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 $[Next]_v$ 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:

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

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.

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

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

We write $\mathcal{M}, w \models \varphi$ instead of $[\![\varphi]\!]^{\mathcal{M}}_w = \mathsf{tt}$. 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 $[\![e]\!]^{\mathcal{K}}_w \in \{\mathsf{tt}, \mathsf{ff}\}$ specializes to the following clauses:

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

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

As an example, coalescing the formula

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

yields the ML-formula

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

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*

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

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). 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

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

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:

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

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

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.