Reasoning with Triggers

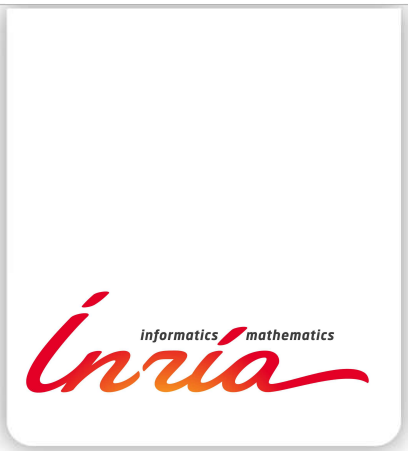
Claire Dross, Sylvain Conchon, Andrei Paskevich

Reasoning with Triggers

Claire Dross, Sylvain Conchon, Johannes Kanig, Andrei Paskevich

RESEARCH
REPORT
N° 7986
Juin 2012
Project-Team ProVal

ISSN 0249-6399 ISRN INRIA/RR--7986--FR+ENG



Reasoning with Triggers

Claire Dross*†‡, Sylvain Conchon*†, Johannes Kanig‡, Andrei Paskevich*†

Project-Team ProVal

Research Report n° 7986 — Juin 2012 — 26 pages

Abstract: SMT solvers can decide the satisfiability of ground formulas modulo a combination of built-in theories. Adding a built-in theory to a given SMT solver is a complex and time consuming task that requires internal knowledge of the solver. However, many theories (arrays [13], reachability [11]), can be easily expressed using first-order formulas. Unfortunately, since universal quantifiers are not handled in a complete way by SMT solvers, these axiomatics cannot be used as decision procedures.
In this paper, we show how to extend a generic SMT solver to accept a custom theory description and behave as a decision procedure for that theory, provided that the described theory is complete and terminating in a precise sense. The description language consists of first-order axioms with triggers, an instantiation mechanism that is found in many SMT solvers. This mechanism, which usually lacks a clear semantics in existing languages and tools, is rigorously defined here; this definition can be used to prove completeness and termination of the theory. We demonstrate on two examples, how such proofs can be achieved in our formalism.

Key-words: Quantifiers, Triggers, SMT Solvers, Theories

* LRI, Université Paris-Sud 11,CNRS, Orsay F-91405
† INRIA Saclay-Île de France, ProVal, Orsay F-91893
‡ AdaCore, Paris F-75009

RESEARCH CENTRE
SACLAY – ÎLE-DE-FRANCE

Parc Orsay Université
4 rue Jacques Monod
91893 Orsay Cedex

Raisonner avec des Déclencheurs

Résumé : Les solveurs SMT décident de la satisfiabilité d'un ensemble de formules closes modulo une combinaison de théories. Ajouter une théorie à cette combinaison est une tâche longue et complèxe qui requiert une connaissance approfondie du fonctionnement interne du solveur. Pourtant, de nombreuses théories (les tableaux [13], l'atteignabilité sur les listes chainées [11]), peuvent être exprimées facilement comme un ensemble de formules du premier ordre. Malheureusement, les solveurs SMT ne traitant pas les quantificateurs de façon complète, ces axiomatisations ne peuvent pas se substituer à l'écriture d'une vrai procédure de décision.

Dans ce papier, nous montrons comment un solveur SMT pour les formules closes peut être modifié pour qu'il accepte une théorie supplémentaire sous forme d'axiomatique. L'axiomatique devra être complète, dans un certain sens, pour que le solveur puisse l'utiliser comme procédure de décision. Pour cela, elle pourra être agrémentée de déclencheurs, des annotations utilisées par de nombreux solveurs pour guider l'instanciation. Ces annotations, habituellement utilisées comme heuristiques, ont une sémantique bien définies dans notre formalisme. Cela permet de les prendre en compte lors de nos preuves de complétude et de terminaison. Nous donnons finalement deux exemples de théories complètes pour notre formalisation.

Mots-clés : Quantificateurs, Déclencheurs, Solveurs SMT, Théories

Contents

1 Introduction

SMT solvers are sound, complete, and efficient tools for deciding the satisfiability of ground formulas modulo combinations of built-in theories such as linear arithmetic, arrays, bit-vectors etc. Usually, they work on top of a SAT solver which handles propositional formulas. Assumed literals are then handed to dedicated solvers for theory reasoning. These solvers are complete. Adding a new theory to the framework is a complex and time consuming task that requires internal knowledge of the solver. For some theories however, it is possible to give a first-order axiomatization. Unfortunately, even if a few SMT solvers also handle first-order formulas, for example, Simplify [7], CVC3 [9], Z3 [6] and Alt-Ergo [3], these axiomatizations cannot be used as theories. Indeed, these solvers are not complete when quantifiers are involved, even in the absence of theory reasoning.

SMT solvers handle universal quantifiers through an instantiation mechanism. They maintain a set of ground formulas (without quantifiers) on which theory reasoning is done. This set is periodically augmented by heuristically chosen instances of universally quantified formulas.

The heuristics for choosing new instances differ between SMT solvers. Nevertheless, it is commonly admitted that user guidance is useful in this matter [7, 12]. The choice of instances can be influenced by manually adding instantiation patterns, also known as *triggers*. These patterns are used to restrict instantiation to *known* terms that have a given form. Here is an example of a universally quantified formula with a trigger in SMT-LIB [2] notation:

```
(forall ((x Int)) (! (= (f x) c) :pattern ((g x))))
```

The syntax for triggers includes a bang (a general syntax for annotating formulas) before the restricted formula `(= (f x) c)` and the keyword `:pattern`

to introduce the trigger `(g x)`. The commonly agreed meaning of the above formula can be stated as follows:

Assume `(= (f t) c)` *only for terms* `t` *such that* `(g t)` *is known.*

Intuitively, a term is *known* when it appears in a ground fact assumed by the solver. However, that rule is quite vague and does not include answers to the following questions: when does a term become known? Is that notion to be considered modulo equality and modulo the built-in theories, and finally, when is this rule applied exactly, and to which formulas? Different provers have found different answers to these questions, consequence of the fact that triggers are considered a heuristics and not a language feature with precise semantics.

We give a proper semantics for first-order formulas with triggers. In this semantics, instantiation of universally quantified formulas is restricted to known terms. This makes it possible to extend a generic SMT solver so that it behaves as a decision procedure on an axiomatization representing a custom theory, provided the theory is complete in our framework. This enables non-expert users to add their own decision procedure to SMT solvers. Unlike first-order axiomatization in SMT solvers handling quantifiers, a proof of completeness and termination of the decision procedure can be attempted and, unlike manual implementation of decision procedures inside SMT solvers, it does not require internal knowledge of the solver.

In Sect. 2, we introduce a formal semantics for first-order logic with a notation for instantiation patterns that restrict instantiation. It formalizes both the notion of trigger and the dual notion of known term. We show that this extension of first-order logic is conservative: formulas without triggers preserve their satisfiability under this semantics. We present in Sect. 3 a theoretical way of extending a generic ground SMT solver so that it can turn an axiomatization T with triggers into a decision procedure, provided that T has some additional properties. Finally, in Sect. 4, we demonstrate on the non-extensional theory of arrays how our framework can be used to demonstrate that an axiomatization with triggers indeed fulfills its requirements.

Related Work. Triggers are a commonly used heuristic in SMT solvers that handle quantifiers. User manuals usually explain how they should be used to achieve the best performance [7, 12, 9]. Triggers can be automatically computed by the solvers. A lot of work has also been done on defining an efficient mechanism for finding the instances allowed by a trigger. These techniques, called E-matching, are described in [7, 13] for Simplify, in [5] for Z3, and in [9] for CVC3. Other heuristics for generating instances include model-based quantifier instantiation [8] and saturation processes closed to the superposition calculus [4].

In this paper, triggers are not handled in the usual manner. On the one hand, since SMT solvers are not complete in general when quantifiers are involved, they favor efficiency over completeness in the treatment of triggers. For example, they usually do not attempt to match triggers modulo underlying theories. On the other hand, in our framework, triggers are used to define theories, and they need therefore to be handled in a complete way.

Triggers can also be used in complete first-order theorem provers to guide the proof search and improve the solver's efficiency. This is done on a complete

solver for a subset of first-order logic with linear arithmetics based on a sequent calculus in [14].

As for using an SMT solver as a decision procedure, the related idea that a set of first-order formulas can be saturated with a finite set of ground instances has been explored previously. For example, in [10], decision procedures for universally quantified properties of functional programs are designed using local model reasoning. In the same way, Ge and de Moura describe fragments of first-order logic that can be decided modulo theory by saturation [8]. Both of these works define a restricted class of universally quantified formulas that can be finitely instantiated. We do not impose such restrictions a priori but rather require a dedicated proof of completeness.

2 First-Order Logic with Triggers and Witnesses

In this section, we extend classical first-order logic, denoted FOL, with constructions to specify instantiation patterns and known terms. The semantics of this extension, denoted FOL*, is defined through an encoding into usual first-order logic. In the rest of the article, we write formulas in standard mathematical notation.

2.1 Syntax

Informally, a trigger is a guard that prevents the usage of a formula until the requested term is known. We write it $[t]\,F$, which should be read *if the term t and all its sub-terms are known then assume F*. Note that we do not require a trigger to be tied to a quantifier. We separate the actual instantiation of a universal formula from the decision to use its result.

A dual construct for $[t]\ F$, which we call *witness*, is written $\langle t\rangle\ F$ and is read *assume that the term t and all its sub-terms are known and assume F*. This new construction explicitly updates the set of known terms, something for which there is no proper syntax in existing languages.

The extended syntax of formulas can be summarized as follows:

$$F ::= A \mid F_1 \wedge F_2 \mid F_1 \vee F_2 \mid \forall x.\ F \mid \exists x.\ F \mid \langle t\rangle\ F \mid [t]\ F \mid \neg F$$

We treat implication ($\rightarrow$) and equivalence ($\leftrightarrow$) as abbreviations, in a standard fashion.

2.2 Denotational Semantics

We define the semantics of our language via two encodings $[\![\ ]\!]^+$ and $[\![\ ]\!]^-$ into first-order language, given in Fig. 1. The notation $[\![\ ]\!]^\pm$ is used when the rule is the same for both polarities and the polarity of the sub-formulas does not change. We introduce a fresh unary predicate symbol *known* which denotes the fact that a term is known. Given a term t or an atomic formula A, we denote with $\mathcal{T}(t)$ (respectively, $\mathcal{T}(A)$) the set of all the non-variable sub-terms of t (resp. A). The expression $\mathit{known}(\mathcal{T}(t))$ stands for the conjunction $\bigwedge_{t'\in\mathcal{T}(t)} \mathit{known}(t')$.

We require universally quantified formulas to be instantiated with known terms. This is consistent with the standard use of triggers: indeed, SMT solvers

$$
\begin{array}{rcl@{\qquad}rcl}
[\![F_1 \wedge F_2]\!]^{\pm} & \triangleq & [\![F_1]\!]^{\pm} \wedge [\![F_2]\!]^{\pm} & [\![\langle t \rangle\ F]\!]^{\pm} & \triangleq & \mathit{known}(\mathcal{T}(t)) \wedge [\![F]\!]^{\pm} \\
[\![F_1 \vee F_2]\!]^{\pm} & \triangleq & [\![F_1]\!]^{\pm} \vee [\![F_2]\!]^{\pm} & [\![[t]\ F]\!]^{\pm} & \triangleq & \mathit{known}(\mathcal{T}(t)) \rightarrow [\![F]\!]^{\pm} \\
[\![\forall x.\ F]\!]^{\pm} & \triangleq & \forall x.\ \mathit{known}(x) \rightarrow [\![F]\!]^{\pm} & [\![\exists x.\ F]\!]^{\pm} & \triangleq & \exists x.\ \mathit{known}(x) \wedge [\![F]\!]^{\pm} \\
[\![\neg F]\!]^{+} & \triangleq & \neg [\![F]\!]^{-} & [\![\neg F]\!]^{-} & \triangleq & \neg [\![F]\!]^{+} \\
[\![A]\!]^{+} & \triangleq & \mathit{known}(\mathcal{T}(A)) \rightarrow A & [\![A]\!]^{-} & \triangleq & \mathit{known}(\mathcal{T}(A)) \wedge A
\end{array}
$$

Figure 1: Semantics of FOL*-formulas ($[\![\,]\!]^{\pm}$ denotes either $[\![\,]\!]^{+}$ or $[\![\,]\!]^{-}$)

require (or compute) a trigger containing each quantified variable for every universal quantifier. Then every term that replaces a universally quantified variable is necessarily known, since sub-terms of a known term are known, too. Dually, every existentially quantified variable is assumed to be known. This is necessary in order to allow instantiation with a witness from an existential formula.

To maintain the invariant that the sub-terms of a known term are also known, our interpretation of $\langle t \rangle\ F$ implies the presence of every non-variable sub-term of t (the presence of variables is assured by interpretation of the quantifiers). Dually, $[t]\ F$ requires the presence of every non-variable sub-term of t; due to the mentioned invariant, this is not a restriction.

Finally, whenever we encounter an atomic formula, regardless of its polarity, we assume the presence of its sub-terms. This is also in agreement with the standard use of triggers.

We define entailment in FOL* as follows:

$$F \vdash^\star G \quad \triangleq \quad \mathit{known}(\omega),\ [\![F]\!]^{-} \vdash [\![G]\!]^{+}$$

where ω is an arbitrary fresh constant supposed to be known *a priori*, and $\vdash$ stands for entailment in FOL.

A peculiar aspect of FOL* is the cut rule is not admissible in it. Indeed, one cannot prove $\forall x\,.\,[f(x)]\ P(f(x)),\ \forall x\,.\,[f(g(x))]\ (P(f(g(x))) \rightarrow Q(x)) \vdash^\star Q(c)$, since the term $f(g(c))$ is not known and neither of the premises can be instantiated. However, $Q(c)$ is provable via an intermediate lemma $P(f(g(c))) \rightarrow Q(c)$.

2.3 Example

Consider the following set of formulas R from the previous section:

$$R = \{\ f(0) \approx 0, \quad f(1) \not\approx 1, \quad \forall x.[f(x+1)]\ f(x+1) \approx f(x) + 1\ \}$$

We want to show that R is unsatisfiable in FOL*, that is to say, $R \vdash^\star \bot$. By definition, we have to prove that $\mathit{known}(\omega),\ [\![R]\!]^{-} \vdash \bot$.

$$[\![R]\!]^{-} = \left\{ \begin{array}{l} \mathit{known}(\mathcal{T}(f(0) \approx 0)) \wedge f(0) \approx 0, \\ \mathit{known}(\mathcal{T}(f(1) \not\approx 1)) \wedge f(1) \not\approx 1, \\ \forall x.\ \mathit{known}(x) \rightarrow \mathit{known}(\mathcal{T}(f(x+1))) \rightarrow \\ \qquad \mathit{known}(\mathcal{T}(f(x+1) \approx f(x)+1)) \wedge f(x+1) \approx f(x) + 1 \end{array} \right\}$$

The set of formulas $[\![R]\!]^{-}$ is unsatisfiable in first-order logic with arithmetic. Therefore, in our framework, the initial set R is unsatisfiable.

2.4 The Extension of First-Order Logic is Conservative

Even if a formula does not contain triggers or witnesses, our encoding modifies it to restrict instantiation of universal formulas. However, it preserves satisfiability of formulas in classical first-order logic with equality.

Theorem 2.1 (Soundness)**.** *For every first-order formula F, if we have $F \vdash^\star \bot$, then we also have $F \vdash \bot$.*

Proof. Since *known* is a fresh predicate symbol, for every model M of F, there is a model M' of F such that M' only differs from M in the interpretation of *known* and $M' \vdash \forall x.known(x)$. By immediate induction, $(\forall x.known(x)) \wedge F \vdash [\![F]\!]^-$. As a consequence, M' is a model of $[\![F]\!]^-$ and $F \not\vdash^\star \bot$. Thus, by contra-position, if $F \vdash^\star \bot$ then there is no model of F and $F \vdash \bot$. □

Theorem 2.2 (Completeness)**.** *For any first-order formula F, if $F \vdash \bot$ then $F \vdash^\star \bot$.*

The rest of the section is dedicated to the proof of this theorem. We will often resort to the following lemma:

Lemma 2.1. *If F is an arbitrary* FOL**-formula, t a ground term, and x a free variable of F, then $[\![F]\!]^-[x \leftarrow t]$, $known(\mathcal{T}(t)) \vdash [\![F[x \leftarrow t]]\!]^-$.*

Instead of working with models, we choose an approach based on inference trees in a certain paramodulation calculus. Since this calculus operates on clauses, we must first show that skolemization and clausification preserve satisfiability. With a small abuse of notation, we use our encoding $[\![\ ]\!]^-$ on clauses. Formally, if C is a clause, $[\![C]\!]^-$ is $[\![\forall \overline{x}.C]\!]^-$ where $\overline{x}$ are the free variables of C.

Definition 2.1. *We introduce a function $\mathcal{S}ko$ that takes a first-order formula in negative normal form and returns its skolemization, and a function $\mathcal{C}ls$ that puts a formula in Skolem negative normal form into conjunctive normal form.*

Lemma 2.2. *For every first-order formula F in negative normal form, if we have $\mathcal{S}ko(F) \vdash^\star \bot$ then we also have $F \vdash^\star \bot$.*

Proof. Skolemization preserves satisfiability. As a consequence, it is enough to show that $\mathcal{S}ko([\![F]\!]^-) \vdash \bot$.

Let G be a sub-formula of F and σ a substitution mapping variables existentially quantified above G of F to their skolem functions. We prove that we have $\mathcal{S}ko([\![G]\!]^-)\sigma \wedge \bigwedge_{t \in \mathcal{T}(Image(\sigma))} known(t) \vdash [\![\mathcal{S}ko(G)\sigma]\!]^-$ by induction over the form of G.

- G is a literal. We compute that $\mathcal{S}ko([\![G]\!]^-) = \bigwedge_{t \in \mathcal{T}(G)} known(t) \wedge G$ and $[\![\mathcal{S}ko(G)\sigma]\!]^- = \bigwedge_{t \in \mathcal{T}(G\sigma)} known(t) \wedge G\sigma$. A non-variable sub-term of $G\sigma$ s is either $t\sigma$ for some non-variable sub-term t of G or a non-variable sub-term of $t \in Image(\sigma)$. In both cases, the presence of s is implied by $\bigwedge_{t \in \mathcal{T}(G)} known(t\sigma) \wedge \bigwedge_{t \in \mathcal{T}(Image(\sigma))} known(t)$. Thus, we have $\mathcal{S}ko([\![G]\!]^-)\sigma \wedge \bigwedge_{t \in Image(\sigma)} known(t) \wedge \bigwedge_{y \in \overline{y}} known(y) \vdash [\![\mathcal{S}ko(G)\sigma]\!]^-$.
- G is a disjunction $G_1 \vee G_2$. We have $\mathcal{S}ko([\![G]\!]^-) = \mathcal{S}ko([\![G_1]\!]^-) \vee \mathcal{S}ko([\![G_2]\!]^-)$. By induction hypothesis, we have $\mathcal{S}ko([\![G_k]\!]^-)\sigma \wedge \bigwedge_{t \in \mathcal{T}(Image(\sigma))} known(t)$ implies $[\![\mathcal{S}ko(G_k)\sigma]\!]^-$ for k in 1..2. Consequently, we deduce $\mathcal{S}ko([\![G]\!]^-) \wedge \bigwedge_{t \in \mathcal{T}(Image(\sigma))} known(t) \vdash [\![\mathcal{S}ko(G_1)]\!]^- \vee [\![\mathcal{S}ko(G_2)]\!]^-$ which is $[\![\mathcal{S}ko(G)]\!]^-$. The same is true for conjunctions.

$$\textsc{Reflexivity}\quad \frac{C \vee t_1 \not\approx t_2 \cdot \rho \qquad \mu = \mathrm{mgu}(t_1\rho, t_2\rho)}{C \cdot \rho\mu}$$

$$\textsc{Paramodulation}\quad \frac{C_1 \vee t_1 \approx t_2 \cdot \rho_1 \qquad C_2 \vee L[s] \cdot \rho_2 \qquad \mu = \mathrm{mgu}(t_1\rho_1, s\rho_2)}{C_1 \vee C_2 \vee L[t_2] \cdot (\rho_1 \oplus \rho_2)\mu}$$

$$\textsc{Factoring}^{1}\quad \frac{C \vee L_1 \vee L_2 \cdot \rho \qquad \mu = \mathrm{mgu}(L_1\rho, L_2\rho) \qquad L\rho\mu = L_1\rho\mu \qquad L \ll L_1 \qquad L \ll L_2}{C \vee L \cdot \rho\mu}$$

Figure 2: Basic paramodulation calculus **BP**

- G is a universally quantified formula $\forall x.\ G'$. We compute $\mathcal{S}ko([\![G]\!]^-) = \forall x.\ known(x) \to [\![G']\!]^-$. By induction hypothesis, we have $\mathcal{S}ko([\![G']\!]^-)\sigma \wedge \bigwedge_{t\in\mathcal{T}(Image(\sigma))} known(t) \vdash [\![\mathcal{S}ko(G')\sigma]\!]^-$. As a consequence, we deduce that $\mathcal{S}ko([\![G]\!]^-) \wedge \bigwedge_{t\in\mathcal{T}(Image(\sigma))} known(t) \vdash \forall x.\ known(x) \to [\![\mathcal{S}ko(G')\sigma]\!]^-$.
- G is an existentially quantified formula $\exists x.\ G'$. $c(\overline{x})$ is a Skolem function such that $\mathcal{S}ko([\![G]\!]^-) = known(c(\overline{x})) \wedge [\![G']\!]^-[x \leftarrow c(\overline{x})]$. By induction hypothesis, $\mathcal{S}ko([\![G']\!]^-)(\sigma\oplus\{x \mapsto c(\overline{x})\}) \wedge \bigwedge_{t\in\mathcal{T}(Image(\sigma\oplus\{x\mapsto c(\overline{x})\}))} known(t) \vdash [\![\mathcal{S}ko(G')\sigma \oplus \{x \mapsto c(\overline{x})\}]\!]^-$. Consequently, $\mathcal{S}ko([\![G]\!]^-) \vdash [\![\mathcal{S}ko(G')\sigma[x \leftarrow c(\overline{x})]]\!]^-$ which is equal to $[\![\mathcal{S}ko(G)]\!]^-$.

□

Lemma 2.3. *For every first-order formula F in negative normal form without existential quantifiers, if $\mathcal{C}ls(F) \vdash^\star \bot$ then $F \vdash^\star \bot$.*

Proof. The result comes from the preservation of conjunction and disjunction through the encoding and the two following facts: $[\![(\forall x.\ F_1) \vee (\forall y.\ F_2)]\!]^-$ is equal to $\forall x, y.\ (known(x) \to [\![F_1]\!]^-) \vee (known(y) \to [\![F_2]\!]^-)$ which implies $\forall x.\ known(x) \to (\forall y.\ known(y) \to ([\![F_1]\!]^- \vee [\![F_2]\!]^-)) = [\![\forall x, y.\ F_1 \vee F_2]\!]^-$; $[\![(\forall x.\ F_1) \wedge (\forall y.\ F_2)]\!]^-$ is equal to $\forall x, y.\ (known(x) \to [\![F_1]\!]^-) \wedge (known(y) \to [\![F_2]\!]^-)$ which implies $\forall x.\ known(x) \to (\forall y.\ known(y) \to ([\![F_1]\!]^- \wedge [\![F_2]\!]^-)) = [\![\forall x, y.\ F_1 \wedge F_2]\!]^-$. □

We define a basic paramodulation calculus **BP** in Fig. 2. This calculus works on *closures*, *i.e.*, pairs consisting of a clause and a substitution. If a closure $C \cdot \sigma$ can be inferred from a set of closures E, then $\{D\mu \mid D \cdot \mu \in E\} \vdash C\sigma$. If σ and ρ are substitutions, we denote their composition with $\sigma\rho$ and their disjoint union with $\sigma \oplus \rho$. Given two formulas F_1 and F_2, we write $F_1 \ll F_2$ if F_1 is more general than F_2, *i.e.*, there is a substitution σ such that $F_1\sigma = F_2$.

Lemma 2.4. *The calculus* **BP** *is sound and complete.*

[1] In traditional factoring rules, either L_1 or L_2 is kept in the conclusion $C \vee L_i$. Our FACTORING rule is not more restrictive. Indeed, such a literal L can always be computed for two unifiable literals L_1 and L_2.

We give in appendix a proof of completeness of **BP** similar to the one in [1].

Definition 2.2. *We define an* inference tree T *to be a deduction by one or more steps of* **BP** *starting from a set of clauses (called* leaves*) associated with empty substitutions. We denote the set of leaves with E_T and the inferred closure with $C_T \cdot \rho_T$ and call C_T the* conclusion *of T and ρ_T the* unifier *of T. An inference tree T is called a* refutation *if C_T is the empty clause.*

Definition 2.3. *We say that a step S of* **BP** involves *a literal L if either S is a reflexivity step and L the resolved disequality, S is a factoring step and L is one of the identified literals, or S is a paramodulation step and L is the literal rewritten or the equality used to rewrite it.*

To prove Theorem 2.2, we start with a refutation of the set $E = \mathcal{C}ls(\mathcal{S}ko(F))$ and show that the set $known(\omega), [\![E]\!]^-$ is unsatisfiable. Then, by Lemmas 2.2 and 2.3, $known(\omega), [\![F]\!]^-$ is unsatisfiable, too. For our proof, we will need a more general statement, namely: for every inference tree T, the set $known(\omega), [\![E_T]\!]^-$ implies every ground part of $[\![C_T]\!]^-$. Formally, this lemma can be stated as follows:

Lemma 2.5. *Given an inference tree T and a ground clause C such that the set $\mathcal{C}ls([\![C_T]\!]^-)$ contains the clause $(\bigwedge_{x \in vars(C_T)} known(x) \to C)$, we have $[\![E_T]\!]^-, known(\omega) \vdash C$.*

Proof. We proceed by induction on T using the following well-founded ordering: an inference tree T_1 is said to be strictly smaller than T_2 if either:

- There is a variable x in E_{T_2} such that T_1 is T_2 with x replaced by some term $t \gg x\rho_{T_2}$.
- T_1 is composed of strictly fewer inference steps than T_2.

If $C_T \in E_T$, then $[\![E_T]\!]^- \vdash [\![C_T]\!]^-$. Since $(\bigwedge_{x \in vars(C_T)} known(x) \to C)$ is in $\mathcal{C}ls([\![C_T]\!]^-)$, we have $known(\omega), [\![E_T]\!]^- \vdash C[vars(C_T) \leftarrow \omega]$. Since C is ground, $known(\omega)$, $[\![E_T]\!]^- \vdash C$.

If E_T is ground, a simple structural induction on T gives $[\![E_T]\!]^- \vdash [\![C_T]\!]^-$.

Lemma 2.6. *For every inference tree T with no variable, $[\![E_T]\!]^- \vdash [\![C_T]\!]^-$.*

Proof. We prove by induction over the form of T, that, for all clause $C \in \mathcal{C}ls[\![C_T]\!]^-$, we have $[\![E_T]\!]^- \vdash C$.

- If the last step of T is an application of Reflexivity:

$$\dfrac{\dfrac{S}{C_1 \vee t \not\approx t \cdot \varnothing}}{C_1 \cdot \varnothing}$$

  The inference tree S is strictly smaller than T. By induction hypothesis, since $C \vee t \not\approx t \in \mathcal{C}ls[\![C_S]\!]^-$, we have $[\![E_S]\!]^- \vdash C \vee t \not\approx t$. We consequently have $[\![E_T]\!]^- = [\![E_S]\!]^- \vdash C$.

- If the last step of T is an application of FACTORING:

$$\dfrac{\dfrac{S}{C_1 \vee L \vee L \cdot \varnothing}}{C_1 \vee L \cdot \varnothing}$$

  The inference tree S is strictly smaller than T. By construction, $C = C'_1 \vee L'$ where $C'_1 \in \mathcal{C}ls([\![C_1]\!]^-)$ and $L' \in \mathcal{C}ls([\![L]\!]^-)$. By induction hypothesis, since $C'_1 \vee L' \vee L' \in \mathcal{C}ls[\![C_S]\!]^-$, we have $[\![E_S]\!]^- \vdash C'_1 \vee L' \vee L'$. We consequently have $[\![E_T]\!]^- \vdash [\![E_S]\!]^- \vdash C$.

- If the last step of T is an application of PARAMODULATION:

$$\dfrac{\dfrac{S_1}{C_1 \vee s \approx t \cdot \varnothing} \qquad \dfrac{S_2}{C_2 \vee L[s] \cdot \varnothing}}{C_1 \vee C_2 \vee L[t] \cdot \varnothing}$$

  The inference trees S_1 and S_2 are strictly smaller than T. By construction, $C = C'_1 \vee C'_2 \vee K$ where $C'_1 \in \mathcal{C}ls([\![C_1]\!]^-)$, $C'_2 \in \mathcal{C}ls([\![C_2]\!]^-)$ and $K \in \mathcal{C}ls([\![L(t)]\!]^-)$. If $K = \mathit{known}(t')$, $t' \in \mathcal{T}(t)$, then $K \in [\![s \approx t]\!]^-$ and $[\![E_T]\!]^- \vdash [\![E_{S_1}]\!]^- \vdash C'_1 \vee K$ by induction hypothesis. Otherwise, $K = L'[t]$. By definition of $[\![\,]\!]^-$, $L'[s] \in \mathcal{C}ls([\![L[s]]\!]^-)$ and, by induction hypothesis, $[\![E_T]\!]^- \vdash [\![E_{S_1}]\!]^- \vdash C'_1 \vee s \approx t$ and $[\![E_T]\!]^- \vdash [\![E_{S_2}]\!]^- \vdash C'_2 \vee L'[s]$. In both cases, $[\![E_T]\!]^- \vdash C$.

□

Since $C \in \mathcal{C}ls([\![C_T]\!]^-)$, we obtain $\mathit{known}(\omega), [\![E_T]\!]^- \vdash C$.

If $E_T\rho_T$ is not ground, we consider a tree T', which is T where the free variables of $E_T\rho_T$ are replaced with ω. It is smaller than T with respect to our ordering and $C \in \mathcal{C}ls([\![C_{T'}]\!]^-)$, since C is ground. By induction hypothesis, $\mathit{known}(\omega), [\![E_{T'}]\!]^- \vdash C$. By Lemma 2.1, we obtain $\mathit{known}(\omega), [\![E_T]\!]^- \vdash C$.

Otherwise, the substitution ρ_T is ground, is non-empty, and ranges over all the variables of E_T. We want to reduce T by instantiating one of the free variables x of E_T with $x\rho_T$. Using Lemma 2.1 and the induction hypothesis, we obtain $\mathit{known}(\omega), [\![E_T]\!]^-, \mathit{known}(\mathcal{T}(x\rho_T)) \vdash C$. Then it suffices to find a variable x such that $\mathit{known}(\omega), [\![E_T]\!]^- \vdash C \vee \mathit{known}(\mathcal{T}(x\rho_T))$.

The idea is to find a literal L involved in T that contains a ground term t such that $t = x\rho_T$ for some x in E_T. We cut off every branch involving L in T and obtain a tree T', strictly smaller than T, and such that $C_{T'}$ contains L and literals of C_T (or instances thereof, if one of the erased inference steps was factoring). With the induction hypothesis, we deduce that $\mathit{known}(\omega), [\![E_{T'}]\!]^- \vdash C \vee \mathit{known}(\mathcal{T}(t))$ (recall that $[\![L]\!]^- \leftrightarrow \mathit{known}(\mathcal{T}(L)) \wedge L$). Since $E_{T'} \subseteq E_T$, we obtain $\mathit{known}(\omega), [\![E_T]\!]^- \vdash C \vee \mathit{known}(\mathcal{T}(t))$.

More formally, we can show that there is a ground term t in the range of ρ_T and a branch B of T such that C_B contains L, $t \in \mathcal{T}(L)$, and the next step of T involves L.

Lemma 2.7. *For every inference tree T such that the substitution ρ_T is ground, is non-empty, and ranges over all the variables of E_T, there is a term $t \in \mathit{Image}(\rho_T)$ and a branch B of T such that $C_B = C \vee L$, $t \in \mathcal{T}(L)$ and the next step of T involves L.*

Proof. Since ρ_T is ground, non-empty and ranges over all the variables of E_T, there is a ground term $t \in \mathit{Image}(\rho_T)$.

By induction over the size of a tree T, we show that, if there is a ground term $t \in \mathcal{T}(t')$ with $t' \in \mathit{Image}(\rho_T)$, then there is a ground term s such that either $s \in \mathit{Image}(\rho_T)$ or $s = t$ and a branch B of T such that $s \in \mathcal{T}(L)$ for L one of the literals of C_B involved in the next step of T.

- If the inference tree T has only one node, either t is a sub-term of an involved literal or one of its proper sub-terms is in $\mathit{Image}(\rho_T)$. In the second case, by immediate induction, there is a term s of $\mathit{Image}(\rho_T)$ that appears in an involved literal.
- Otherwise, by construction of ρ_T, there is a branch B of T such that $t' \in \mathcal{T}(L\rho_B)$, L being one of the literals of B involved in the next step of T.
  - $t \in \mathcal{T}(L)$. The term $s = t$ has the required properties.
  - There is a term $t' \in \mathit{Image}(\rho_B)$ such that $t \in \mathcal{T}(t')$. The branch B is strictly smaller than T. By induction hypothesis, there is a ground term s such that either $s \in \mathit{Image}(\rho_B)$ or $s = t$ and a branch B' of B such that $s \in \mathcal{T}(L)$ for L one of the literals of $C_{B'}$ involved in the next step of T. Since B' is also a branch of T, the proof is over.
  - There is a term $t' \in \mathit{Image}(\rho_B)$ such that $t' \in \mathcal{T}(t)$. By induction hypothesis, there is a ground term s such that either $s \in \mathit{Image}(\rho_B)$ or $s = t'$ and a branch B' of B such that $s \in \mathcal{T}(L)$ for L one of the literals of $C_{B'}$ involved in the next step of T. Since $t' \in \mathit{Image}(\rho_B)$, the conclusion follows in both cases.

□

By a direct induction over the form of T, there is a tree T' strictly smaller than T such that $E_{T'} \subseteq E_T$, $C_{T'} = L \vee D'$ and there is a clause $D \subseteq C_T$ such that $D \ll D'$.

Lemma 2.8. *For every branch B of a tree T and for every literal $L \in C_B$ involved in the next step of T, there is a tree T' with strictly fewer deductions than T such that $E_{T'} \subseteq E_T$ and $C_{T'} = L \vee C'$ where C' is such that there is a clause $C \subseteq C_T$ such that $C \ll C'$ and $C'\rho_{T'} \ll C\rho_T$.*

Proof. We proceed by induction over the number of steps following B in T.

- If the next step following B in T is the last one, then it is straightforward to check, by case analysis, that $T' = B$ has the requested properties.
- Otherwise we proceed by case analysis over the last step of the proof.
  - If the last step of T is an application of Reflexivity:

$$\frac{\dfrac{S}{C_1 \vee t_1 \not\approx t_2 \cdot \rho_S} \qquad \mu = \mathrm{mgu}(t_1\rho_S, t_2\rho_S)}{C_1 \cdot \rho_S\mu}$$

By induction hypothesis, there is S' strictly smaller than S such that $C_{S'} = L \vee C'$ and there is $C \subseteq C_S$ such that $C \ll C'$ and $C'\rho_{S'} \ll C\rho_S$. If $C \subseteq C_1$, the tree S' has the requested properties. Otherwise, since $C'\rho_{S'} \ll C\rho_S$, the last step of the proof can be reproduced. As a consequence, $C_{T'}$ can be written $L \vee C'$ with $C'\rho_{T'} = (C \setminus t_1 \not\approx t_2)\rho_T$ and $C' \subseteq C_{S'}$ less general than $C \setminus t_1 \not\approx t_2$.

– If the last step of T is an application of Factoring:

$$\frac{\dfrac{S}{C_1 \vee L_1 \vee L_2 \ \cdot\ \rho_S} \qquad \mu = \mathrm{mgu}(L_1\rho_S, L_2\rho_S) \quad L\rho_S\mu = L_1\rho_S\mu \quad L \ll L_1 \quad L \ll L_2}{C_1 \vee K \ \cdot\ \rho_S\mu}$$

By induction hypothesis, there is S' strictly smaller than S such that $C_{S'} = L \vee C'$ and there is $C \subseteq C_S$ such that $C \ll C'$ and $C'\rho_{S'} \ll C\rho_S$. If $C \subseteq C_1 \vee L_i$, the tree S' has the requested properties, L_i being less general than K. Otherwise, since $C'\rho_{S'}$ is more general than $C\rho_S$, the last step of the proof can be reproduced. As a consequence, $C_{T'}$ can be written $L \vee C' \vee K'$. Since $C_{S'}$ is less general than C, K' can be less general than K and $C' \subseteq C_{S'}$.

– If the last step of T is an application of Paramodulation:

$$\frac{\dfrac{S}{C_1 \vee t_1 \approx t_2 \ \cdot\ \rho_S} \qquad \dfrac{\dots}{C_2 \vee L[s] \ \cdot\ \rho} \qquad \mu = \mathrm{mgu}(t_1\rho_S, s\rho)}{C_1 \vee C_2 \vee L[t_2] \ \cdot\ (\rho_S \oplus \rho)\mu}$$

By induction hypothesis, there is S' strictly smaller than S such that $C_{S'} = L \vee C'$ and there is $C \subseteq C_S$ such that C' is less general than C and $C'\rho_{S'}$ is more general than $C\rho_S$. If $C \subseteq C_1$, the tree S' has the requested properties. Otherwise, since $C'\rho_{S'}$ is more general than $C\rho_S$ and C' is less general than C, the last step of the proof can be reproduced.

– If the last step of T is an application of Paramodulation:

$$\frac{\dfrac{\dots}{C_1 \vee t_1 \approx t_2 \ \cdot\ \rho} \qquad \dfrac{S}{C_2 \vee L[s] \ \cdot\ \rho_S} \qquad \mu = \mathrm{mgu}(t_1\rho, s\rho_S)}{C_1 \vee C_2 \vee L[t_2] \ \cdot\ (\rho \oplus \rho_S)\mu}$$

By induction hypothesis, there is S' strictly smaller than S such that $C_{S'} = L \vee C'$ and there is $C \subseteq C_S$ such that C' is less general than C and $C'\rho_{S'}$ is more general than $C\rho_S$. If $C \subseteq C_2$, the tree S' has the requested properties. Otherwise, since $C'\rho_{S'}$ is more general than $C\rho_S$, the last step of the proof can be reproduced.

□

Then there is $C' \subseteq C$ such that $(\bigwedge_{x \in vars(C_{T'})} known(x) \to C' \vee known(t')) \in \mathcal{C}ls([\![C_{T'}]\!]^-)$ for every $t' \in \mathcal{T}(t)$. Since T' is strictly smaller than T, we obtain, by induction hypothesis, that $known(\omega), [\![E_{T'}]\!]^- \vdash C' \vee known(t')$ for every $t' \in \mathcal{T}(t)$. Thus, $known(\omega), [\![E_T]\!]^- \vdash C \vee known(\mathcal{T}(t))$. □

3 Adding a Customizable Theory to a SMT Solver for Ground Formulas

In this section, we define a wrapper over a generic SMT solver for ground formulas that accepts a theory written as a set of formulas with triggers. This solver is a theoretical model and it is not meant to be efficient. We prove it sound with respect to our framework.

It is easy to show that conversion to NNF does not change the semantics of a FOL*-formula.

Definition 3.1. *We define a* skolemization transformation $\mathcal{S}ko_T$ *for* FOL**-formulas in negative normal form. Given a formula* $F = \exists x.\, G$*, we have* $\mathcal{S}ko_T(F) \triangleq \langle c(\overline{y})\rangle\ \mathcal{S}ko_T(G[x \leftarrow c(\overline{y})])$*, where* $\overline{y}$ *is the set of free variables of* F*, and* c *is a fresh function symbol.*

We put the witness $\langle c(\overline{y})\rangle$ to preserve satisfiability. Indeed, $\mathcal{S}ko(\exists x.\, [x]\,\bot)$ is $[c]\,\bot$ which is satisfiable, while $\exists x.\, [x]\,\bot$ is not. In the following, we work with FOL*-formulas in Skolem negative normal form.

3.1 A Solver for Ground Formulas

To reason about FOL*-formulas, we use a solver $\mathbf{S}$ for ground formulas.

Definition 3.2. *We denote implication over ground formulas with theories* $\vdash_o$ *to distinguish it from implication in first-order logic with theories* $\vdash$*.*

We make a few assumptions about the interface of the ground solver $\mathbf{S}$:

- It returns $\mathit{Unsat}(U)$ when called on an unsatisfiable set of ground formulas R where U is an unsatisfiable core of R. We assume that U is a set of formulas that occur in R such that $R \vdash_o U$ and $U \vdash_o \bot$.
- It returns $\mathit{Sat}(M)$ when called on a satisfiable set of ground formulas R where M is a model of R. We assume that M is a set of literals of R such that $M \vdash_o R$.

We write $R \rightsquigarrow_{\mathbf{S}} \mathit{Unsat}(U)$ (resp. $R \rightsquigarrow_{\mathbf{S}} \mathit{Sat}(M)$) to express that the solver $\mathbf{S}$ returns $\mathit{Unsat}(U)$ (resp. $\mathit{Sat}(M)$) when launched on a set of ground formulas R.

3.2 Deduction Rules for First-Order Formulas with Triggers

The solver $\mathcal{L}\mathit{ift}(\mathbf{S})$ takes a set of formulas with triggers T and a set of ground formulas S as input and decides whether S is satisfiable modulo T. It is constructed on top of a solver for ground formulas $\mathbf{S}$ and works on a set of ground formulas R that is augmented incrementally. While the solver $\mathbf{S}$ returns a model M of R, new facts are deduced from M and added to R.

The set R initially contains the formulas from the input S as well as those from the theory T where literals, quantified formulas, and formulas with triggers or witnesses are replaced by fresh atoms. The atom replacing a formula F is written $\boxed{F}$ and is called a *protected formula.*

Definition 3.3. *We say that a model M* produces *a pair F, t of a formula F and a term t if either F is the atom $\top$ and there is a literal L in M from S such that $t \in \mathcal{T}(L)$, F is a protected witness $\boxed{\langle s \rangle\ G} \in M$ and $t \in \mathcal{T}(s)$, or F a protected literal $\boxed{L} \in M$ and $t \in \mathcal{T}(L)$. We write it $M \uparrow F, t$.*

The following deduction rules are used to retrieve information from the protected formulas of a model M:

$$\textsc{Pos Unfold}\ \frac{\boxed{\langle t \rangle\ F} \in M}{\boxed{\langle t \rangle\ F} \to F} \qquad \textsc{Lit Unfold}\ \frac{\boxed{L} \in M}{\boxed{L} \to L}$$

$$\textsc{Neg Unfold}\ \frac{\boxed{[t]\ F} \in M \qquad M \uparrow G, t' \qquad M \cup \{t \not\approx t'\} \rightsquigarrow_{\mathbf{S}} \mathit{Unsat}(U \cup \{t \not\approx t'\})}{\boxed{[t]\ F} \wedge G \wedge U \to F}$$

$$\textsc{Inst}\ \frac{\boxed{\forall x.\ F} \in M \qquad M \uparrow G, t \qquad M \cup \{\neg F[x \leftarrow t]\} \rightsquigarrow_{\mathbf{S}} \mathit{Sat}(M')}{\boxed{\forall x.\ F} \wedge G \to F[x \leftarrow t]}$$

Rule Inst adds to R an instantiation of a universal formula with a known term. It is restricted by the premise $M \cup \{\neg F[x \leftarrow t]\} \rightsquigarrow_{\mathbf{S}} \mathit{Sat}(M')$ so that it does not instantiate a quantified formula if the result of the instantiation is already known. Rule Pos Unfold (resp. Lit Unfold) unfolds formulas with witnesses (resp. literals). Rule Neg Unfold removes a trigger when it is equal to a known term. Note that every deduction rule returns an implication: in a model where, say, $\langle t \rangle\ F$ is not true, F does not need to be true either.

The solver for FOL*-formulas $\mathcal{L}\mathit{ift}(\mathbf{S})$ returns *Unsat* on R, as soon as $\mathbf{S}$ returns *Unsat* on the current set of formulas. It returns *Sat* on R if the ground solver $\mathbf{S}$ returns a model M from which nothing new can be deduced by the above deduction rules.

Here is an example of execution of the solver $\mathcal{L}\mathit{ift}(\mathbf{S})$ on the set of ground formulas S modulo the theory T:

$$S = \{f(0) \approx 0,\ f(1) \not\approx 1\}$$

$$T = \{\forall x.[f(x+1)]\ f(x+1) \approx f(x) + 1\}$$

Let us show how the solver $\mathcal{L}\mathit{ift}(\mathbf{S})$ can deduce that

$$R_0 = \left\{ \begin{array}{l} f(0) \approx 0,\ f(1) \not\approx 1, \\ \boxed{\forall x.[f(x+1)]\ f(x+1) \approx f(x) + 1} \end{array} \right\}$$

is unsatisfiable.

1. The ground solver returns the only possible model M_0 of R_0, namely R_0 itself. Since $f(0) \approx 0 \in M_0$, M_0 produces the pair $\top, 0$. As a consequence, the rule Inst can instantiate x with 0 in the universal formula:

$$R_1 = R_0 \cup \left\{ \begin{array}{c} \boxed{\forall x.\ [f(x+1)]\ f(x+1) \approx f(x) + 1} \wedge \top \to \\ \boxed{[f(0+1)]\ f(0+1) \approx f(0) + 1} \end{array} \right\}$$

2. The solver returns the model $M_1 = M_0 \cup \{\boxed{[f(0+1)]\ f(0+1) \approx f(0)+1}\}$ of R_1. Since $f(1) \not\approx 1 \in M_1$, M_1 produces the pair $\top, f(1)$. Based on results from the theory of arithmetics, the ground solver can deduce that $f(0+1) \not\approx f(1)$ is unsatisfiable. Thus the rule Neg Unfold can add another formula to R_1:

$$R_2 = R_1 \cup \left\{ \begin{array}{c} \boxed{[f(0+1)]\ f(0+1) \approx f(0)+1} \wedge \top \rightarrow \\ \boxed{f(0+1) \approx f(0)+1} \end{array} \right\}$$

3. The ground solver returns the model $M_2 = M_1 \cup \{\boxed{f(0+1) \approx f(0)+1}\}$ of R_2. With the rule Lit Unfold, we can now unfold the protected literal $\boxed{f(0+1) \approx f(0)+1}$:

$$R_3 = R_2 \cup \{\boxed{f(0+1) \approx f(0)+1} \rightarrow f(0+1) \approx f(0)+1\}$$

4. Any model of R_3 contains $f(0+1) \approx f(0)+1$, $f(0) \approx 0$ and $f(1) \not\approx 1$. The ground solver returns $Unsat(\_)$ on R_3. As expected, the initial set S is reported to be unsatisfiable modulo T.

3.3 Properties

In this section, we prove that our solver is sound and complete on a particular class of axiomatics. In the following section, we demonstrate on an example how our framework can be used to check that a given axiomatics is in this class.

Completeness We say that a set of formulas with triggers T is *complete* if, for every finite set of literals G, $[\![G \cup T]\!]^-$ and $G \cup T$, triggers being ignored, are equisatisfiable in FOL.

Termination We say that a set of formulas with triggers T is *terminating* if, from every finite set of literals G, there can only be a finite number of instances of formulas of T. In our framework, we enforce three rules to enable reasoning about termination:

- instantiation is always done with known terms
- new known terms cannot be deduced if they are protected by a trigger
- an instance of a formula F with a term t is not generated if an instance of F has already been generated with t' equal to t.

Our solver is sound and complete if it works modulo a complete and terminating theory T:

Theorem 3.1. *If $\mathcal{L}ift(\mathbf{S})$ returns* Unsat *on a set of ground formulas S modulo a theory T then $S \cup T$, triggers being ignored, is unsatisfiable in* FOL.

Proof. Let M be a model returned by $\mathbf{S}$ and S be the set of terms contained in literals of M. It is straightforward to check that, for every deduction rule with conclusion F, $known(S) \vdash [\![F]\!]^-$. As a consequence, if the solver returns *Unsat* on G modulo T, then $[\![S \cup T]\!]^- \cup \forall x.known(x) \vdash \bot$. □

Theorem 3.2. *If $\mathcal{L}ift(\mathbf{S})$ returns* Sat *on a set of ground formulas S modulo a complete theory T then $S \cup T$, triggers being ignored, is satisfiable in* FOL.

Proof. If $S \supseteq R$ is saturated by the deduction rules and M is a model of S produced by $\mathbf{S}$, then we define M' as the set containing every literal in M and $known(t)$ for every term t such that $M \uparrow (\_, t)$. We then extend M' with $\neg known(t)$ for each t such that $M' \nvdash known(t)$. We show that M' is a model of $[\![M]\!]^-$.

We define a partial well-founded order on FOL*-formulas in Skolem negative normal form. A formula F_1 is smaller than a formula F_2 if and only if there is a ground substitution σ and a proper sub-formula F_2' of F_2 such that $F_2'\sigma = F_1$.

Lemma 3.1. *Let F be a* FOL**-formula such that $M \vdash_o F$. Assume that, for all formulas $F' \in M$ at least as small as F, $M' \vdash [\![F']\!]^-$. We show that, if $M \vdash_o F$ then $M' \vdash [\![F]\!]^-$.*

Proof. By definition of $\vdash_o$, there is a subset S_F of M such that $S_F \vdash_o F$ and every element of S_F is either a literal or a positive sub-formula of F. By hypothesis, $M' \vdash [\![S_F]\!]^-$. As a consequence, $M' \vdash [\![F]\!]^-$. □

We show by well-founded induction over the formulas $F \in M$ that $M' \vdash [\![F]\!]^-$.

- If F is a literal, by definition of M', $M' \vdash L$.
- If F is a protected literal $\boxed{L}$, by definition of M', $M' \vdash known(t)$ for every $t \in \mathcal{T}(L)$. Since S is saturated, $M \vdash_o L$ and $M' \vdash L$. Thus, $M' \vdash [\![F]\!]^-$.
- If F is a protected formula $\boxed{\langle t \rangle\ F'}$, by definition of M', $M' \vdash known(t')$ for every $t' \in \mathcal{T}(t)$. Since S is saturated, $M \vdash_o F \to F'$ and $M \vdash_o F'$. Since F' is strictly smaller than F, by induction hypothesis, we apply Lemma 3.1 and deduce $M' \vdash [\![F']\!]^-$. Thus, $M' \vdash [\![F]\!]^-$.
- If F is a protected formula $\boxed{[t]\ F'}$, either $M' \vdash \neg known(t)$ and, by definition of $[\![\,]\!]^-$, $M' \vdash [\![F]\!]^-$ or there is a pair G, t' such that $M \uparrow G, t'$ and $M' \vdash t \approx t'$. By definition of M', if $M' \vdash t \approx t'$ then $M \vdash_o t \approx t'$. By saturation of S, there is a subset U of M such that $M \vdash_o F \wedge G \wedge U \to F'$. Thus, $M \vdash_o F'$. Since F' is strictly smaller than F, by induction hypothesis, we can apply Lemma 3.1 and deduce $M' \vdash [\![F']\!]^-$. We obtain $M' \vdash [\![F]\!]^-$.
- If F is a protected universally quantified formula $\boxed{\forall(x).\ F'}$ and t a term, either $M' \vdash \neg known(t)$ and, by definition of $[\![\,]\!]^-$, $M' \vdash known(t) \to [\![F']\!]^-[x \leftarrow t]$ or $M' \vdash known(t)$ and there is a pair G, t' such that $M \uparrow G, t'$ and $M' \vdash t \approx t'$. Since S is saturated, $M \vdash_o F \wedge G \to F'[x \leftarrow t']$ and $M \vdash_o F'[x \leftarrow t']$. Since $F'[x \leftarrow t']$ is strictly smaller than F, by induction hypothesis, we can apply Lemma 3.1 and deduce $M' \vdash [\![F'[x \leftarrow t']]\!]^-$. Since $M' \vdash t \approx t'$, we also have $M' \vdash [\![F'[x \leftarrow t]]\!]^-$. As a consequence, $M' \vdash known(t) \to [\![F']\!]^-[x \leftarrow t]$. Since t can be any term, $M' \vdash [\![F]\!]^-$.

□

Theorem 3.3. *If the theory T is terminating, then the solver $\mathcal{L}ift(\mathbf{S})$ will terminate on any set of ground literal S.*

Proof. We prove that Lit Unfold, Pos Unfold, and Neg Unfold cannot be applied an infinite number of times on a finite set R. Lit Unfold (resp. Pos Unfold) can be applied once per ground literal (resp. ground formula with a witness) that appears in R and Neg Unfold can be applied once per pair of a ground formula with a trigger that appears in R and a term produced by a ground sub-formula of R. □

4 Completeness and Termination of a theory

Within our framework, we can reason about a theory T written as a set of formulas with triggers and demonstrate that it has the requested properties for our solver $\mathcal{L}ift(\mathbf{S})$ to be sound and complete. This section demonstrates how it can be done on an axiomatization of the non-extensional theory of arrays and a more complex axiomatization of reachability in finite acyclic imperative lists.

4.1 Non-Extensional Theory of Arrays

We show that Greg Nelson's proof of completeness for his decision procedure for arrays [13] can be turned into a proof of completeness of our solver on an axiomatization with carefully chosen triggers. For terms a, x and v, we write $access(a, x)$ the *access* in the array a at the index x and $update(a, x, v)$ the *update* of the array a by the element v at the index x. The following set of first-order formulas T is an axiomatization of the classical theory from McCarthy:

$$\forall a, x, v.\ [update(a, x, v)]\ access(update(a, x, v), x) \approx v \tag{1}$$

$$\forall a, x_1, x_2, v.\ [access(update(a, x_1, v), x_2)]\ x_1 \not\approx x_2 \rightarrow access(update(a, x_1, v), x_2) \approx access(a, x_2) \tag{2}$$

$$\forall a, x_1, x_2, v.\ [access(a, x_2)]\ [update(a, x_1, v)]\ x_1 \not\approx x_2 \rightarrow access(update(a, x_1, v), x_2) \approx access(a, x_2) \tag{3}$$

Note that (2) and (3) are in fact duplications of the same first order formula with different triggers[2]. Both of them are needed for completeness. For example, without (2) (resp. (3)), the set of formulas $\{y \not\approx x,\ access(update(a, y, v_1), x) \not\approx access(update(a, y, v_2), x)\}$ (resp. the set of formulas $\{y \not\approx x,\ access(a_1, x) \not\approx access(a_2, x),\ update(a_1, y, v) = update(a_2, y, v)\}$) cannot be proven unsatisfiable.

We prove that this axiomatics is complete and terminating.

Termination: If G is a set of ground literals, there can only be a finite number of instances from G and T. From (1), at most one term $access(update(a, x, v), x)$ can be created per *update* term $update(a, x, v)$ of G. No new *update* term can be created, so there will be only one instantiation of (1) per *update* term of G. Equations (2) and (3) can create at most one *access* term per couple comprising an index term (sub-term of an *access* term at the rightmost position) and an

[2]Most provers have a dedicated syntax for using several triggers for the same axiom.

update term. We deduce that at most one term per couple comprising the equality classes of an index term and an *update* term of G can be deduced.

Completeness: The set of formulas T gives a complete axiomatics. We prove that for every set of ground formulas G such that $[\![G \cup T]\!]^-$ is satisfiable, $[\![G \cup T]\!]^- \cup \forall t.\mathit{known}(t)$ is also satisfiable. Since assuming $\mathit{known}(t)$ for every term t removes triggers and witnesses, this shows that $G \cup T$ is satisfiable, triggers being ignored.

The proof is similar to the proof of Greg Nelson's decision procedure for arrays [13]. We first define the set of every array term a' such that $\mathit{access}(a', x)$ is equated to a given term $\mathit{access}(a, x)$ by (2) or (3):

Definition 4.1. *For a set of formulas S and two terms a and x known in S, we define the set $S_{a,x}$ to be the smallest set of terms containing a and closed by*

(i) if $a' \in S_{a,x}$ then every known term $\mathit{update}(a', y, v)$ such that $S \nvdash y \approx x$ is in $S_{a,x}$ and

(ii) for every term $\mathit{update}(a', y, \_) \in S_{a,x}$, if $S \nvdash y \approx x$ then a' is in $S_{a,x}$.

We now prove that, for every *access* or *update* term t, if S is a satisfiable set of ground formulas saturated by T then it can be extended to another satisfiable set S' saturated by T that contains $t = t$. Since, by definition of $[\![\,]\!]^-$, $[\![t = t]\!]^-$ is equivalent to $\bigwedge_{t' \in \mathcal{T}(t)} \mathit{known}(t)$, this is enough to have the completeness of T.

This proof is an induction over the size of t. We assume that every sub-term of t has already been added. If $\mathit{known}(t)$ is already implied by $[\![S]\!]^-$, then we are done. If t is neither an *access* nor an *update* term, then assuming the presence of t does not allow any new deduction.

Assume t is an *update* term $\mathit{update}(a, x, v)$. With the presence of t, (1) deduces the literal $\mathit{access}(t, x) \approx v$. This cannot lead to an inconsistency since nothing can be known about t otherwise t would be known in $[\![S]\!]^-$. Equations (2) and (3) deduce $\mathit{access}(t, y) = \mathit{access}(a', y)$ for all terms a' and y such that $[\![S]\!]^- \vdash \mathit{known}(\mathit{access}(a', y))$ and $t \in S_{a',y}$. Like the first one, this deductions cannot cause an inconsistency. The new set S' obtained by adding these literals to S is saturated by T. Indeed, if it is not, one of the formulas of T can deduce something that is not in S'. It cannot be (1) since we have applied it to the only new *update* term of S'. If it is (2) or (3) then it comes from a term $\mathit{access}(a', y) \in S'$ and $S'_{t,y} = S'_{a',y}$. By construction of S', the result is in S'.

Assume t is an *access* term $\mathit{access}(a, x)$. With the presence of t, (2) and (3) deduce $t = \mathit{access}(a', x)$ for every $a' \in S_{a,x}$. This deduction cannot cause an inconsistency. Indeed, nothing can be known about $\mathit{access}(a', x)$ otherwise t would have been known in S by (2) and (3). The new set S' obtained by adding these literals to S is saturated by T. Indeed, if it is not, one of the formulas of R can deduce something that is not in S'. It cannot be (1) since there is no new *update* term in S'. If it is (2) or (3) then it comes from a term $\mathit{access}(a', y) \in S'$ and $S'_{a,y} = S'_{a',y}$. By construction of S', the result is in S'.

4.2 Axiomatics for Reachability in Finite Acyclic Imperative Lists

We define an axiomatics for reachability in finite acyclic imperative lists. It is inspired by the axiomatics in [11]. Disjoint parts of memory are represented as functional arrays. Each array maps indexes to options that represent next indexes in a chained data structure. Lists can be modified by updating the corresponding array.

We first need an axiomatization for options. We use three symbols: the two constructors *None* and *Some* and a predicate symbol, *option* that allows declaring that a term is an option. To allow *option* to appear in triggers, we use a fresh constant $\top$ and define $option(v)$ to be syntactic sugar for $option(v) \approx \top$. We do not assume $\top$ to be part of any theory. We introduce the set R_{option} of axioms:

$$\forall v.\ [option(v)]\ option(v) \rightarrow (v \approx None \vee \exists x.\ v \approx Some\ x) \tag{4}$$

$$\forall x.\ [Some\ x]\ Some\ x \not\approx None \tag{5}$$

$$\forall x_1, x_2.\ [Some\ x_1,\ Some\ x_2]\ Some\ x_1 \approx Some\ x_2 \rightarrow x_1 \approx x_2 \tag{6}$$

If G is a set of ground terms, there can only be a finite number of applications of the rule Inst from $G \cup R_{option}$. Indeed, the set of formulas R_{option} creates at most a Skolem constant x per known term $option(v)$ by (4).

We can now design an axiomatics for reachability in finite acyclic imperative lists. In addition to symbols for options and arrays, we introduce a predicate symbol, *acyclic*, used to specify that a given list is finite and acyclic. Equations from (1) to (6) are used to handle arrays and options. Fig. 3 gives the new axioms for finite acyclic lists.

Equation (7) states that elements of arrays are options. Equations (8) and (9) are applications of the reflexivity rule for reachability. For the sake of simplicity, we do not rewrite the same formula when it has several triggers but use pipes instead. Equations (10), (11) and (12) express that, on acyclic lists, reachability is a partial order. Equations (13), (14), (15) and (16) relate reachability to the next element in the chained data-structure. All the remaining formulas are used to describe how reachability properties on lists are modified when the list is updated. We call R the set of formulas involving these eighteen axioms plus the six axioms for arrays and options.

Definition 4.2. *We say that a term a is an* array term *in a set of literals S if it appears as the first argument of a reach term of S. In the same way, x is an* index term *in S if it appears as the second argument of a reach term or as the only argument of a Some term of S.*

We first need to show that, if G is a finite set of ground formulas, the solver terminates on $G \cup R$. Take a finite set of ground formulas containing G and saturated through (1), (2) and (3). Equation (7) can then be used to deduce at most an option term per access term. From this new finite set, formula (4) can deduce a finite number of new terms. Equations from (1) to (7) can no longer create any term. The remaining formulas can only deduce reach terms. It will deduce at most a reach term per triplet of an array term and two index terms.

Now, we would like to prove that R is complete on finite acyclic lists. We consider a finite set of ground formulas G. It can be written as a disjunction of conjunctions of literals. Let C be a conjunction of G. We assume that, if an

$$\forall a, x.\ [access(a, x)]\ option(access(a, x)) \tag{7}$$
$$\forall a, x.\ [access(a, x) \mid update(a, x, None)]\ reach(a, x, x) \tag{8}$$
$$\forall a, x_1, x_2.\ [update(a, x_1, Some\ x_2) \mid reach(a, x_1, x_2)]$$
$$reach(a, x_1, x_1) \wedge reach(a, x_2, x_2) \tag{9}$$
$$\forall a, x_1, x_2.\ [acyclic(a),\ reach(a, x_1, x_2),\ reach(a, x_2, x_1)]$$
$$acyclic(a) \to reach(a, x_1, x_2) \to reach(a, x_2, x_1) \to x_1 \approx x_2 \tag{10}$$
$$\forall a, x_1, x_2, x_3.\ [reach(a, x_1, x_2),\ reach(a, x_2, x_3)]$$
$$reach(a, x_1, x_2) \to reach(a, x_2, x_3) \to reach(a, x_1, x_3) \tag{11}$$
$$\forall a, x_1, x_2, x_3.\ [reach(a, x_1, x_2),\ reach(a, x_1, x_3)]\ reach(a, x_1, x_2) \to$$
$$reach(a, x_1, x_3) \to reach(a, x_2, x_3) \vee reach(a, x_3, x_2) \tag{12}$$
$$\forall a, x_1, x_2.\ [access(a, x_1),\ Some\ x_2]\ access(a, x_1) \approx Some\ x_2 \to reach(a, x_1, x_2) \tag{13}$$
$$\forall a, x_1, x_2.\ [access(a, x_1), reach(a, x_1, x_2)]$$
$$access(a, x_1) \approx None \to reach(a, x_1, x_2) \to x_1 \approx x_2 \tag{14}$$
$$\forall a, x_1, x_2.\ [access(a, x_1),\ reach(a, x_1, x_2),\ Some\ x_3]$$
$$access(a, x_1) \approx Some\ x_3 \to reach(a, x_1, x_2) \to x_1 \approx x_2 \vee reach(a, x_3, x_2) \tag{15}$$
$$\forall a, x_1, x_2.\ [acyclic(a),\ access(a, x_1),\ reach(a, x_2, x_1),\ Some\ x_3]$$
$$acyclic(a) \to access(a, x_1) \approx Some\ x_3 \to reach(a, x_2, x_1) \to x_2 \not\approx x_3 \tag{16}$$
$$\forall a, e_1, e_2.\ [acyclic(update(a, e_1, Some\ e_2))]$$
$$acyclic(update(a, e_1, Some\ e_2)) \to \neg reach(a, e_2, e_1) \tag{17}$$
$$\forall a, e, x_1, x_2, k.\ [reach(update(a, e, k), x_1, x_2)]$$
$$reach(update(a, e, k), x_1, x_2) \to reach(a, x_1, e) \vee reach(a, x_1, x_2) \tag{18}$$
$$\forall a, e_1, e_2, x_1, x_2.\ [reach(update(a, e_1, Some\ e_2), x_1, x_2)]$$
$$reach(update(a, e_1, Some\ e_2), x_1, x_2) \to reach(a, x_1, e_1) \to$$
$$e_1 \approx x_2 \vee (reach(a, x_1, x_2) \wedge \neg reach(a, e_1, x_2)) \vee reach(a, e_2, x_2) \tag{19}$$
$$\forall a, e, x_1, x_2.\ [reach(update(a, e, None), x_1, x_2)]$$
$$reach(update(a, e, None), x_1, x_2) \to reach(a, x_1, e) \to$$
$$e \approx x_2 \vee (reach(a, x_1, x_2) \wedge \neg reach(a, e, x_2)) \tag{20}$$
$$\forall a, e, x_1, x_2, k.\ [update(a, e, k),\ reach(a, x_1, x_2)]\ reach(a, x_1, x_2) \to$$
$$(reach(a, e, x_2) \wedge reach(a, x_1, e)) \vee reach(update(a, e, k), x_1, x_2) \tag{21}$$
$$\forall a, e, x, k.\ [update(a, e, k),\ reach(a, x, e)]$$
$$reach(a, x, e) \to reach(update(a, e, k), x, e) \tag{22}$$
$$\forall a, e_1, e_2, x_1, x_2.\ [update(a, e_1, Some\ e_2),\ reach(a, x_1, x_2),\ reach(a, e_2, x_2)]$$
$$reach(a, x_1, x_2) \to reach(a, e_2, x_2) \to reach(update(a, e_1, Some\ e_2), x_1, x_2) \tag{23}$$
$$\forall a, e_1, e_2, x_1, x_2.\ [update(a, e_1, Some\ e_2),\ reach(a, x_1, e_1),\ reach(a, e_2, x_2)]$$
$$reach(a, x_1, e_1) \to reach(a, e_2, x_2) \to reach(update(a, e_1, Some\ e_2), x_1, x_2) \tag{24}$$

Figure 3: Axiomatization of finite acyclic lists

array term t appears in C, then $C \vdash \mathit{acyclic}(t)$. We now prove that, if $[\![R \cup C]\!]^-$ is satisfiable, then C has a model. That is to say, there is a collection of finite acyclic lists on which every assumption of C holds. Note that every formula F of R is written $\forall \overline{x}.\ [t_1 \dots t_n]\ L_1 \to \dots \to L_m \to C_1 \vee \dots \vee C_k$ where C_i is a conjunction of literals and L_i a literal. We define the premises of F to be the set $\{\mathit{known}(\overline{x}), \mathit{known}(t_1), \dots, \mathit{known}(t_n), L_1, \dots, L_m\}$ and its conclusion to be the disjunction $C_1 \vee \dots \vee C_k$. We say that a set of ground literals S is saturated through R if, whenever the premises of a formula $F \in R$ are true in S for some ground substitution σ, there is at least a conjunction C in the conclusion of F such that $S \vdash C\sigma$.

We consider a set S of literals saturated through R. Since no formula of R can create new array terms, we can assume that, if an array term t appears in S, then $S \vdash \mathit{acyclic}(t)$. Since there can only be a finite number of instantiations from $R \cup G$, we can also assume S to be finite. In the rest of the section, we create a model for S by adding a direct link between reachable indexes.

Definition 4.3. *We define the set of indexes reachable in an array a from an index x as $\mathit{reach}_{a,x} \triangleq \{y \mid S \vdash \mathit{reach}(a,x,y)\}$. By (12) and (10), the minimal option reachable from x in a can be defined uniquely (a is acyclic) as:*

$$\mathit{next}_{a,x} \triangleq \left\{ \begin{array}{ll} \mathit{None} & \mathit{if}\ \mathit{reach}_{a,x} = \{x\} \\ \mathit{Some}\ y & y \in \mathit{reach}_{a,x},\ x \not\approx y,\ \forall z \in \mathit{reach}_{a,x}.\ z \approx x \vee z \in \mathit{reach}_{a,y} \end{array} \right\}$$

We define a set S' containing S and the literal $\mathit{access}(a,x) \approx \mathit{next}_{a,x}$ for every array term a and every index term x in S. We prove, by case analysis, that the rules for reachability are sufficient to deduce from S, for every $a' \in S_{a,x}$[3], that $\mathit{next}_{a,x} \approx \mathit{next}_{a',x}$.

The set $S_{a,x}$ is constructed iteratively by including array terms b such that either a can be written $b[e \leftarrow k]$ and $S \nvdash e \approx x$ or b can be written $a[e \leftarrow k]$ and $S \nvdash e \approx x$. We prove that, in the first case, $S \vdash \mathit{next}_{a,x} \approx \mathit{next}_{b,x}$. The proof of the second case is omitted.

If $\mathit{reach}_{a,x} = \{x\}$ then, by construction of *next*, $\mathit{next}_{a,x} = \mathit{None}$. We show that we have $\mathit{reach}_{b,x} = \{x\}$. By (22), we cannot have $S \vdash \mathit{reach}(b,x,e)$ or e would be in $\mathit{reach}_{a,x}$. As a consequence, by (21), we cannot have $S \vdash \mathit{reach}(b,x,y)$ for any y such that $S \nvdash y \approx x$. We deduce that $\mathit{next}_{b,x} = \mathit{None}$.

Otherwise, $\mathit{next}_{a,x} = \mathit{Some}\ y$ with $S \nvdash y \approx x$. We show that $S \vdash \mathit{reach}(b,x,y)$ and, with $\mathit{next}_{b,x} = \mathit{Some}\ z$, $S \vdash \mathit{reach}(b,y,z)$. This obviously implies that $z \approx y$. First, we need an intermediate lemma:

Lemma 4.1. *For any index terms z_1 and z_2, if we have $S \vdash \mathit{reach}(a,z_1,z_2)$ then either $S \vdash \mathit{reach}(b,z_1,z_2)$ or $S \vdash \mathit{reach}(b,z_1,e)$, $S \vdash k \approx \mathit{Some}\ e_2$ and $S \vdash \mathit{reach}(b,e_2,z)$.*

Proof. By (18), we have either $S \vdash \mathit{reach}(b,z_1,z_2)$ or $S \vdash \mathit{reach}(b,z_1,e)$. If we have $S \nvdash \mathit{reach}(b,z_1,z_2)$, by (20), $S \vdash k \approx \mathit{Some}\ e_2$. By (19), we then have $S \vdash \mathit{reach}(b,e_2,z)$. □

If $k = \mathit{None}$, by Lemma 4.1 we have $S \vdash \mathit{reach}(b,x,y)$. Now let us show that, if $\mathit{next}_{b,x} = \mathit{Some}\ z$, $S \vdash \mathit{reach}(a,y,z)$. Since $S \vdash \mathit{reach}(b,x,z)$ and

[3]Recall that $S_{a,x}$ is the set of every array term a' such that $\mathit{access}(a,x) \approx \mathit{access}(a',x)$ can be deduced from S by (2) and (3).

$S \nvdash z \approx x$, by (21), we have either $S \vdash \mathit{reach}(a, x, z)$, or $S \vdash \mathit{reach}(b, x, e)$ and $S \vdash \mathit{reach}(b, e, z)$. In the last case, by definition of *next*, $S \vdash z \approx e$. By (22), if $S \vdash z \approx e$ then $S \vdash \mathit{reach}(a, x, z)$. Thus, $S \vdash \mathit{reach}(a, x, z)$ and, by definition of *next*, $S \vdash \mathit{reach}(a, y, z)$. By Lemma 4.1, we get $S \vdash \mathit{reach}(b, y, z)$.

If $k = \mathit{Some}\ e_2$, we first show that $S \vdash \mathit{reach}(b, x, y)$. By Lemma 4.1, we have either $S \vdash \mathit{reach}(b, x, y)$ or $S \vdash \mathit{reach}(b, x, e)$ and $S \vdash \mathit{reach}(b, e_2, y)$. In the second case, by (22), we have $S \vdash \mathit{reach}(a, x, e)$. By definition of *next*, $S \vdash \mathit{reach}(a, y, e)$. From Lemma 4.1, we deduce either $S \vdash \mathit{reach}(b, y, e)$ which, by transitivity, gives $S \vdash \mathit{reach}(b, e_2, e)$. This contradicts (17). As a consequence, $S \vdash \mathit{reach}(b, x, y)$. Now let us show that, if $\mathit{next}_{b,x} = \mathit{Some}\ z$, $S \vdash \mathit{reach}(a, y, z)$. Since $S \vdash \mathit{reach}(b, x, z)$, by (21), either $S \vdash \mathit{reach}(a, x, z)$ or $S \vdash \mathit{reach}(b, x, e)$ and $S \vdash \mathit{reach}(b, e, z)$. In the second case, by definition of *next*, $S \vdash z \approx e$ and we have $S \vdash \mathit{reach}(a, x, z)$ by (22). Thus we have $S \vdash \mathit{reach}(a, x, z)$. By definition of *next*, we have $S \vdash \mathit{reach}(a, y, z)$ and then, by Lemma 4.1, either $S \vdash \mathit{reach}(b, y, z)$ or $S \vdash \mathit{reach}(b, y, e)$ and $S \vdash \mathit{reach}(b, e_2, z)$. The second case is not possible by minimality of z.

We have proven that, for every $a' \in S_{a,x}$, $S \vdash \mathit{next}_{a,x} \approx \mathit{next}_{a',x}$. What is more, by (7), if the term $\mathit{access}(a, x)$ appears in S then it is either equal to *None* or *Some* y and, by (14) or (15), $S \vdash \mathit{access}(a, x) \approx \mathit{next}_{a,x}$. As a consequence, assuming $\mathit{access}(a, x) \approx \mathit{next}_{a,x}$ cannot introduce any contradiction and S' is still satisfiable. What is more, by construction of S', exactly every positive reach literal of S can be deduced using (13) and (11) from array terms of S' and all the others must be false by (14). As a consequence, the model for the array terms of S' is a model of S'.

5 Conclusion

We have presented a new first-order logic with a syntax for triggers and given it a clear semantics. We have shown that a solver accepting a theory written as a set of formulas with triggers T can be implemented on top of an off-the-shelf SMT solver, and we have identified properties requested from T for the resulting solver to be sound and complete on ground formulas. Finally, we have demonstrated, on the non-extensional theory of arrays, that our framework can be used to prove that a theory expressed as a set of first-order formulas with triggers indeed has the requested properties.

In future work, we would like to integrate our technique of quantifier handling directly inside a DPLL(T)-based solver. Once a solver implementing our semantics exists, a static analysis could be done to detect too restrictive or too permissive axiomatizations, and matching loops. We believe that such an analysis will help theory designers avoid common pitfalls when writing axiomatizations.

References

[1] Leo Bachmair, Harald Ganzinger, Christopher Lynch, and Wayne Snyder. Basic paramodulation. Research Report MPI-I-93-236, Max-Planck-Institut für Informatik, Im Stadtwald, D-66123 Saarbrücken, Germany,

September 1993. Revised version in Information and Computation 121(2), pp. 172–192, 1995.

[2] Clark Barrett, Aaron Stump, and Cesare Tinelli. The SMT-LIB standard version 2.0. *Technical report, University of Iowa*, december 2010.

[3] François Bobot, Sylvain Conchon, Évelyne Contejean, and Stéphane Lescuyer. Implementing polymorphism in SMT solvers. In *SMT'08*, volume 367 of *ACM ICPS*, pages 1–5, 2008.

[4] L. de Moura and N. Bjørner. Engineering dpll (t)+ saturation. *Automated Reasoning*, pages 475–490, 2008.

[5] Leonardo de Moura and Nikolaj Bjørner. Efficient E-matching for SMT solvers. *CADE'07*, 2007.

[6] Leonardo de Moura and Nikolaj Bjørner. Z3: An efficient SMT solver. *TACAS*, 2008.

[7] David Detlefs, Greg Nelson, and James B. Saxe. Simplify: a theorem prover for program checking. *J. ACM*, 52(3):365–473, 2005.

[8] Y. Ge and L. De Moura. Complete instantiation for quantified formulas in satisfiabiliby modulo theories. In *Computer Aided Verification*, pages 306–320. Springer, 2009.

[9] Yelting Ge, Clark Barrett, and Cesare Tinelli. Solving quantified verification conditions using satisfiability modulo theories. *CADE*, 2007.

[10] Swen Jacobs and Viktor Kuncak. Towards complete reasoning about axiomatic specifications. In *Proceedings of VMCAI*, pages 278–293. Springer, 2011.

[11] S. Lahiri and S. Qadeer. Back to the future: revisiting precise program verification using smt solvers. In *ACM SIGPLAN Notices*, volume 43, pages 171–182. ACM, 2008.

[12] Michal Moskal. Programming with triggers. *Proceedings of the 7th International Workshop on Satisfiability Modulo Theories*, pages 20–29, 2009.

[13] Greg Nelson. Techniques for program verification. *Technical Report CSL81-10, Xerox Palo Alto Research Center*, 1981.

[14] P. Rümmer. E-matching with free variables. 2012.

Proof of completeness of BP

Soundness of **BP** an be checked straightforwardly. As a consequence, we focus in this section on the proof of refutational completeness of **BP**.

We assume given a reduction ordering $\prec$ which is total on ground terms. We identify a positive literal $s \approx t$ with the multi-set (of multi-sets) $\{\{s\},\{t\}\}$, and a negative literal $s \not\approx t$ with the multi-set $\{\{s,t\}\}$. This gives an ordering on literals $(\prec_{mul})_{mul}$ and on clauses $((\prec_{mul})_{mul})_{mul}$.

We say that a literal $L[s']_p$ is *order-reducible* (at position p) by an equation $s \approx t$, if $s' = s\rho$, $s\rho \succ t\rho$ and $L \succ s\rho \approx t\rho$.

A non-variable position p in a clause $C\sigma$ is called a *substitution position* in the closure C, σ if it can be written as $p = p'q$, where p' is a variable position in C.

We say that a ground closure $C \cdot \sigma$ is *reduced* with respect to a rewrite system R (or R-reduced) *at a position* p if $C\sigma$ is order-irreducible by R at or below p. The closure $C \cdot \sigma$ is simply called *reduced* with respect to R if it is reduced at all substitution positions.

We say that a clause $C \vee s \approx t$ is *reductive* for $s \approx t$ if $t \not\succeq s$ and $s \approx t$ is a strictly maximal literal in the clause.

Let N be a set of closures saturated by the system. For each ground instance C of a closure of N, we define two sets E_C and R_C of equations. To construct the sets associated with the clause C, we assume that $E_{C'}$ and $R_{C'}$ have been defined for all ground instances C' of N for which $C \succ C'$. We then define:

$$
\begin{aligned}
R_C &= \bigcup_{C \succ C'} E_{C'} \\
E_C &= \begin{cases} \{s \approx t\} & \text{If } C \text{ is a productive clause.} \\ \varnothing & \text{Otherwise.} \end{cases}
\end{aligned}
$$

where C is said to be *productive* if and only if:

- $C = D \vee s \approx t$ is a reduced ground instance of N with respect to R_C,
- C is false in R_C,
- C is reductive for $s \approx t$ and
- for all $L \in C$, L is order-irreducible by R_C.

We also define $R = \bigcup_C E_C$. By construction of R and R_C, we now have a few properties:

Lemma .1. *If a ground closure $C \cdot \sigma$ is R_C-reduced then it is R-reduced.*

Proof. If C is not reduced with respect to R, then there is some clause D which produces an equation $s \approx t$, and some literal L in C which is reducible at a substitution position by $s \approx t$ and such that $s \approx t \prec L$. Since $s \approx t$ is strictly maximal in D, clearly $D \prec C$, and C is not reduced with respect to R_C. □

Lemma .2. *If a ground clause C is true in R_C then it is also true in R and R_D for every clause $D \succ C$.*

Proof. If C contains a literal that is true in R_C then the result is obvious. Otherwise, C contains a disjunction $s \not\approx t$ such that $s \approx t$ is not true in R_C. By construction, the new rewrite rules of R and R_D involve terms that are strictly greater than s and t. As a consequence, $s \approx t$ is false both in R and R_D. □

Theorem .1. *If N is a saturated set of closures which does not contain the empty clause, R a rewrite system constructed from N according to the definition, and $C = \tilde{C}\rho\sigma$ is an R-reduced ground instance of a closure $\tilde{C} \cdot \rho$ in N, then:*

1. *If C is order-reducible by R_C, then C is true in R_C.*
2. *If C contains at least a negative equation, then C is true in R_C.*
3. *If C is false in R_C then C is a productive clause of the form $C = D \vee s \approx t$ (where $s \approx t$ is the equation produced) and D is false in R.*
4. *C is true in R and in R_D for every $D \succ C$.*

Proof. For any such clause C, we assume that the four properties are true for every clause strictly smaller than C and we show that they also hold for C:

1. If C is order-reducible by R_C, then C can be written $L[s] \vee C'$ and there is an equation $s \approx t \prec L[s]$ in R_C. By construction of R_C, $s \approx t$ comes from a productive ground instance $\tilde{D} \cdot \tau$ of a closure of N where $\tilde{D}$ can be written $\tilde{s} \approx \tilde{t} \vee \tilde{D}'$. Since C is R-reduced, the rewrite cannot appear at a substitution position. As a consequence, $\tilde{C}$ can be written $\tilde{L}[\tilde{s}'] \vee \tilde{C}'$ and we can apply a paramodulation step.

   We get a closure $\tilde{L}[\tilde{t}] \vee \tilde{C}' \vee \tilde{D}' \cdot \mu$, for a substitution μ more general than $\tau \oplus \rho\sigma$. By definition of saturation, $\tilde{L}[\tilde{t}] \vee \tilde{C}' \vee \tilde{D}' \cdot \mu$ is in N. Since $\tilde{D}, \tau$ is productive, $\tilde{D}, \tau$ is R_D-reduced. By Lemma .1, it is also R-reduced. Since both $\tilde{C} \cdot \rho\sigma$ and $\tilde{D} \cdot \tau$ are R-reduced, so is $\tilde{L}[\tilde{t}] \vee \tilde{C}' \vee \tilde{D}' \cdot \tau \oplus \rho\sigma$. Since D is reductive for $s \approx t$, $s \approx t$ is strictly greater than the literals of $D' = \tilde{D}'\tau$ and $s \succ t$. As a consequence, $L[t] \vee C' \vee D'$ is strictly smaller than C. The property (3) holds for $L[t] \vee C' \vee D'$ which is true in R_C.

   Since D' is false in R_D, D' is also false in R_C. As a consequence, $L[t] \vee C'$ is true in R_C and, since $s \approx t \in R_C$, so is C.

2. If C is order-reducible by R_C, then, by property (1), C is true in R_C. Assume C is order-irreducible by R_C. By hypothesis, C can be written $C' \vee s \not\approx s'$.

   Since $s \neq s'$ appears in C and is order-irreducible by R_C, it is also order-irreducible by R. As a consequence, if $s \neq s'$ then $s \not\approx s'$ is true in R and so is C.

   Otherwise, $\tilde{C}$ can be written $\tilde{C}' \vee \tilde{s} \not\approx \tilde{s}'$ where $\tilde{s}\rho$ and $\tilde{s}'\rho$ are unifiable. As a consequence, a reflexivity step can be applied to $\tilde{C}, \rho$. We get a closure $\tilde{C}', \mu$, for a substitution μ more general than $\rho\sigma$. By definition of saturation, $\tilde{C}' \cdot \mu \in N$. Since $\tilde{C} \cdot \rho\sigma$ is R-reduced, so is $\tilde{C}' \cdot \rho\sigma$. Since $C \succ C'$, the property (3) holds for C' which is true in R_C.

   Since $C' \subseteq C$, C is also true in R_C.

3. By property (2), if C is false in R_C then it contains no negative equation. Since C cannot be empty, let $s \approx t$, with $s \succeq t$, be one of the maximal equations of C. We assume that $s \succ t$ since otherwise, C would contain a tautology $s \approx s$ and therefore be true in any rewrite system.

   If $s \approx t$ is not strictly maximal in C then, since $\succ$ is total on ground clauses, $\tilde{C}$ can be written $\tilde{s} \approx \tilde{t} \vee \tilde{s}' \approx \tilde{t}' \vee \tilde{C}'$ with $(\tilde{s} \approx \tilde{t})\rho$ and $(\tilde{s}' \approx \tilde{t}')\rho$ unifiable. As a consequence, a factoring step can be applied. We get a closure $\tilde{C}' \vee x \approx y \cdot \mu$, for a substitution μ more general than $\rho\sigma$. By definition of saturation, $\tilde{C}' \vee x \approx y \cdot \mu \in N$. Since $(x \approx y)\rho\sigma$ is equal to $s \approx t$, $\tilde{C}' \vee x \approx y \cdot \rho\sigma$ is R-reduced. We also have $C \succ C' \vee s \approx t$. As a consequence, the property (3) holds for $C' \vee s \approx t$ which is true in R_C. Since $C' \vee s \approx t \subseteq C$, C is also true in R_C. This contradicts the hypothesis.

   Since $s \succ t$ and $s \approx t$ is strictly maximal in C, C is reductive for $s \approx t$. By property (1), C is also order-irreducible by R_C. What is more, by hypothesis, C is also R-reduced and false in R_C. Since $R_C \subseteq R$, C is R_C-reduced. As a consequence, C is a productive clause for R_C.

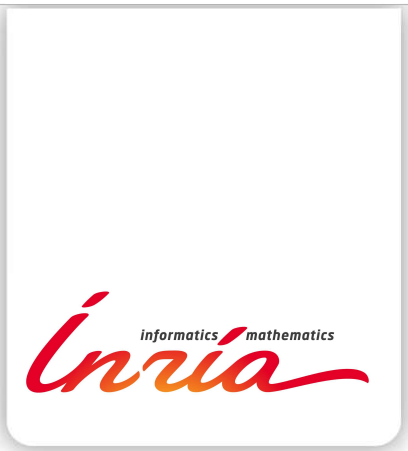
4. If C is true in R_C, by Lemma .2, it is also true in R and R_D. Otherwise, by property (3), C is a productive clause for R_C. By definition of R and R_D, C is true in R and in R_D for every $D \succ C$.

□

Corollary .1. *For every unsatisfiable set of clauses K, the empty clause can be deduced from the set of closures $\{C \cdot \varnothing \mid C \in K\}$ by paramodulation.*

Proof. If the empty clause cannot be deduced from $\{C \cdot \varnothing \mid C \in K\}$, then there is a saturated set of closures N including $\{C \cdot \varnothing \mid C \in K\}$ that does not contain the empty clause. Let R be the rewrite system computed from N. We show that R is a model of K.

For each ground instance $C\tau$ of K, we consider ρ such that, for every variable x of C, $x\rho$ is the normal form of $x\tau$. By construction, $C \cdot \rho$ is a ground instance of the closure $C \cdot \varnothing \in N$ that is reduced by R. As a consequence, by theorem .1, $C\rho$ is true in R. By definition of ρ, so is $C\tau$. As a consequence, R is a model of K and K is satisfiable. □



RESEARCH CENTRE
SACLAY – ÎLE-DE-FRANCE

Parc Orsay Université
4 rue Jacques Monod
91893 Orsay Cedex

Publisher
Inria
Domaine de Voluceau - Rocquencourt
BP 105 - 78153 Le Chesnay Cedex
inria.fr

ISSN 0249-6399