Reasoning with Triggers

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

Sylvain Conchon
LRI, Université Paris-Sud 11, CNRS, Orsay F-91405
INRIA Saclay-Île de France, ProVal, Orsay F-91893

Johannes Kanig
AdaCore, Paris F-75009

Andrei Paskevich
LRI, Université Paris-Sud 11, CNRS, Orsay F-91405
INRIA Saclay-Île de France, ProVal, Orsay F-91893

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 using the theory of arrays, how such proofs can be achieved in our formalism.

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 [6], CVC3 [9], Z3 [5] and Alt-Ergo [2], 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 [6, 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 [1] 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 [6, 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 [6, 13] for Simplify, in [4] 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 [3].

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

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

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

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 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 $\forall x\,.\,[f(x)]\ P(f(x)),\ \forall x\,.\,[f(g(x))]\ (P(f(g(x))) \to Q(x)) \vdash^\star Q(c)$, 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 $P(f(g(c))) \to Q(c)$.

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 $\mathit{known}(\omega),\ [\![R]\!]^- \vdash \bot$.

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

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 $M' \vdash \forall x.\mathit{known}(x)$. By immediate induction, $(\forall x.\mathit{known}(x)) \wedge F \vdash [\![F]\!]^-$. 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 proof, based on inference trees in a certain paramodulation calculus, can be found in the technical report [7].

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

3. The ground solver returns the model $M_2 = M_1 \cup \{\boxed{f(0+1) \approx f(0)+1}\}$ of R_2. The rule Lit Unfold 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.

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.

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.*

The proofs of these three theorems may be found in the technical report [7].

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.

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. Another example is given in the technical report [7]. 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[1]. 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 *access* 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 *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, $[\![G \cup T]\!]^- \cup \forall t.known(t)$ is also satisfiable. Since assuming $known(t)$ 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 $access(a', x)$ is equated to a given term $access(a, x)$ 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 $update(a', y, v)$ such that $S \vdash y \approx x$ is in $S_{a,x}$ and

(ii) for every term $update(a', y, \_) \in S_{a,x}$, if $S \vdash y \approx x$ 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 $\bigwedge_{t' \in \mathcal{T}(t)} known(t)$, 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 $known(t)$ 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.

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

Assume t is an *update* term $update(a, x, v)$. With the presence of t, (1) deduces the literal $access(t, x) \approx v$. 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 $access(t, y) = access(a', y)$ for all terms a' and y such that $[\![S]\!]^- \vdash known(access(a', y))$ 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 $access(a', y) \in S'$ and $S'_{t,y} = S'_{a',y}$. By construction of S', the result is in S'.

Assume t is an *access* term $access(a, x)$. With the presence of t, (2) and (3) deduce $t = access(a', x)$ for every $a' \in S_{a,x}$. This deduction cannot cause an inconsistency. Indeed, nothing can be known about $access(a', x)$ 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 $access(a', y) \in S'$ and $S'_{a,y} = S'_{a',y}$. By construction of S', the result is in 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] Clark Barrett, Aaron Stump, and Cesare Tinelli. The SMT-LIB standard version 2.0. *Technical report, University of Iowa*, december 2010.

[2] 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.

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

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

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

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

[7] Claire Dross, Sylvain Conchon, Johannes Kanig, and Andrei Paskevich. Reasoning with triggers. Research Report RR-7986, INRIA, June 2012.

[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.