\wedge$, and the lazy operators $[\,]$ and $\langle\,\rangle$. The strict operators are used in formulas in the usual way. A strict operator is well-typed only if both of its arguments are well-typed. The lazy operators are used in relativizations. A lazy operator is well-typed if its left argument is false, or its left argument is true, and its right argument is well-typed.

The goal of the current paper is to sketch a theorem proving procedure for PCL. The procedure is based on geometric logic. The current paper is a shortened version of [7], which contains a more detailed discussion of PCL, and all the proofs. We will define the syntax and semantics of PCL, give examples, and explain how PCL contexts can be decomposed into sets of sequents. (which can be alternatively viewed as sets of formulas.) In Section 2, we introduce the notion of widening, which is the key to obtaining theorem proving procedures for PCL. Widening is also important for understanding the relation between PCL, Kleene logic and classical logic, but we will not discuss this in this paper. It is discussed in [7]. In Section 3, we sketch a theorem proving procedure, based on [8], for three-valued, Kleene logic. In Section 4, we explain the last step of the transformation from sequents into geometric logic. We now define the syntax, and the semantics of PCL:

Definition 1.1. *We first define* terms*: If f is a function symbol with arity $n \geq 0$, and $t_1, \ldots, t_n$ are terms, then $f(t_1, \ldots, t_n)$ is a term as well. Using terms, we define the set of formulas as follows:*

- *$\bot, E$ and $\top$ are formulas.*
- *If t_1 and t_2 are terms, then $t_1 \approx t_2$ is a formula.*
- *If p is a predicate symbol with arity $n \geq 0$, and $t_1, \ldots, t_n$ are terms, then $p(t_1, \ldots, t_n)$ is a formula.*
- *If F is a formula, then $\neg F$ and $\# F$ are formulas.*
- *If F and G are formulas, then $F \vee G$, $F \wedge G$, $F \rightarrow G$, and $F \leftrightarrow G$ are formulas.*
- *If F and G are formulas, then $\langle F \rangle G$ and $[F] G$ are formulas.*
- *If F is a formula, and x is a variable, then $\forall x\ F$ and $\exists x\ F$ are formulas.*

Definition 1.2. *An* interpretation *is an object of form $I = (D, [\,])$. The set D is the domain of the interpretation. The function $[\,]$ interprets function symbols as functions from D^n to D and predicate symbols as functions from D^n to $\{\mathbf{f}, \mathbf{e}, \mathbf{t}\}$ in accordance with the arity of the symbol.*

Definition 1.3. *Let $I = (D, [\,])$ be an interpretation. Starting with $[\,]$, we define the interpretation function $I()$, that interprets all terms and formulas.*

- *If f is a function symbol of arity n, and $t_1, \ldots, t_n$ are terms, then $I(\ f(t_1, \ldots, t_n)\ ) = [f](I(t_1), \ldots, I(t_n))$.*
- *If p is a predicate symbol of arity n, and $t_1, \ldots, t_n$ are terms, then $I(\ p(t_1, \ldots, t_n)\ ) = [p](I(t_1), \ldots, I(t_n))$.*
- *$I(\bot) = \mathbf{f}$, $I(E) = \mathbf{e}$, and $I(\top) = \mathbf{t}$.*
- *For a unary propositional operator $\star$, the interpretation of $I(\star A)$ is defined from the value in the corresponding table below, using the value of $I(A)$ in the order $\mathbf{f}, \mathbf{e}, \mathbf{t}$:*

$\neg :$

t	**e**	**f**

$\# :$

t	**f**	**t**

- *For a binary propositional operator $\star$, the interpretation of $I(A \star B)$ is defined from the value in the corresponding table below. The row is determined by the value of $I(A)$ in the order $\mathbf{f}, \mathbf{e}, \mathbf{t}$, and the column is determined by the value of $I(B)$, also in the order $\mathbf{f}, \mathbf{e}, \mathbf{t}$.*

$\vee$:

f	**e**	**t**
e	**e**	**e**
t	**e**	**t**

$\wedge$:

f	**e**	**f**
e	**e**	**e**
f	**e**	**t**

$\leftrightarrow$:

t	**e**	**f**
e	**e**	**e**
f	**e**	**t**

$\rightarrow$:

t	**e**	**t**
e	**e**	**e**
f	**e**	**t**

$\langle\ \rangle$:

f	**f**	**f**
e	**e**	**e**
f	**e**	**t**

$[\,]$:

t	**t**	**t**
e	**e**	**e**
f	**e**	**t**

- *For a quantifier Q, the value of $I(Qx\ F)$ is obtained as follows: First define $R_F = \{I^x_d(F) \mid d \in D\}$. Then $I(Qx\ F)$ is obtained by selecting from R_F the most preferred value using the preference list for Q :*

$$\forall : \mathbf{e} > \mathbf{f} > \mathbf{t}, \qquad \exists : \mathbf{e} > \mathbf{t} > \mathbf{f}.$$

Theories are constructed in *contexts.* A context is a mixture of assumptions and theorems. Assumptions are assumptions in the usual sense, and type specifications. The order of formulas in a context is essential, because theorems must be provable from the formulas that occur before them, and types of functions and predicates have to be specified by assumptions before they are used.

Definition 1.4. *We call an object of form $\|\Gamma_1, \ldots, \Gamma_m\|$, in which Γ_j $(m \geq 0)$ are formulas, and in which some of the Γ_j are possibly marked with a ϑ, a* context*.*

Formulas that are marked with ϑ denote theorems, which means that they must be provable from the formulas occurring before them. Unmarked formulas are assumptions.

Example 1.5. $\|\ \#A,\ \#B,\ A,\ B,\ (A \wedge B)^\vartheta\ \|$ *is an example of a context. The formula $(A \wedge B)^\vartheta$ is marked as theorem, which is correct, because it is provable from the formulas A and B. The formulas A and B can be assumed because $\#A$ and $\#B$ occur before them.*

Definition 1.6. *Let $\|\Gamma\|$ be a context. We say that $\|\Gamma\|$ is* strongly valid *if in every interpretation I, for which there is an i, s.t. $I(\Gamma_i) \neq \mathbf{t}$, the first such i satisfies the following condition:*

- *Γ_i is not marked as theorem and $I(\Gamma_i) = \mathbf{f}$.*

Strong validity captures the intuition that theorems must be provable from the formulas before them, and assumptions must be well-defined assuming the formulas before them.

Example 1.7. *The context $\|\ A,\ B,\ (A \wedge B)^\vartheta\ \|$ is not strongly valid, because Definition 1.6 is not met by I with $I(A) = \mathbf{e}$, $I(B) = \mathbf{t}$. The context $\|\ \#A,\ A,\ B,\ (A \wedge B)^\vartheta\ \|$ is still not strongly valid, because it is possible that $I(B) = \mathbf{e}$, which would make B with $I(B) \neq \mathbf{f}$ the first formula with $I(B) \neq \mathbf{t}$.*

In order to obtain a context that is strongly valid, we also have to add $\#B$, so that we get $\|\ \#A,\ \#B,\ A,\ B,\ (A \wedge B)^\vartheta\ \|$.

Example 1.8. *The context $\|\ G(s),\ \forall x\ G(x) \rightarrow M(x),\ M^\vartheta(s)\ \|$ is not strongly valid, despite the fact that $G(s),\ \forall x\ G(x) \rightarrow M(x)$ implies $M(s)$.*

In order to make the context strongly valid, one has to make sure that predicates G and M (Greek and Mortal) are always well-defined: $\|\ \forall x\ \#G(x),\ \forall x\ \#M(x),\ G(s),\ \forall x\ G(x) \rightarrow$

$M(x),\ M^{\vartheta}(s)\ \|$. The predicates G and M are currently too general. In order to be more realistic, they can be restricted to be subpredicates of a predicate H (Human):

$$\left\| \begin{array}{lll} \forall x\ \#H(x), & \forall x\ H(x) \rightarrow \#G(x), & \forall x\ H(x) \rightarrow \#M(x) \\ \langle H(s) \rangle\ G(s), & \forall x\ [H(x)]\ G(x) \rightarrow M(x), & M^{\vartheta}(s) \end{array} \right\|$$

If the formula $\langle H(s) \rangle\ G(s)$ would be replaced by $H(s) \wedge G(s)$, the resulting context would be not strongly valid anymore, because one can have $I(H(s)) = \mathbf{f},\ I(G(s)) = \mathbf{e}$, while making all the formulas before it true.

Similarly, replacing $\forall x\ [H(x)]\ G(x) \rightarrow M(x)$ by $\forall x\ H(x) \wedge G(x) \rightarrow M(x)$ would make the context not strongly valid anymore, because there could exist a term t with $I(H(t)) = \mathbf{f},\ I(G(t)) = \mathbf{e}$.

We now show that a context can be decomposed into a set of sequents. An assumption results in one sequent, representing type correctness of the assumed formula. A theorem results in two sequents, representing type correctness and correctness of the theorem.

Definition 1.9. *A* sequent *is an object of form $S \vdash \bot$, in which S is a set of formulas.*

Definition 1.10. *Let $\|\Gamma\| = \|\Gamma_1, \ldots, \Gamma_m\|$ be a context. The* sequent expansion Seq($\|\Gamma\|$) *of $\|\Gamma\|$ is defined as $\bigcup_{1 \leq i \leq m} E(\ \|\Gamma\|,\ i) \cup \bigcup_{1 \leq i \leq m} E_{\vartheta}(\ \|\Gamma\|,\ i)$, where*

- *$E(\ \|\Gamma\|,\ i) = \{\ \{\Gamma_1, \ldots, \Gamma_{i-1}, \neg(\#\Gamma_i)\} \vdash \bot\ \}$.*
- *$E_{\vartheta}(\ \|\Gamma\|,\ i) = \{\ \{\Gamma_1, \ldots, \Gamma_{i-1}, \neg\Gamma_i\} \vdash \bot\ \}$ if Γ_i is marked as theorem, and $\{\ \}$ otherwise.*

Definition 1.11. *We say that a set of formulas S is* strongly unsatisfiable *if for every interpretation I, there is a formula $A \in S$, s.t. $I(A) = \mathbf{f}$.*

We say that a set of sequents $S_1 \vdash \bot, \ldots, S_n \vdash \bot$ strongly represents *a property P iff*

1. *P is true implies that all of the S_i are strongly unsatisfiable.*
2. *P is false implies that (at least) one of the S_i is satisfiable.*

The advantage of the use of strong representation is that it makes it possible to ignore error values during proof search. Either, it is possible to make all formulas in all sequents true, or there always is a false formula in some sequent.

Theorem 1.12. *For every context Γ, the sequent expansion* Seq($\|\Gamma\|$) *strongly represents the property that $\|\Gamma\|$ is strongly valid.*

For the proof, we refer to [7].

Example 1.13. *Suppose we want to prove that the context of Example 1.8 is strongly valid. First define:*

$$\begin{array}{lll lll} A_0 & = & \forall x\ \#H(x) & B_0 & = & \neg\#\ \forall x\ \#H(x) \\ A_1 & = & \forall x\ H(x) \rightarrow \#G(x) & B_1 & = & \neg\#\ \forall x\ H(x) \rightarrow \#G(x) \\ A_2 & = & \forall x\ H(x) \rightarrow \#M(x) & B_2 & = & \neg\#\ \forall x\ H(x) \rightarrow \#M(x) \\ A_3 & = & \langle H(s) \rangle\ G(s) & B_3 & = & \neg\#\ \langle H(s) \rangle\ G(s) \\ A_4 & = & \forall x\ [H(x)]\ G(x) \rightarrow M(x) & B_4 & = & \neg\#\ \forall x\ [H(x)]\ G(x) \rightarrow M(x) \\ G & = & \neg M(s) & B_5 & = & \neg\#M(s) \end{array}$$

The sequent expansion consists of the following sequents:

$$\begin{array}{ll} B_0 \vdash \bot & A_0, A_1, A_2, A_3, B_4 \vdash \bot \\ A_0, B_1 \vdash \bot & A_0, A_1, A_2, A_3, A_4, B_5 \vdash \bot \\ A_0, A_1, B_2 \vdash \bot & A_0, A_1, A_2, A_3, A_4, G \vdash \bot \\ A_0, A_1, A_2, B_3 \vdash \bot & \end{array}$$

The last two sequents in the second column originate from $M^{\vartheta}(s)$. It created two sequents because it is a theorem.

2 Transformation to Kleene Logic

Decomposition Seq can be used to decompose a context into a set of sequents. By Theorem 1.12, this set of sequents strongly represents strong validity of the original context. In this section, we transform the resulting set of sequents into a set of sequents in Kleene logic, that still strongly represents strong validity the original context.

The notion of strong representation contains a gap between the property being true and the property being false. When the property being represented is true, every sequent always contains a false formula. When the property is false, there is a sequent that can contain only true formulas in some interpretation. This means that strong representation says nothing about **e**, and there is no need to preserve it during transformations. As a consequence, transformations do not need to preserve all truth-values. The following property is sufficient:

Definition 2.1. *We define the* widening relation $\preceq$ *as follows:* $A \preceq B$ *if in every interpretation* I*, we have* $I(A) = \mathbf{f} \Rightarrow I(B) = \mathbf{f}$*, and* $I(A) = \mathbf{t} \Rightarrow I(B) = \mathbf{t}$.

We write $A \equiv B$ *if* $A \preceq B$ *and* $B \preceq A$*. It can be easily checked that* $A \equiv B$ *iff* $I(A) = I(B)$ *in every interpretation* I.

Theorem 2.2. *Let* $\{S_1 \vdash \bot,\ \ldots,\ S_n \vdash \bot\}$ *be a set of sequents that strongly represents a property* P*. Let* A *and* B *be two formulas with* $A \preceq B$.

Let $\{S_1' \vdash \bot,\ \ldots,\ S_n' \vdash \bot\}$ *be obtained from* $\{S_1 \vdash \bot,\ \ldots,\ S_n \vdash \bot\}$ *by possibly replacing some occurrences of* A *by* B.

Then $\{S_1' \vdash \bot,\ \ldots,\ S_n' \vdash \bot\}$ *strongly represents property* P *as well.*

The proof is given in [7]. Since we have now established that one can use $\preceq$ in transformations, there is no need anymore to distinguish between PCL operators and Kleene operators. As a consequence, PCL operators can be replaced by their Kleene counterparts. We will use the notation $\otimes$ for Kleene conjunction, $\oplus$ for Kleene disjunction, Π for Kleene universal quantification, and Σ for Kleene existential quantification.

Definition 2.3. *We extend the set of formulas, defined in Definition 1.1, with two binary operators and two quantifiers as follows: If* F *and* G *are formulas, then* $F \otimes G$ *and* $F \oplus G$ *are formulas as well. If* F *is a formula and* x *is a variable, then* $\Pi x\ F$ *and* $\Sigma x\ F$ *are formulas. We extend the interpretation of formulas, defined in Definition 1.3, as follows:*

- *The semantics of the binary operators* $\otimes$ *and* $\oplus$ *is defined by the following truth tables:*

$\otimes$:

f	**f**	**f**
f	**e**	**e**
f	**e**	**t**

$\oplus$:

f	**e**	**t**
e	**e**	**t**
t	**t**	**t**

- *The semantics of the quantifiers* Π *and* Σ *is defined by the following preferences:*

$$\Pi : \mathbf{f} > \mathbf{e} > \mathbf{t}, \qquad \Sigma : \mathbf{t} > \mathbf{e} > \mathbf{f}.$$

The first row of Figure 1 lists the rules that replace PCL operators by their corresponding Kleene operators. The rules in the second row can be used for pushing the #-operator inward. Operator # is intended to be used for declaring symbols, as in $\forall mn\ N(m) \wedge N(n) \rightarrow \#(m \leq n)$.

Figure 1: Kleening Rules

Rules for replacing PCL operators (left) and NNF rules (right):

$$
\begin{array}{lcl@{\qquad\qquad\qquad}lcl}
A \wedge B & \preceq & A \otimes B & \neg\bot & \equiv & \top \\
A \vee B & \preceq & A \oplus B & \neg E & \equiv & E \\
A \rightarrow B & \preceq & \neg A \oplus B & \neg\top & \equiv & \bot \\
A \leftrightarrow B & \preceq & (\neg A \vee B) \otimes (A \vee \neg B) & \neg\neg A & \equiv & A \\
[A]B & \preceq & \neg A \oplus B & \neg(A \oplus B) & \equiv & \neg A \otimes \neg B \\
\langle A \rangle B & \preceq & A \otimes B & \neg(A \otimes B) & \equiv & \neg A \oplus \neg B \\
\forall x\ F[x] & \preceq & \Pi x\ F[x] & \neg(\ \Pi x\ F[x]\ ) & \equiv & \Sigma x\ \neg F[x] \\
\exists x\ F[x] & \preceq & \Sigma x\ F[x] & \neg(\ \Sigma x\ F[x]\ ) & \equiv & \Pi x\ \neg F[x]
\end{array}
$$

Rules for #:

$$
\begin{array}{lcl@{\qquad\qquad}lcl}
\#\top & \equiv & \top & \#(A \wedge B) & \equiv & \#A\ \otimes\ \#B \\
\#E & \equiv & \bot & \#(A \vee B) & \equiv & \#A\ \otimes\ \#B \\
\#\bot & \equiv & \top & \#(A \rightarrow B) & \equiv & \#A\ \otimes\ \#B \\
\#(\neg A) & \equiv & \#A & \#(A \leftrightarrow B) & \equiv & \#A\ \otimes\ \#B \\
\#(\#A) & \equiv & \top & \#(\ [A]B\ ) & \equiv & \#A\ \otimes\ (\ \neg A \vee \#B\ ) \\
\#(\ \forall x\ F[x]\ ) & \equiv & \Pi x\ \#F[x] & \#(\ \langle A \rangle B\ ) & \equiv & \#A\ \otimes\ (\ \neg A \vee \#B\ ) \\
\#(\ \exists x\ F[x]\ ) & \equiv & \Pi x\ \#F[x] & \#(t_1 \approx t_2) & \equiv & \top
\end{array}
$$

Because of this, we expect application of # on compound formulas to be rare, so that the rules in the second column probably will not be used often. We included them for completeness.

Theorem 2.4 states that the replacements of Figure 1 agree with $\preceq$ when they are made on top level. Theorem 2.5 states that $\preceq$ can be lifted into formula contexts that do not contain #, so that replacements can be made on subformulas as well.

Theorem 2.4. *For every rule of Figure 1, that is written as $A \preceq B$, it is indeed the case that $A \preceq B$. For every rule that is written as $A \equiv B$, it is indeed the case that $A \equiv B$.*

The propositional cases can be checked by case analysis.

Theorem 2.5. *All of the logical operators of PCL and the Kleene operators, with the exception of #, are monotone relative to $\preceq$. More precisely:*

- *If $A \preceq B$, then $\neg A \preceq \neg B$.*
- *If $A_1 \preceq B_1$ and $A_2 \preceq B_2$, then*

$$
\begin{array}{lll}
A_1 \wedge A_2 \preceq B_1 \wedge B_2, & A_1 \vee A_2 \preceq B_1 \vee B_2, & A_1 \rightarrow A_2 \preceq B_1 \rightarrow B_2, \\
A_1 \leftrightarrow A_2 \preceq B_1 \leftrightarrow B_2, & [A_1]A_2 \preceq [B_1]B_2, & \langle A_1 \rangle A_2 \preceq \langle B_1 \rangle B_2.
\end{array}
$$

- *If $P[x] \preceq Q[x]$, then*

$$
\begin{array}{ll}
\forall x\ P[x] \preceq \forall x\ Q[x], & \exists x\ P[x] \preceq \exists x\ Q[x], \\
\Pi x\ P[x] \preceq \Pi x\ Q[x], & \Sigma x\ P[x] \preceq \Sigma x\ Q[x].
\end{array}
$$

It can be easily seen that # is not monotone: For example, one has $E \preceq \top$, but not $\#E \preceq \#\top$.

At this point, we have established that the rules in Figure 1 can be used to rewrite a formula into a normal form. The restriction to contexts not containing # is unproblematic, because # can be pushed inwards down to the atoms.

Definition 2.6. *For a formula A in PCL (possibly mixed with Kleene operators), we define the* Kleening *of A, written as* $\mathrm{Kl}(A)$, *as the result of normalizing A using the rules for replacing PCL operators, and the rules for # in Figure 1.*

It is possible to define $\mathrm{Kl}(A)$ by a single, recursive definition. Such definition is given in [7].

Theorem 2.7. *For every formula A, we have* $A \preceq \mathrm{Kl}(A)$.

Theorem 2.7 follows from Theorem 2.4 and Theorem 2.5.

Definition 2.8. *We say that a formula A is* in Kleene logic *if it contains only logical operators from* $\bot, \top, E, \neg, \#, \otimes, \oplus, \Pi, \Sigma$ *(and* $\approx$*), and the operator # is applied only on non-equality atoms.*

It is easily checked that $\mathrm{Kl}(A)$ is always in Kleene logic. The rules for $\mathrm{Kl}(\ A \leftrightarrow B\ )$, $\mathrm{Kl}^{\#}(\ \langle A \rangle B\ )$, and $\mathrm{Kl}^{\#}(\ [A]B\ )$ may cause an exponential increase in size of the formula. This problem can be avoided by using suitable subformula replacement rules.

During the rest of the tranformation, it is convenient to push negation inwards as far as possible. This has the advantage that the polarity of non-trivial subformulas is always positive, which will simplify the remaining transformations.

Definition 2.9. *Let A be a Kleene formula. We say that A is* in negation normal form (NNF) *if negation is applied only on atoms and on formulas of form* $\#p(t_1, \ldots, t_n)$.

Definition 2.10. *We define the* negation normal form *of A, for which we write* $\mathrm{NNF}(A)$, *as the normal form that is obtained when A is normalized using the NNF rules in Figure 1.*

As with Kl, the negation normal form can be defined in a single recursive definition.

Example 2.11. *We apply Kleening and compute the negation normal form of the formulas in Example 1.13.*

$$\begin{array}{lll@{\qquad}lll}
A_0 & \preceq & \Pi x\ \#H(x) & B_0 & \preceq & \Sigma x\ \bot \\
A_1 & \preceq & \Pi x\ (\neg H(x) \oplus \#G(x)) & B_1 & \preceq & \Sigma x\ (\neg\#H(x) \oplus \bot) \\
A_2 & \preceq & \Pi x\ (\neg H(x) \oplus \#M(x)) & B_2 & \preceq & \Sigma x\ (\neg\#H(x) \oplus \bot) \\
A_3 & \preceq & H(s) \otimes G(s) & B_3 & \preceq & \neg\#H(s) \oplus (H(s) \otimes \neg\#G(s)) \\
A_4 & \preceq & \Pi x\ (\neg H(s) \oplus & B_4 & \preceq & \Sigma x\ (\neg\#H(x) \oplus (H(x) \otimes \\
 & & \quad \neg G(s) \oplus M(s)) & & & \quad (\neg\#G(x) \oplus \neg\#M(x)))) \\
G & \preceq & \neg M(s) & B_5 & \preceq & \neg\#M(s)
\end{array}$$

The Kleening transformation forgets type information. For example, the PCL formulas $\forall x\ [H(x)]\ G(x) \rightarrow M(x)$, $\forall x\ [H(x) \wedge G(x)]\ M(x)$, $\forall x\ H(x) \wedge G(x) \rightarrow M(x)$, $\forall x\ H(x) \rightarrow [G(x)]\ M(x)$, $\forall x\ [H(x)]\ [G(x)]\ M(x)$ all have the same Kleening $\Pi x\ \neg H(x) \oplus \neg G(x) \oplus M(x)$.

The formulas were different in PCL, but Kleening has widened them into the same formula. The formulas F still have different meanings in PCL, because the Kleenings of the formulas $\neg\#F$ differ, so that they will be typechecked in different ways.

The fact that Kleening removes type information can be reformulated as: Once a formula has been type checked, its type information can be forgotten. Alternatively, one can say: A typechecked formula can be considered equivalent to its Kleening. In the full paper [7], it is argued that in most cases, the Kleene translation of a theorem is equivalent to its classical translation, so that one obtains: A typechecked formula can be considered equivalent to its relativization in classical logic. Since people tend to think of their favorite formulas, which are always well-typed, people tend to believe that all type information can be expressed by relativization. In practice, many formulas in applications are not particularly liked by anyone, and for those, type checking will be very useful. The next step in the transformation is called *radicalization*:

Definition 2.12. *For each predicate symbol p with arity n, we define the following abbreviations:*

$$\begin{cases} p_{\emptyset}(t_1,\ldots,t_n) & := \bot \\ p_{\mathbf{f}}(t_1,\ldots,t_n) & := \#p(t_1,\ldots,t_n) \otimes \neg p(t_1,\ldots,t_n) \\ p_{\mathbf{e}}(t_1,\ldots,t_n) & := \neg\#p(t_1,\ldots,t_n) \\ p_{\mathbf{t}}(t_1,\ldots,t_n) & := \#p(t_1,\ldots,t_n) \otimes p(t_1,\ldots,t_n) \\ p_{\mathbf{f},\mathbf{e}}(t_1,\ldots,t_n) & := \neg\#p(t_1,\ldots,t_n) \oplus \neg p(t_1,\ldots,t_n) \\ p_{\mathbf{e},\mathbf{t}}(t_1,\ldots,t_n) & := \neg\#p(t_1,\ldots,t_n) \oplus p(t_1,\ldots,t_n) \\ p_{\mathbf{f},\mathbf{t}}(t_1,\ldots,t_n) & := \#p(t_1,\ldots,t_n) \\ p_{\mathbf{f},\mathbf{e},\mathbf{t}}(t_1,\ldots,t_n) & := \top \end{cases}$$

Theorem 2.13. *For every propositional atom $p_\lambda(t_1,\ldots,t_n)$, for every interpretation I, we have $I(p_\lambda(t_1,\ldots,t_n)) \in \{\mathbf{f},\mathbf{t}\}$.*

Definition 2.14. *Let A be a Kleene formula in NNF. We recursively define* the radicalization Rad(A) *of A as follows:*

$$\begin{array}{lcl} \mathrm{Rad}(\ p(t_1,\ldots,t_n)\ ) & = & p_{\mathbf{t}}(t_1,\ldots,t_n) \\ \mathrm{Rad}(\neg p(t_1,\ldots,t_n)\ ) & = & p_{\mathbf{f}}(t_1,\ldots,t_n) \\ \mathrm{Rad}(\#p(t_1,\ldots,t_n)\ ) & = & p_{\mathbf{f},\mathbf{t}}(t_1,\ldots,t_n) \\ \mathrm{Rad}(\#p(t_1,\ldots,t_n)\ ) & = & p_{\mathbf{e}}(t_1,\ldots,t_n) \\ & & \\ \mathrm{Rad}(\ t_1 \approx t_2\ ) & = & t_1 \approx t_2 \\ & & \\ \mathrm{Rad}(\top) & = & \top \\ \mathrm{Rad}(E) & = & \bot \\ \mathrm{Rad}(\bot) & = & \bot \\ \mathrm{Rad}(A \otimes B) & = & \mathrm{Rad}(A) \otimes \mathrm{Rad}(B) \\ \mathrm{Rad}(A \oplus B) & = & \mathrm{Rad}(A) \oplus \mathrm{Rad}(B) \\ & & \\ \mathrm{Rad}(\Pi x\ P[x]) & = & \Pi x\ \mathrm{Rad}(P[x]) \\ \mathrm{Rad}(\Sigma x\ P[x]) & = & \Sigma x\ \mathrm{Rad}(P[x]) \end{array}$$

Radicalization is called ‘radicalization’ because in the resulting formula, every non-atomic subformula always has a definite truth value. This has a surprising consequence, namely there is no need to use Kleene operators anymore. In the resulting formula, each Kleene operator can be replaced by its corresponding classical (or PCL) operator without changing the truth value of the formula. This implies that in the last four rules of the definition of Rad, one could have used $\wedge, \vee, \forall, \exists$ instead of $\otimes, \oplus, \Pi, \Sigma$.

Theorem 2.15. *For every Kleene formula A in NNF, we have $A \preceq$ Rad(A).*

Proof. First check (by case analysis) that

$$\begin{array}{lll} p(t_1,\ldots,t_n) & \preceq & p_{\mathbf{t}}(t_1,\ldots,t_n), \\ \neg p(t_1,\ldots,t_n) & \preceq & p_{\mathbf{f}}(t_1,\ldots,t_n), \\ \#p(t_1,\ldots,t_n) & \preceq & p_{\mathbf{f},\mathbf{t}}(t_1,\ldots,t_n), \\ \neg\#p(t_1,\ldots,t_n) & \preceq & p_{\mathbf{e}}(t_1,\ldots,t_n), \\ E & \preceq & \bot. \end{array}$$

After that, apply Theorem 2.5. □

We give an example of radicalization:

Example 2.16. *Radicalizing the formulas in Example 2.11 gives:*

$$\begin{array}{lll} A_0 & = & \Pi x\ H_{\mathbf{f},\mathbf{t}}(x) \\ A_1 & = & \Pi x\ (H_{\mathbf{f}}(x) \oplus G_{\mathbf{f},\mathbf{t}}(x)) \\ A_2 & = & \Pi x\ (H_{\mathbf{f}}(x) \oplus M_{\mathbf{f},\mathbf{t}}(x)) \\ A_3 & = & H_{\mathbf{t}}(s) \otimes G_{\mathbf{t}}(s) \\ A_4 & = & \Pi x\ (H_{\mathbf{f}}(s) \oplus G_{\mathbf{f}}(s) \oplus M_{\mathbf{t}}(s)) \\ G & = & M_{\mathbf{f}}(s) \end{array} \qquad \begin{array}{lll} B_0 & = & \Sigma x\ \bot \\ B_1 & = & \Sigma x\ (H_{\mathbf{e}}(x) \oplus \bot) \\ B_2 & = & \Sigma x\ (H_{\mathbf{e}}(x) \oplus \bot) \\ B_3 & = & H_{\mathbf{e}}(s) \oplus (H_{\mathbf{t}}(s) \otimes G_{\mathbf{e}}(s)) \\ B_4 & = & \Sigma x\ (H_{\mathbf{e}}(x) \oplus (H_{\mathbf{t}}(x) \otimes \\ & & \quad (G_{\mathbf{e}}(x) \oplus M_{\mathbf{e}}(x)))) \\ B_5 & = & M_{\mathbf{e}}(s) \end{array}$$

The resulting sequents still strongly represent strong validity of the context in Example 1.8.

At this point, we are close to classical logic, so that it would be possible to define a resolution procedure in the standard way. One can Skolemize the radicalized sequents, factor them into clausal normal form, and apply resolution between atoms $p_\lambda(t_1,\ldots,t_n)$ and $p_\mu(u_1,\ldots,u_n)$, if $\lambda \cap \mu = \emptyset$, and the t_i are unifiable with u_i. Since we are interested in using geometric logic, for reasons that were explained in [8], we will develop a theorem proving procedure for geometric logic in the remaining section of this paper.

3 Three-Valued Geometric Logic

Geometric logic for theorem proving was introduced in [3]. The proof search algorithm for geometric logic is closely related to model generation (see [12]) or to proof search based on hyper tableaux (see [2]). In [8], we introduced a variant of geometric logic, in which function symbols are replaced by predicates, which is able to deal directly with equality, and which learns a lemma, whenever it closes a branch in the search tree. The search algorithm is similar to the algorithm in [4]. Whenever the algorithm encounters an existential quantifier, it first tries all existing domain elements as possible witness. If this does not succeed, it extends the model with a new domain element. The difference with [4] is that our method replaces function symbols by predicate symbols, and that it relies on lemma learning during search. In this paper, we describe the search procedure without learning, because the learning rules for three-valued, geometric resolution are quite involved. They are derived from the learning rules in [8]. A detailed description can be found in [7]. We first describe the format of geometric (Kleene) formulas:

Definition 3.1. *A* geometric atom *is an atom of one of the following three forms:*

1. *An atom $p_\lambda(x_1,\ldots,x_n)$, where $x_1,\ldots,x_n$ are variables, not necessarily distinct, and $\lambda = \{\mathbf{f}\},\{\mathbf{e}\},\{\mathbf{t}\},\{\mathbf{f},\mathbf{e}\}$ or $\{\mathbf{f},\mathbf{t}\}$.*

2. *An equality atom $x_1 \approx x_2$, with x_1, x_2 distinct variables.*
3. *An existentially quantified atom $\Sigma y\ p_\lambda(x_1, \ldots, x_n)$, where $x_1, \ldots, x_n$ are variables, not necessarily distinct, and $\lambda = \{\mathbf{e}\}$ or $\{\mathbf{t}\}$. There must be at least one occurrence of y among the x_i.*

   We will usually write $\Sigma y\ p_\lambda(x_1, \ldots, x_n, y)$, even when y does not have to be at the last position, and there can be more than one occurrence of y.

A geometric formula *is a formula of form $\Pi \overline{x}\ A_1 \oplus \cdots \oplus A_p$, where each A_i is a geometric atom with all its free variables among $\overline{x}$. It is not required that A_i contains all variables of $\overline{x}$.*

The term 'geometric atom' is slightly misleading because an object of form $\exists y\ p_\lambda(x_1, \ldots, x_n, y)$ is not an atom. We think that it is sufficiently close to an atom, so that it can still be called 'atom'. The labels in the formulas play a role similar to the signs in [14]. The search procedure is an adaptation of the procedure of [8] for classical logic. In this paper, we only describe the simplified procedure without learning.

Definition 3.2. *We assume a countably infinite set of domain elements $\mathcal{E}$. A* ground substitution Θ *is a partial function from variables to $\mathcal{E}$. If A is a geometric atom, all whose free variables are defined in Θ, then $A\Theta$ is the result of replacing each free variable x by its corresponding $x\Theta$. We call the atoms that can be obtained in this way* ground atoms.

As with 'geometric atom', the term 'ground atom' is not completely correct, because 'ground atoms' are not always ground, and also not always atoms, but we think they are close enough.

Definition 3.3. *An* interpretation *is a pair (E, M) in which $E \subseteq \mathcal{E}$ is a set of elements, and M is a set of ground atoms of form $p_\lambda(e_1, \ldots, e_n)$, s.t. $\lambda = \mathbf{e}$ or $\lambda = \mathbf{t}$, and all $e_i \in E$. It is not allowed that M contains a conflicting pair of form $p_{\mathbf{e}}(e_1, \ldots, e_n),\ p_{\mathbf{t}}(e_1, \ldots, e_n)$.*

An interpretation stores the ground atoms that have truth assignments different from the default value $\mathbf{f}$. Using this semantics, one can define when a ground atom is true in an interpretation. An atom of form $p_{\mathbf{t}}(e)$ is true in M if $p_{\mathbf{t}}(e)$ occurs in M. An atom of form $p_{\mathbf{f}}(e)$ is true in M if both of $p_{\mathbf{e}}(e)$ and $p_{\mathbf{t}}(e)$ do not occur in M. An atom of form $p_{\mathbf{f},\mathbf{t}}(e)$ is true in M if $p_{\mathbf{e}}(e)$ does not occur in M.

When an atom is not true in an interpretation, there are two possibilities: Either it can be made true by extending M, or it can be made true only by making M smaller. In the second case, we call the atom *in conflict* with M.

Definition 3.4. *Let (E, M) be an interpretation. Let A be a ground atom with all its elements in E. We say that A is* false *in (E, M) if one of the following holds:*

1. *A has form $e_1 \approx e_2$ and $e_1 \neq e_2$.*
2. *A has form $p_{\{\lambda\}}(e_1, \ldots, e_n)$, $\lambda \neq \mathbf{f}$, and M does not contain $p_\lambda(e_1, \ldots, e_n)$.*
3. *A has form $p_\lambda(e_1, \ldots, e_n)$, $\mathbf{f} \in \lambda$, and M contains an atom of form $p_\mu(e_1, \ldots, e_n)$ with $\mu \notin \lambda$.*
4. *A has form $\Sigma y\ p_\lambda(e_1, \ldots, e_n, y)$ with $\lambda = \{\mathbf{e}\}, \{\mathbf{t}\}$, and there is no $e \in E$, s.t. $p_\lambda(e_1, \ldots, e_n, e)$ occurs in M.*

We say that A is true in (E, M) if A is not false in (E, M). We say that A is in conflict *with (E, M) if one of the following holds:*

1. *A has form $e_1 \approx e_2$ and $e_1 \neq e_2$.*
2. *A has form $p_\lambda(e_1, \ldots, e_n)$, and M contains an atom of form $p_\mu(e_1, \ldots, e_n)$ with $\mu \notin \lambda$.*

It can be seen from Definition 3.4 that an atom of form $\Sigma y\ p_\lambda(e_1, \ldots, e_n, y)$ is never in conflict with an interpretation. This is because it is always possible to add a new element e to E, and add $p_\lambda(e_1, \ldots, e_n, e)$ to M. It is easily shown that a conflict atom is always false:

Lemma 3.5. *Let (E, M) be an interpretation. Let A be a ground atom all whose elements occur in E. If A is in conflict with (E, M), then A is false in (E, M).*

It is also easy to show that the only way of repairing a conflict is by backtracking:

Lemma 3.6. *Let (E, M) be an interpretation. Let A be a ground atom all whose elements occur in E. If A is in conflict with (E, M), then A is also in conflict with every interpretation (E', M'), s.t. $E \subseteq E'$ and $M \subseteq M'$.*

If an atom is false in an interpretation, but not in conflict, then it can be made true by extending the interpretation as follows:

Definition 3.7. *Let (E, M) be an interpretation. Let A be a ground atom with all its elements in E. Assume that A is false in (E, M), but not in conflict with (E, M). We say that A* extends *(E, M) into (E', M') if either*

1. *A has form $p_\lambda(e_1, \ldots, e_n)$ with $\lambda = \{\mathbf{e}\}, \{\mathbf{t}\}$, and $(E', M') = (E, M \cup \{\ p_\lambda(e_1, \ldots, e_n)\ \})$.*
2. *A has form $\Sigma y\ p_\lambda(e_1, \ldots, e_n, y)$ with $\lambda \in \{\mathbf{e}\}, \{\mathbf{t}\}$, and there exists an $e \in E$, s.t. $(E', M') = (\ E,\ M \cup \{\ p_\lambda(e_1, \ldots, e_n, e)\ \})$.*
3. *A has form $\Sigma y\ p_\lambda(e_1, \ldots, e_n, y)$, and there exists an $\hat{e} \notin E$, s.t. $(E', M') = (\ E \cup \{\hat{e}\},\ M \cup \{\ p_\lambda(e_1, \ldots, e_n, \hat{e})\ \})$.*

We write $(E, M) \Rightarrow_A (E', M')$, if A extends (E, M) into (E', M'),

In case A is existential, the relation $\Rightarrow_A$ is non-deterministic, because one can choose either to use an existing $e \in E$ as witness, or to create a new $\hat{e} \notin E$. In the latter case, the actual $\hat{e}$ chosen does not matter. We will always assume that there is a fixed way of obtaining a new $\hat{e} \notin E$, and that only one $\hat{e}$ will be considered.

The following lemmas states that it is always possible to extend, that atoms made true by extension will remain true, and that extension is the smallest modification that makes an atom true.

Lemma 3.8. *Let (E, M) be an interpretation. Let A be a ground atom all whose elements are in E. Assume that A is false in (E, M), but not in conflict with (E, M). Then the following hold:*

1. *There exists an interpretation (E', M'), s.t. $(E, M) \Rightarrow_A (E', M')$.*
2. *A is true in every interpretation (E', M') for which $(E, M) \Rightarrow_A (E', M')$, and in every interpretation (E'', M''), s.t. $E' \subseteq E''$ and $M' \subseteq M''$.*
3. *For every interpretation (E'', M'') with $E \subseteq E''$ and $M \subseteq M''$ in which A is true, there exists an interpretation (E', M'), s.t. $(E, M) \Rightarrow_A (E', M')$ and $E' \subseteq E''$, $M' \subseteq M''$.*

Note that part 3 implies that false atoms that are not in conflict, cannot be made true by backtracking.

Definition 3.9. *Let $F = \Pi\overline{x}\ A_1 \oplus \cdots \oplus A_p$ be a geometric formula. Let Θ be a ground substitution that is defined for all variables in $\overline{x}$. We say that F* is false *in (E, M) with Θ, if all instantiated atoms $A_i\Theta$ are false in (E, M).*

We say that F conflicts *(E, M) with Θ if each of its instantiated atoms $A_i\Theta$ is in conflict with (E, M). In that case, we call F* a conflict formula *of (E, M).*

Figure 2: Propositional Geometric Formulas

(1)	$A_{\mathbf{f},\mathbf{t}}$
(2)	$A_{\mathbf{f}} \oplus B_{\mathbf{f},\mathbf{t}}$
(3)	$A_{\mathbf{f}} \oplus C_{\mathbf{f},\mathbf{t}}$
(4)	$A_{\mathbf{e}} \oplus A_{\mathbf{t}}$
(5)	$A_{\mathbf{e}} \oplus B_{\mathbf{e}}$

Using extension, we define the search algorithm. It tries to extend an interpretation (E, M) into an interpretation (E', M') that makes all formulas true with every ground substitution.

At each stage, the algorithm looks for a formula F and a substitution Θ, s.t. $F\Theta$ is false in the current interpretation. If no F and Θ are found, then (E, M) is a satisfying interpretation. If F is a conflict formula, then the algorithm fails, and it backtracks. If F is not a conflict formula, then the algorithm backtracks through all possible extensions (E', M'), based on an atom A of F that is false, but not in conflict with (E, M).

Definition 3.10. *Algorithm $S_t(\mathcal{G}, E, M)$ is called with a set of geometric formulas $\mathcal{G}$ and an interpretation (E, M). It tries to extend the interpretation into an interpretation (E', M'), that makes all geometric formulas in $\mathcal{G}$ true. If no such interpretation exists, it returns $\perp$.*

If such an extension exists, it either returns an interpretation (E', M') with $E \subseteq E'$, $M \subseteq M'$, in which all $\mathcal{G}$ are true, or it does not terminate. $S_t(\mathcal{G}, E, M)$ is defined by case analysis:

MODEL: *If for all formulas $F \in \mathcal{G}$ all ground instances $F\Theta$ that use only elements in E, are true in (E, M), then $S_t(\mathcal{G}, E, M)$ returns (E, M).*

SELECT: *Otherwise, $\mathcal{G}$ contains at least one formula F, for which there exists a substitution Θ, s.t. $F\Theta$ is false in (E, M). Let $A_1, \ldots, A_m$ be the atoms in $F\Theta$ that are not in conflict with (E, M). Select an F and a Θ, for which m is minimal.*

FAIL: *If $m = 0$, then F is a conflict formula, and $S_t(\mathcal{G}, E, M)$ returns $\perp$.*

EXTEND: *If $m > 0$, then for every A_i, for every (E', M') with $(E, M) \Rightarrow_{A_i} (E', M')$, do the following:*

- *Assign $r = S_t(\mathcal{G}, E', M')$. If r is an interpretation, then return r.*

If we reached the end, we know that all recursive calls returned $\perp$. In that case, $S_t(\mathcal{G}, E, M)$ also returns $\perp$.

Example 3.11. *Assume that we want to refute the sequent $\#[A](B \wedge C),\ \neg\#[A]B \vdash \perp$. Kleening the first formula results in $\#A \otimes (\neg A \oplus (\#B \otimes \#C))$. Radicalization results in $A_{\mathbf{f},\mathbf{t}} \otimes (A_{\mathbf{f}} \oplus (B_{\mathbf{f},\mathbf{t}} \otimes C_{\mathbf{f},\mathbf{t}}))$. This formula can be factored into the first three geometric formulas in Figure 2. Kleening the second formula results in $\neg\#A \oplus (A \otimes \neg\#B)$. Radicalization results in $A_{\mathbf{e}} \oplus (A_{\mathbf{t}} \otimes B_{\mathbf{e}})$. This formula can be factored into the last two formulas in Figure 2.*

We try to refute the set of formulas using algorithm S_t. We start with interpretation (E_0, M_0), defined by $E_0 = \{\}$ and $M_0 = \{\}$. Since the example is propositional, we will ignore the ground substitutions. All formulas are true in (E_0, M_0), except for the last two formulas $A_{\mathbf{e}} \oplus A_{\mathbf{t}}$ and $A_{\mathbf{e}} \oplus B_{\mathbf{e}}$.

Both formulas are not in conflict with (E_0, M_0). In the formula $A_{\mathbf{e}} \oplus A_{\mathbf{t}}$, both atoms $A_{\mathbf{e}}$ and $A_{\mathbf{t}}$ are false in (E_0, M_0) but not in conflict with (E_0, M_0). We have $(E_0, M_0) \Rightarrow_{A_{\mathbf{e}}} (E_0, M_0 \cup \{A_{\mathbf{e}}\})$, and $(E_0, M_0) \Rightarrow_{A_{\mathbf{t}}} (E_0, M_0 \cup \{A_{\mathbf{t}}\})$.

We continue search with $(E_1, M_1) = (\{\}, \{A_{\mathbf{e}}\})$*. Now the first formula* $A_{\mathbf{f},\mathbf{t}}$ *is false in* (E_1, M_1) *and it is in conflict with* (E_1, M_1)*.*

We backtrack and enter the other branch, which results in $(E_2, M_2) = (\{\}, \{A_{\mathbf{t}}\})$*. All formulas are true in* (E_2, M_2)*, except for the last formula* $A_{\mathbf{e}} \oplus B_{\mathbf{e}}$*.*

Atom $A_{\mathbf{e}}$ *is in conflict with* (E_2, M_2)*. The other atom* $B_{\mathbf{e}}$ *is false in* (E_2, M_2)*, but not in conflict. We have* $(E_2, M_2) \Rightarrow_{B_{\mathbf{e}}} (E_2, M_2 \cup \{B_{\mathbf{e}}\})$*.*

The resulting interpretation is $(E_3, M_3) = (\{\}, \{A_{\mathbf{t}}, B_{\mathbf{e}}\})$*. Now the second clause* $A_{\mathbf{f}} \oplus B_{\mathbf{f},\mathbf{t}}$ *is false in* (E_3, M_3)*, and it is in conflict with* (E_3, M_3)*.*

Since we have exhausted all possibilities, we have shown that the set of formulas is unsatisfiable.

This completes the discussion of the search algorithm. In its present form, the algorithm is extremely inefficient, which mostly due to the fact that it backtracks through all elements of E whenever it encounters an existential quantifier. In order to overcome these problems, the calculus has been extended with lemma learning. Details can be found in [7]. Experiments with the two-valued version of classical logic ([16]) have shown that by using learning, it is possible to obtain a practical calculus.

4 Flattening

It remains to show that sequents consisting of radicalized formulas can be transformed into sequents of geometric formulas. The transformation is mostly straightforward, and similar to standard transformations to clauses. (See [15]) The main difference is that, in order to obtain geometric formulas, function symbols have to be replaced by relation symbols. The transformations used in [1, 8] can be adopted without difficulty.

Definition 4.1. *We assume a mapping that maps every* n*-arity function symbol* f *to a unique* $(n+1)$*-arity predicate symbol* $\overline{F}$*.*

Let A *be a Kleene formula that is in NNF and radicalized. An* anti-Skolemization *of* A *is a formula that is the result of making the following replacement as long as* A *contains functional terms:*

Select a functional term $f(x_1, \ldots, x_n)$ *in which all* $x_1, \ldots, x_n$ *are variables. Such a term necessarily exists. It is possible that* $n = 0$*.*

Write A *in the form* $A[\ B[\ f(x_1, \ldots, x_n)\ ]\ ]$*, where* B *is a subformula of* A *that contains at least one of the occurrences of* $f(x_1, \ldots, x_n)$*. Replace* $A[\ B[\ f(x_1, \ldots, x_n)\ ]\ ]$ *by* $A[\ \Pi z\ \overline{F}_{\mathbf{f},\mathbf{e}}(x_1, \ldots, x_n, z) \oplus B[z]\ ]$*.*

After anti-Skolemization we almost have geometric logic. First rewrite the formulas, using the rewrite rules in Figure 3. If a formula has form $A \otimes B$, then it can be replaced by A and B seperately.

As long as one of the sequents contains a formula A that contains an existentially quantified subformula $\Sigma y\ P[y]$ for which $P[y]$ is not a geometric atom or does not contain y, write A in the form $A[\ \Sigma y\ P[x_1, \ldots, x_n, y]\ ]$, where $x_1, \ldots, x_n$ are the other free variables of P. Replace $A[\ \Sigma y\ P[x_1, \ldots, x_n, y]\ ]$ by $A[\ \Sigma y\ p_{\mathbf{t}}(x_1, \ldots, x_n, y)\ ]$ and $\Pi x_1 \cdots x_n\ \Pi y\ p_{\mathbf{f},\mathbf{e}}(x_1, \ldots, x_n, y) \oplus P[x_1, \ldots, x_n, y]$, using a new predicate symbol p.

Now every formula has form $\Pi \overline{x}\ (A_1 \oplus \cdots \oplus A_p)$, where $\overline{x}$ is a set of variables, and each element of $\overline{A}$ has one of the following five forms: **(1)** A geometric atom. (See Definition 3.1), **(2)** a negative equality $x_1 \not\approx x_2$ with $x_1, x_2 \in \overline{x}$, **(3)** a positive equality of form $x \approx x$ with $x \in \overline{x}$, (Positive equalities with distinct variables are geometric atoms) **(4)** $\bot$, or **(5)** $\top$. For

Figure 3: Rewriting into Geometric Formulas

$$\begin{array}{lcl}
A \oplus (B \otimes C) & \Rightarrow & (A \oplus B) \otimes (A \oplus C) \\
(A \otimes B) \oplus C & \Rightarrow & (A \oplus C) \otimes (B \oplus C) \\
\\
(\Pi x\ P[x]) \oplus A & \Rightarrow & \Pi x\ (P[x] \oplus A) \\
A \oplus \Pi x\ P[x] & \Rightarrow & \Pi x\ (A \oplus P[x]) \\
\Pi x\ (P[x] \otimes Q[x]) & \Rightarrow & (\Pi x\ P[x]) \otimes (\Pi x\ Q[x])
\end{array}$$

each of these possibilities, we proceed as follows: If $x \approx x$ or $\top$ occurs among the A_i, then the formula can be completely removed. If one of the A_i has form $\bot$ or $x \not\approx x$, then A_i can be removed from the formula. If one of the A_i is a negative equality of form $x_1 \not\approx x_2$ with $x_1 \neq x_2$, it can be substituted away. The result is:

$$\Pi \overline{x} \backslash \{x_2\}\ A_1[x_1 := x_2] \oplus \cdots \oplus A_{i-1}[x_1 := x_2] \oplus A_{i+1}[x_1 := x_2] \oplus \cdots \oplus A_n[x_1 := x_2].$$

When this procedure is finished, all formulas are geometric.

Example 4.2. *Consider the sequent*

$$\left.\begin{array}{l}
\forall x\ \#N(x) \\
N(0) \\
\forall x\ N(x) \rightarrow N(s(x)) \\
\neg N(s(s(0)))
\end{array}\right\} \vdash \bot$$

The sequent contains two function symbols 0 *and* s*. In order to translate it into geometric format, we introduce a predicate symbol* Z *with arity* 1 *and a predicate symbol* S *with arity* 2*. The result is*

$$\left.\begin{array}{l}
\Sigma y\ Z_{\mathbf{t}}(y) \\
\Pi x\ \Sigma y\ S_{\mathbf{t}}(x, y) \\
\\
\Pi x\ N_{\mathbf{f},\mathbf{t}}(x) \\
\Pi z\ Z_{\mathbf{f},\mathbf{e}}(z) \oplus N_{\mathbf{t}}(z) \\
\Pi xz\ S_{\mathbf{f},\mathbf{e}}(x, z) \oplus N_{\mathbf{f}}(x) \oplus N_{\mathbf{t}}(z) \\
\Pi z_1 z_2 z_3\ Z_{\mathbf{f},\mathbf{e}}(z_1) \oplus S_{\mathbf{f},\mathbf{e}}(z_1, z_2) \oplus S_{\mathbf{f},\mathbf{e}}(z_2, z_3) \oplus N_{\mathbf{f}}(z_3)
\end{array}\right\} \vdash \bot$$

5 Conclusions

We have introduced a theorem proving procedure for PCL, that is based on geometric logic. We chose geometric logic, partially because we are used to it, and partially because we expect that it can be tuned to terminate on sequents that originate from type checking conditions. Experiments with **Geo** (an implementation of two-valued, geometric logic), have shown that geometric logic is good at terminating on unprovable goals ([16]), and reasonably good at proving goals. Although geometric logic for Kleene logic is more complicated than geometric logic for two-valued logic, it is not fundamentally different. Because of this, we expect no difficulties with implementing it.

References

[1] 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.

[2] Peter Baumgartner, Ulrich Furbach, and Ilkka Niemelä. Hyper Tableaux. In *Logics in Artificial Intelligence (JELIA '96)*, number 1126 in LNAI. Springer, 1996.

[3] Marc Bezem and Thierry Coquand. Automating coherent logic. In Geoff Sutcliffe and Andrei Voronkov, editors, *LPAR*, volume 3835 of *LNCS*, pages 246–260. Springer, 2005.

[4] François Bry and Sunna Torge. A deduction method complete for refutation and finite satisfiability. In Jürgen Dix, Luis Fariñas del Cerro, and Ulrich Furbach, editors, *JELIA*, volume 1489 of *LNCS*, pages 122–138. Springer Verlag, 1998.

[5] Ádám Darvas, Farhad Mehta, and Arsenii Rudich. Efficient well-definedness checking. In Alessandro Armado, Peter Baumgartner, and Gilles Dowek, editors, *International Joint Conference on Automated Reasoning (IJCAR) 2008*, volume 5195 of *LNAI*, pages 100–115. Springer Verlag, 2008.

[6] Hans de Nivelle. Classical logic with partial functions. In Jürgen Giesl and Reiner Hähnle, editors, *International Joint Conference on Automated Reasoning (IJCAR) 2010*, volume 6173 of *LNAI*, pages 203–217. Springer Verlag, 2010.

[7] Hans de Nivelle. Theorem proving for classical logic with partial functions by reduction to Kleene logic. *Journal of Logic and Computation*, pages 1–49?, 2014? (accepted for publication, a preprint can be obtained from `www.ii.uni.wroc.pl/~nivelle/publications/`.

[8] Hans de Nivelle and Jia Meng. Geometric resolution: A proof procedure based on finite model search. In John Harrison, Ulrich Furbach, and Natarajan Shankar, editors, *International Joint Conference on Automated Reasoning 2006*, volume 4130 of *Lecture Notes in Artificial Intelligence*, pages 303–317, Seattle, USA, August 2006. Springer.

[9] William M. Farmer. Mechanizing the traditional approach to partial functions. In W. Farmer, M. Kerber, and M. Kohlhase, editors, *Proceedings of the Workshop on the Mechanization of Partial Functions (associated to CADE 13)*, pages 27–32, 1996.

[10] Reiner Hähnle. Many-valued logic, partiality, and abstraction in formal specification languages. *Logic Journal of the IGPL*, 13(4):415–433, 2005.

[11] Manfred Kerber and Michael Kohlhase. A mechanization of strong Kleene logic for partial functions. In *Automated Deduction - CADE 12*, volume 814 of *LNAI*, pages 371–385. Springer Verlag, June 1994.

[12] Sun Kim and Hantao Zhang. ModGen: Theorem proving by model generation. In Barbara Hayes-Roth and Richard Korf, editors, *Proceedings of AAAI-94*, pages 162–167, 1994.

[13] Farhad Mehta. A practical approach to partiality - a proof based approach. In Shaoying Liu and Tom Maibaum, editors, *International Conference on Formal Engineering Methods, (ICFEM)*, volume 5256 of *LNCS*, pages 238–257. Springer, 2008.

[14] Neil Murray and Erik Rosenthal. Signed formulas: A liftable meta-logic for multiple-valued logics. In Jan Komorowski and Zbigniew Raś, editors, *Proceedings of the 7th International Symposium on Methodologies for Intelligent Systems (ISMIS)*, volume LNCS 689, pages 275–284, 1993.

[15] Andreas Nonnengart and Christoph Weidenbach. Computing small clause normal forms. In Alan Robinson and Andrei Voronkov, editors, *Handbook of Automated Reasoning*, volume I, chapter 6, pages 335–367. Elsevier Science B.V., 2001.

[16] G. Sutcliffe. The 3rd IJCAR Automated Theorem Proving Competition. *AI Communications*, 20(2):117–126, 2007.