Complete instantiation for quantified formulas in Satisfiability Modulo Theories

Yeting Ge[1] and Leonardo de Moura[2]

[1] Department of Computer Science, New York University, NY, NY 10012, USA
yeting@cs.nyu.edu

[2] Microsoft Research, One Microsoft Way, Redmond, WA 98052, USA
leonardo@microsoft.com

Abstract. Quantifier reasoning in Satisfiability Modulo Theories (SMT) is a long-standing challenge. The practical method employed in modern SMT solvers is to instantiate quantified formulas based on heuristics, which is not refutationally complete even for pure first-order logic. We present several decidable fragments of first order logic modulo theories. We show how to construct models for satisfiable formulas in these fragments. For richer undecidable fragments, we discuss conditions under which our procedure is refutationally complete. We also describe useful heuristics based on model checking for prioritizing or avoiding instantiations.

1 Introduction

Applications in software verification have benefited greatly from recent advances in automated reasoning. Applications in this field often require determining the satisfiability of first-order formulas with respect to some background theories. Satisfiability Modulo Theories (SMT) solvers have proven highly scalable, efficient and suitable for integrated theory reasoning. Most SMT solvers are restricted to ground formulas. However, for numerous applications in software verification, quantifiers are needed. For example, quantifiers are convenient for capturing frame conditions over loops, summarizing auxiliary invariants over heaps, and for supplying axioms of theories that are not already equipped with decision procedures for ground formulas.

Quantifier reasoning in SMT is a long-standing challenge. Because most quantified SMT formulas contain both interpreted and uninterpreted symbols, it is difficult to have a general decision procedure for quantifiers in SMT. For example, there is no sound and complete procedure for first-order logic formulas of linear arithmetic with uninterpreted function symbols [1]. Some SMT solvers [2, 3] integrate the superposition calculus with ground decision procedures. These solvers are refutationally complete for pure first-order logic with equality, but do not provide any guarantee when other interpreted symbols appear in quantified formulas. Several first-order calculi have been proposed based on the idea of theory resolution [4]. These calculi provide nice theoretical results, yet no efficient

implementations, because the computation of theory unifiers is too expensive or impossible for background theories of interest. In general, it is inefficient to use general first-order theorem prover to check the satisfiability of SMT formulas when the background theory does not have a finite axiomatization (e.g., arithmetic).

Most state-of-the-art SMT solvers with support for quantifiers use heuristic quantifier instantiation [5–7, 3] for incorporating quantifier reasoning with ground decision procedures. A well known heuristic instantiation-based approach is the E-matching algorithm introduced by the Simplify theorem prover [8]. Although heuristic instantiation is relatively effective for some software verification applications [9, 10], it suffers from several problems: it is not refutationally complete for first-order logic, hints (triggers) are usually required, it is sensitive to the syntactic structure of the formula, and it fails to prove formulas that can be easily discharged by saturation-based provers.

Instantiation-based approaches are attractive because SMT solvers have efficient ground decision procedures for many useful theories. For some fragments of first order logic modulo theories, we have complete decision procedures based on quantifier instantiation. We call this type of decision procedure complete instantiation.

In this paper, we investigate several decidable fragments of first-order logic modulo theories. The new fragments subsume well-known fragments such as the Bernays-Schönfinkel class, stratified vocabularies for many-sorted logic [11], and the Array Property Fragment [12]. We also consider richer fragments which are not decidable, and we discuss conditions under which our procedure is refutationally complete. The proposed decision procedures can be directly used to prove complex quantified array properties. Arrays are common in most programming languages and provide a natural model for memories. Decision procedures for array theories are of great interest for verification applications. Our approach is also suitable for formulas coming from verification of parameterized systems, and the axiomatization application specific theories (e.g., the axiomatization of the Spec# type system based on the theory of partial orders).

In software verification, models of satisfiable verification conditions are of great interest because they usually suggest potential errors. Therefore, we also show how to construct models for satisfiable quantified formulas in these fragments. On the implementation side, we describe useful heuristics based on model checking for prioritizing or avoiding instantiations.

The ground terms used for instantiating quantified formulas come from the least solution of systems of set constraints. We first consider a fragment in which quantified variables only occur as arguments of uninterpreted function (predicate) symbols. Then, we introduce more fragments, by relaxing some restrictions and by augmenting the system of constraints. We give examples to illustrate the usefulness of these fragments as well.

2 Background

We will assume the usual notions and terminology of first order logic and model theory. Let Σ be a *signature* consisting of a set of *function* and *predicate* symbols. Each function symbol f is associated with a non-negative integer, the *arity* of f, denoted by $\mathit{arity}(f)$. We call 0-arity function symbols *constant* symbols, and usually denote them by a, b, c and d. We use f, g, h to denote non-constant function symbols, and x_1, x_2, x_3,... to denote variables. We use t, r, s to denote arbitrary terms. An f-application is a term of the form $f(t_1, \ldots, t_n)$. We call an *atomic formula* a formula of the form $p(t_1, \ldots, t_n)$, where p is a predicate symbol. A *literal* is an atomic formula or the negation of one. A *clause* is a disjunction $l_1 \vee \ldots \vee l_n$ of literals. A *CNF formula* is a conjunction $C_1 \wedge \ldots \wedge C_n$ of clauses. We use C_i to denote the i-th clause in a CNF formula. Without loss of generality, we assume every formula that is being checked for satisfiability is in CNF, and any variable in a clause C is universally quantified. We also assume that existentially quantified variables were eliminated using *Skolemization*. For instance, the formula $\forall x\colon \exists y\colon \neg p(x) \vee q(y)$ is converted into the equisatisfiable formula $\forall x\colon \neg p(x) \vee q(f_y(x))$, where f_y is a fresh function symbol. We will write CNF formulas replacing the $\wedge$ connectives by commas. We use $C[x_1, \ldots, x_n]$ to denote a clause that may contain variables x_1, ..., x_n, and a similar notation $t[x_1, \ldots, x_n]$ is defined for a term t. Where there is no confusion, we denote $C[x_1, \ldots, x_n]$ by $C[\overline{x}]$ and $t[x_1, \ldots, x_n]$ by $t[\overline{x}]$. In the rest of this paper, the difference between functions and predicates is trivial, and we will thus only discuss functions except at a few places.

A first order Σ-theory T is a set of deductively closed Σ-sentences. Interpreted symbols are those symbols whose interpretation is restricted to the models of a certain theory T, whereas free or uninterpreted symbols are those symbols not in Σ. In this paper we assume the usual interpreted symbols for equality, linear arithmetic and bit-vectors. We use $\simeq$ to denote the interpreted predicate symbol for equality.

A Σ-structure M consists of a non-empty universe $|M|$ and an interpretation for variables and symbols in Σ. For each symbol f in Σ, the interpretation of f is denoted by $M(f)$. For a function symbol f with $\mathit{arity}(f) = n$, the interpretation $M(f)$ is a total n-ary function on $|M|$. For each predicate symbol p with $\mathit{arity}(p) = n$, $M(p)$ is a subset of $|M|^n$. For a variable x, $M(x)$ is an element in $|M|$. The interpretation of an arbitrary term t is denoted by $M[\![t]\!]$ and defined as: $M[\![m]\!] = M(m)$ for constant or variable m, and $M[\![f(t_1, \ldots, t_n)]\!] = M(f)(M[\![t_1]\!], \ldots, M[\![t_n]\!])$. If S is a set of terms, $M(S)$ means the set $\{M(t) \mid t \in S\}$.

We use $M\{x \mapsto v\}$ to denote a structure where the variable symbol x is interpreted as v, $v \in |M|$, and all other variables, function and predicate symbols have the same interpretation as in M. That is $M\{x \mapsto v\}(x) = v$. $M\{\overline{x} \mapsto \overline{v}\}$ denotes $M\{x_1 \mapsto v_1\}\{x_2 \mapsto v_2\} \ldots \{x_n \mapsto v_n\}$.

Satisfaction $M \models \phi$ is defined as usual. In particular, given atomic formula $p(t_1, \ldots, t_n)$, $M \models p(t_1, \ldots, t_n)$ if and only if $(M[\![t_1]\!], \ldots, M[\![t_n]\!]) \in M(p)$. Given a vector $\overline{x}$ of n variables, $M \models C[\overline{x}]$ if for all $\overline{v} \in |M|^n$ there is a literal l in $C[\overline{x}]$

such that $M\{\overline{x} \mapsto \overline{v}\} \models l$. M is a model for a formula F if $M \models F$. A formula is *satisfiable* if and only if it has a model. A formula F is satisfiable modulo theory T if there is a model for $\{F\} \cup T$. A formula F is satisfiable modulo a class Ω of intended structures if there is a M in Ω that is a model for F.

3 Essentially Uninterpreted Formulas

Given a theory T, we say a formula F is *essentially uninterpreted* if any variable in F appears only as an argument of uninterpreted function or predicate symbols.

Example 1 (essentially uninterpreted clause). In the following example, the symbols $+$ and $\leq$ are interpreted by the theory of arithmetic, and a, b, f, g, h and p are uninterpreted symbols.

$$f(g(x_1) + a) \leq h(x_1) \vee p(f(x_1) + b, x_2)$$

We show that every essentially uninterpreted formula F is equisatisfiable to a (potentially infinite) set of ground formulas F^*. For that purpose, let us introduce some additional notational conventions. For a term $t[x_1, \ldots, x_n]$, $t[r_1, \ldots, r_n]$ is the result of simultaneously substituting r_i for x_i $(1 \leq i \leq n)$ in t. When S_i $(1 \leq i \leq n)$ are sets, we shall write $t[S_1, \ldots, S_n]$ for $\{t[r_1, \ldots, r_n] \,|\, r_1 \in S_1, \ldots, r_n \in S_n\}$. For a clause C, $C[r_1, \ldots, r_n]$ and $C[S_1, \ldots, S_n]$ are defined in the obvious way, where r_i are ground terms and S_i are sets of ground terms. For each variable x_i in every C_k, we introduce a set $S_{k,i}$. For each uninterpreted function symbol f with arity n, we introduce the sets $A_{f,1}, \ldots, A_{f,n}$. We obtain the sets $S_{k,i}$ and $A_{f,j}$ as the least solution to a system of set constraints Δ_F induced by F. The constraints in Δ_F describe relationships between sets of terms. We follow the set constraint conventions also used in [13]. We generate Δ_F using the following rules based on the occurrences $f(\overline{s})$ of uninterpreted function symbols in F.

j-th argument of f in C_k	*Set constraint*
a ground term t	$t \in A_{f,j}$
$t[x_1, \ldots, x_n]$	$t[S_{k,1}, \ldots, S_{k,n}] \subseteq A_{f,j}$
x_i	$S_{k,i} = A_{f,j}$

Informally, the first two rules capture the *relevant domain*, a set of ground terms, of an uninterpreted function symbol f. The first says that any ground argument of an f-application in F is relevant, and the second says the *relevant domain* of a function symbol f may be increased when we instantiate a non-ground clause containing f. In the second rule, we are implicitly assuming t is a non ground term. The last rule states that it is sufficient to instantiate x_i using terms of the *relevant domain*. Without loss of generality, we assume the least solution of Δ_F contains only non-empty sets of terms. We can always add extra constraints of the form $a \in S$ to Δ_F to force S to be non-empty. Since we are discussing essentially uninterpreted formulas, for each $S_{k,i}$ there must be a

equation $S_{k,i} = A_{f,j}$ in Δ_F. Intuitively, $A_{f,j}$ contains all ground terms that can be the j-th argument of f, and $S_{k,i}$ is the set of ground terms that can appear in the place of x_i in C_k. To illustrate the construction of Δ_F, consider the following example.

Example 2 (Δ_F construction). Let F be the following four clauses.

$$g(x_1, x_2) \simeq 0 \vee h(x_2) \simeq 0,$$
$$g(f(x_1), b) + 1 \leq f(x_1),$$
$$h(b) \simeq 1, \quad f(a) \simeq 0$$

Δ_F is:

$$S_{1,1} = A_{g,1}, \quad S_{1,2} = A_{g,2}, \quad S_{1,2} = A_{h,1}$$
$$S_{2,1} = A_{f,1}, \quad b \in A_{g,2}, \quad f(S_{2,1}) \subseteq A_{g,1}$$
$$b \in A_{h,1}, \quad a \in A_{f,1}$$

The least solution of Δ_F is $S_{1,1} = A_{g,1} = \{f(a)\}$, $S_{1,2} = A_{g,2} = A_{h,1} = \{b\}$, $S_{2,1} = A_{f,1} = \{a\}$.

Now we define F^* as the set of ground clauses $\{C_k[S_{k,1}, \ldots, S_{k,m}] \,|\, C_k \text{ in } F\}$. That is, F^* is obtained by instantiating clauses with ground terms from $S_{k,i}$. For the above example, F^* is

$$g(f(a), b) \simeq 0 \vee h(b) \simeq 0, \quad g(f(a), b) + 1 \leq f(a), \quad h(b) \simeq 1, \quad f(a) \simeq 0$$

Proposition 1. *For every ground term $f(\ldots, t_j, \ldots)$ in F^*, where t_j is the j-th argument and f is uninterpreted, t_j is in $A_{f,j}$.*

Suppose F^* has a model M, for each $A_{f,j}$, we define a projection function $\pi_{f,j}$ on $|M|$ such that $\pi_{f,j}(v) \in M(A_{f,j})$ and $\pi_{f,j}(v) = v$ when $v \in M(A_{f,j})$. The projection functions essentially map the domain of f to its relevant subset. Similarly we define a projection function $\pi_{k,i}$ for each $S_{k,i}$ and require $\pi_{k,i} = \pi_{f,j}$ if $S_{k,i} = A_{f,j}$ appears in Δ_F.

The functions $\pi_{f,j}$ and $\pi_{k,i}$ are well-defined because we assume the least solution of Δ_F contains only non-empty sets of terms. We use $\pi_k(\overline{v})$ to denote the tuple $\langle \pi_{k,1}(v_1), \ldots, \pi_{k,m}(v_m) \rangle$.

Suppose F^* has a model M, we construct a model M^π for F. In other words, if F^* is satisfiable, so is F. We say M^π is a *π-extension* of M if $|M^\pi| = |M|$, $M^\pi(a) = M(a)$ for every constant a, $M^\pi(f) = M(f)$ for every interpreted function symbol f, and $M^\pi(f)(v_1, \ldots, v_n) = M(f)(\pi_{f,1}(v_1), \ldots, \pi_{f,n}(v_n))$ for every uninterpreted function symbol f.

Lemma 1. *For every ground term t in F^*, $M[\![t]\!] = M^\pi[\![t]\!]$.*

Proof. (Sketch)[3] By induction on the complexity of t. The proof is based on the observation that $M[\![s_i]\!] \in M(A_{f,i})$ whenever s_i is the i-th argument of an uninterpreted function symbol f in the set F^*.

[3] The full proofs for the proof sketches in this paper can be found at [14] (http://research.microsoft.com/~leonardo/citr09.pdf).

Lemma 1 implies that M^π is also a model of F^* if M is. Now, let us show that M^π is also a model for F.

Lemma 2. *For every term $t[\overline{x}]$ in clause C_k, where $t[\overline{x}]$ is not a variable, we have that for all tuples of $\overline{v}$, $M^\pi\{\overline{x} \mapsto \overline{v}\}[\![t[\overline{x}]]\!] = M\{\overline{x} \mapsto \pi_k(\overline{v})\}[\![t[\overline{x}]]\!]$.*

Proof. (Sketch) By induction on the complexity of t. The only interesting case is when t is $f(\overline{s})$, and f is an uninterpreted symbol. Then, it suffices to show that $\pi_{f,j}(M^\pi\{\overline{x} \mapsto \overline{v}\}[\![s_j[\overline{x}]]\!]) = M\{\overline{x} \mapsto \pi_k(\overline{v})\}[\![s_j[\overline{x}]]\!]$, for each s_j in $\overline{s}$. We consider three cases: $s_j[\overline{x}]$ is ground, a variable, or a composite non-ground term. The first case follows from Lemma 1. The second is based on the observation that $\pi_{k,i} = \pi_{f,j}$ whenever x_i is the j-th argument of some f in a clause C_k. The final case follows from the induction hypothesis, and the fact that Δ_F contains the constraint $s_j[S_{k,1}, \ldots, S_{k,m}] \subseteq A_{f,j}$.

Theorem 1. *F and F^* are equisatisfiable.*

Proof. If F^* is unsatisfiable, then so is F, since F^* is the conjunction of ground instances of F. Suppose that F^* is satisfiable, but F is not. Let M be a model for F^* and M^π be its π-extension. Since F is unsatisfiable, there is a clause $C_k[\overline{x}]$ in F such that for some $\overline{v}$, $M^\pi\{\overline{x} \mapsto \overline{v}\} \not\models C_k[\overline{x}]$. By Lemma 2, $M\{\overline{x} \mapsto \pi_k(\overline{v})\} \not\models C_k[\overline{x}]$. Let $\overline{v}$ be the tuple $\langle v_1, \ldots, v_m\rangle$, then for every $\pi_{k,i}(v_i)$ in $\pi_k(\overline{v})$, there is some ground term r_j in $S_{k,j}$ such that $M[\![r_j]\!] = \pi_{k,j}(v_j)$. Thus, $M\{\overline{x} \mapsto \pi_k(\overline{v})\} \models C_k[\overline{x}]$ if and only if $M \models C_k[\overline{r}]$, and consequently $M \not\models C_k[\overline{r}]$, contradicting the assumption that F^* is satisfiable since $C_k[\overline{r}]$ is in F^*.

We say a formula F is in the *finite essentially uninterpreted fragment* (FEU) if every $S_{k,i}$ is finite in the least solution of Δ_F. A system Δ_F is *stratified* if there is a function *level* from set variables into natural numbers such that for each constraint $S_{k,j} = A_{f,i}$, $\mathit{level}(S_{k,j}) = \mathit{level}(A_{f,i})$, and for each constraint $t[S_{k,1}, \ldots, S_{k,n}] \subseteq A_{f,j}$, $\mathit{level}(A_{f,j}) < \mathit{level}(S_{k,i})$, for all $i = 1, \ldots, n$.

Proposition 2. *The least solution of Δ_F is finite if and only if Δ_F is stratified.*

By proposition 2, a formula F is in the FEU fragment if and only if Δ_F is stratified. Theorem 1 suggests a simple decision procedure for the formulas in the FEU fragment. We just generate F^* and check its satisfiability using a SMT solver such as Z3 [3]. The decidability problem for FEU-formulas is NEXPTIME-hard, because F^* is finite for any formula in the Bernays-Schönfinkel (EPR) class. The EPR class comprise of formulas of the form $\forall\overline{x}\colon \varphi(\overline{x})$, where $\varphi(\overline{x})$ is a quantifier-free formula with relations, equality, constants, but without non constant function symbols. The size of F^* is at most doubly exponential in the size of F. The first exponential blowup is caused by the construction of the least solution of Δ_F. For example, assume Δ_F contains the following constraints:

$$a \in S_1,\ b \in S_1,\ f_1(S_1, S_1) \subseteq S_2,\ f_2(S_2, S_2) \subseteq S_3,\ \ldots,\ f_n(S_n, S_n) \subseteq S_{n+1}$$

The second exponential blowup is caused by the instantiation of the clauses $C_k[\overline{x}]$.

Compactness The least solution of Δ_F is infinite if some $S_{k,i}$ in the least solution of Δ_F is infinite. If Δ_F is infinite, then F^* is an infinite set of ground clauses. Therefore, a tempting possibility is to assume a refutationally complete procedure can be devised by using the Compactness Theorem for first-order logic. The Compactness Theorem says that for any unsatisfiable set of first-order formulas F, there is a finite subset F' that is also unsatisfiable. In this paper, we are interested in the satisfiability of a Σ'-formula F modulo a Σ-theory T, where the signature Σ' includes Σ. Then, in principle, the satisfiability of F modulo T is equivalent to the satisfiability of $F \cup T$ in pure first-order logic, and the Compactness Theorem can be applied to $F \cup T$. This approach can be used to handle useful background theories such as: algebraic/real closed fields and finite size bit-vectors. However, in practice, we are also interested in checking the satisfiability of F modulo a class Ω of intended structures. Before continuing, let us introduce some notational conventions. Let Σ' be any signature including Σ. An expansion M' to Σ' of a Σ-structure M is a Σ'-structure that has the same universe as M, and agrees with M on the interpretation of the symbols in Σ. We denote by $Exp_{\Sigma'}(\mathcal{T})$ the class of all Σ'-structures that are expansions of the Σ-structure $\mathcal{T}$. Note that Theorem 1 guarantees that F and F^* are equisatisfiable modulo a class $Exp_{\Sigma'}(\mathcal{T})$ of intended structures, because M and M^π only differ on the interpretation of symbols that are in Σ' but not in Σ. Thus, if M is in $Exp_{\Sigma'}(\mathcal{T})$, then so is M^π. A Σ-theory $Th(\Omega)$ for a class Ω of Σ-structures is the set of all Σ-sentences ϕ such that $M \models \phi$ for every M in Ω. Now, consider the following example:

Example 3 (Nonstandard models of arithmetic). Let Σ be the signature $(0, 1, +, -, <)$. Let $\mathcal{Z}$ be the structure that interprets these symbols in the usual way over the integers. Let Σ' be the signature $(0, 1, +, -, <, f)$. Now, let us check the satisfiability of the following set of Σ'-clauses F modulo the background theory $Th(\mathcal{Z})$.

$$f(x_1) < f(f(x_1)),\ f(x_1) < a,\ 1 < f(0)$$

By Theorem 1, these three clauses are equisatisfiable to the set of ground clauses $F^* = \{f(0) < f^2(0), f^2(0) < f^3(0), \ldots, f(0) < a, f^2(0) < a, \ldots, 1 < f(0)\}$ modulo $Th(\mathcal{Z})$. Since, every finite subset of $F^* \cup Th(\mathcal{Z})$ is satisfiable, then by compactness $F^* \cup Th(\mathcal{Z})$ is satisfiable. This is counterintuitive, since clause $f(x_1) < f(f(x_1))$ implies that the range of any interpretation of f contains infinite strictly increasing chains $M(f)(v) < M(f)^2(v) < \ldots < M(f)^n(v) < \ldots$, and clause $f(x_1) < a$ says there is a value $M(a)$ greater than any value in the range of $M(f)$. The problem here is that $Th(\mathcal{Z})$ has nonstandard models. Now suppose we want to check the satisfiability of F modulo the class of structures $Exp_{\Sigma'}(\mathcal{Z})$. F^* is still equisatisfiable to F modulo $Exp_{\Sigma'}(\mathcal{Z})$, but we cannot apply the Compactness Theorem. Therefore, if F^* is infinite, the procedures described in this paper are not refutationally complete for satisfiability modulo $Exp_{\Sigma'}(\mathcal{Z})$. Note also that Theorem 1 does not hold if the background theory is $Th(Exp_{\Sigma'}(\mathcal{Z}))$ because this theory restricts the interpretations of the function

symbols in $\Sigma' \setminus \Sigma$. For instance, it contains a sentence stating that if f is a strictly increasing function, then the range of f does not have a supremum.

From hereafter, we only consider the problem of checking the satisfiability of F modulo a theory T, instead of satisfiability modulo a class Ω of intended structures. Therefore, if F^* is unsatisfiable, there is a finite subset of F^* that is also unsatisfiable. Given a fair enumeration of F^*, we obtain a refutationally complete procedure. By fair, we mean a sequence $F^1 \subseteq F^2 \subseteq \ldots F^i \subseteq \ldots \subseteq F^*$, where each F^i is a finite set, and for each clause C in F^* there is an n such that C is in F^n. A fair enumeration of F^* can be obtained by performing a fair enumeration of the least solution of the system Δ_F. It is not difficult to generate such enumeration [14]. For each $S_{k,i}$ in Δ_F, we have a sequence $S^0_{k,i} \subseteq S^1_{k,i} \subseteq S^2_{k,i} \subseteq \ldots \subseteq S_{k,i}$, where each $S^j_{k,i}$ is finite, and for each t in $S_{k,i}$, there is an n such that t is in $S^n_{k,i}$. Note that these sequences are finite when Δ_F is stratified.

4 Almost uninterpreted formulas

In an essentially uninterpreted formula, a variable x can only be the argument of uninterpreted function and predicate symbols. In this section, we present many extensions of the framework described so far. The first trivial extension is to use *destructive equality resolution* (DER) as a preprocessing step. In DER, the clause $\neg(x \simeq t) \vee C[x]$ is simplified to $C[t]$, when x does not occur in t. From hereafter, all proposed extensions come equipped with new rules for generating constraints for Δ_F. As before, the idea is to show that a formula F in the extended fragment is equisatisfiable to a set of ground formulas F^*. Moreover, if the least solution of Δ_F is finite, the satisfiability of F can be determined in finite time.

Arithmetical literals First, let us consider literals of the form $\neg(x_i \leq x_j)$, $\neg(x_i \leq t)$, $\neg(t \leq x_i)$, and $x_i \simeq t$, where t is a ground term. The literal $x_i \simeq t$ is in the new fragment only if x_i ranges over integers. We say these literals are *arithmetical*. Positive literals of the form $x_i \leq t$ can be rewritten into $\neg(t{+}1 \leq x_i)$ if x_i ranges over integers. In order to support arithmetical literals, we use the following additional rules to generate the system Δ_F.

Literal of C_k	*Set constraint*
$\neg(x_i \leq x_j)$	$S_{k,i} = S_{k,j}$
$\neg(x_i \leq t), \neg(t \leq x_i)$	$t \in S_{k,i}$
$x_i \simeq t$	$\{t+1, t-1\} \subseteq S_{k,i}$

We say a formula F is *almost uninterpreted* if any variable in F appears as an argument of an arithmetical literal, or as an argument of uninterpreted function or predicate symbols. To handle almost uninterpreted formulas, we define a new projection function $\pi_{k,i}$. With a small abuse of notation, we use $v_1 \leq v_2$ to denote $(v_1, v_2) \in M(\leq)$, and $v_1 > v_2$ to denote $(v_1, v_2) \notin M(\leq)$. Then, $\pi_{k,i}(v) = v_1$ such that $v_1 \in M(S_{k,i})$, and either $v_1 \leq v$ and for all $v_2 \in M(S_{k,i})$, $v_2 \leq v_1$ or

$v_2 > v$; or $v_1 > v$ and for all $v_2 \in M(S_{k,i})$, $v_1 \leq v_2$. As before, $\pi_{k,i} = \pi_{f,j}$ if $S_{k,i} = A_{f,j}$ appears in Δ_F. We remark that the range of $\pi_{k,i}$ is equal to $M(S_{k,i})$, and $\pi_{k,i}(v) = v$ for any $v \in M(S_{k,i})$. Thus, the proof of Lemma 2 is not affected.

Proposition 3. *The projection functions $\pi_{k,j}$ defined above are monotonic. That is, for all v_1 and v_2 in $|M|$, $v_1 \leq v_2$ implies $\pi_{k,j}(v_1) \leq \pi_{k,j}(v_2)$.*

Theorem 2. *F and F^* are equisatisfiable.*

Proof. It suffices to show that for each arithmetical literal $l[\overline{x}]$, if $M^\pi\{\overline{x} \mapsto \overline{v}\} \not\models l[\overline{x}]$ then $M\{\overline{x} \mapsto \pi_k(\overline{v})\} \not\models l[\overline{x}]$. This is an immediate consequence of Proposition 3, and the fact that $\pi_{k,i} = \pi_{k,j}$ when $l[\overline{x}]$ is of the form $\neg(x_i \leq x_j)$. The rest of the proof is identical to the proof of Theorem 1.

Example 4 (Stratified Arrays). The fragment described in this section can decide the following set of satisfiable clauses. In this example, f should be viewed as an array of pointers, and h as a heap from pointers to values, h' is the heap h after an update at position a with value b. The first clause states that the array f is sorted in the range $[0, n]$. If we replace c with a, the example becomes unsatisfiable.

$$\neg(0 \leq x_1) \vee \neg(x_1 \leq x_2) \vee \neg(x_2 \leq n) \vee h(f(x_1)) \leq h(f(x_2)),$$
$$\neg(0 \leq x_1) \vee \neg(x_1 \leq n) \vee f(x_1) \not\simeq c,$$
$$\neg(x_1 \simeq a) \vee h'(x_1) \simeq h(x_1), \quad h'(a) = b$$
$$0 \leq i, \quad i \leq j, \quad j \leq n, \quad h'(f(i)) > h'(f(j))$$

Offsets We now consider terms of the form $f(\ldots, x_i + r, \ldots)$, where r is a ground term. For this extension, we use the following additional rule:

j-th argument of f in C_k	*Set constraint*
$x_i + r$	$S_{k,i} + r \subseteq A_{f,j},\ A_{f,j} + (-r) \subseteq S_{k,i}$

Without this additional rule, it is not possible, for instance, to detect the unsatisfiability of $\{p(f(x+1)),\ \neg p(f(a))\}$. The set $S + r$ is defined as the set of ground terms $\{t \oplus r \mid t \in S\}$, where $t \oplus r$ creates a term equivalent to $t + r$ modulo the simplification rules: $(x + y) + (-y) \rightsquigarrow x$, and $(x + (-y)) + y \rightsquigarrow x$. For example, $(t + (-r)) \oplus r = t$. These simplifications prevent Δ_F from being trivially infinite. Again, with a small abuse of notation, we use $v_1 + v_2$ to denote $M(+)(v_1, v_2)$. Similarly, $v_1 - v_2$ denotes $M(-)(v_1, v_2)$. For this extension, we use the same projection functions used for handling arithmetical literals.

Proposition 4. *If $x_i + r$ is the j-th argument of a term $f(\overline{s})$ in clause C_k, then for all $v \in |M|$, $v \in M(A_{f,j})$ if and only if $v - M[\![r]\!] \in M(S_{k,i})$.*

The proof of Lemma 2 has to be updated, since $x_i + r$ can be the argument of an f-application. For this extra case, it suffices to show that $\pi_{f,j}(v_i + M[\![r]\!]) = \pi_{k,i}(v_i) + M[\![r]\!]$. This equality follows from Proposition 4, and from the definition of the projection functions [14].

Example 5 (Shifting). The following clause states that the segment $[2, n+2]$ of the array f is equal to the segment $[0, n]$ of the array g.

$$\neg(0 \leq x_1) \vee \neg(x_1 \leq n) \vee f(x_1 + 2) \simeq g(x_1)$$

Similarly, we can add support for literals of the form $\neg(x_i \leq x_j + r)$. The idea is to include the constraints $S_{k,i} + (-r) \subseteq S_{k,j}$, and $S_{k,j} + r \subseteq S_{k,i}$, for each literal $\neg(x_i \leq x_j + r)$ in a clause C_k.

4.1 Many-sorted first-order logic

Sorts naturally arise in SMT applications and in some cases sort information significantly simplifies the problem. SMT solvers such as CVC3 [15] and Z3 [16] have support for sorts. We say a sort σ is uninterpreted if it is not in the signature of the background theory. We use $\simeq_\sigma$ to denote the equality predicate for elements of sort σ. Given a formula F in many-sorted logic, we can support any literal using the equality predicate $\simeq_\sigma$, when σ is an uninterpreted sort. The basic idea is to axiomatize the equality predicate $\simeq_\sigma$, and treat it as an uninterpreted predicate symbol. That is, we add the clauses EQ_σ asserting that $\simeq_\sigma$ is reflexive, symmetric, transitive, and congruent. This is a standard technique used in saturation-based provers that do not have built-in support for equality. The previous theorems asserting the equisatisfiability of F and F^* can be easily adapted to the many-sorted case. In practice, we do not really need to add the clauses EQ_σ, since any SMT solver has built-in support for equality. It is sufficient to add to Δ_F any constraints that are induced by EQ_σ. We denote by $dom_{f,j}$ the sort of the j-th argument of f. We introduce the auxiliary set S_σ in Δ_F. Intuitively, S_σ contains the ground terms of sort σ. We use the following additional rules to generate Δ_F.

argument of $\simeq_\sigma$ *in* C_k	*Set constraint*
x_i	$S_{k,i} = S_\sigma$
$t[x_1, \ldots, x_n]$	$t[S_{k,1}, \ldots, S_{k,n}] \subseteq S_\sigma$

Sort declaration	*Set constraint*
$dom_{f,j} = \sigma$	$A_{f,j} = S_\sigma$

For example, now we can handle the anti-symmetry axiom used in the axiomatization of the *subtype* relation in ESC/Java [10]: $\neg subtype(x_1, x_2) \vee \neg subtype(x_2, x_1) \vee x_1 \simeq_\sigma x_2$. From hereafter, we suppress the σ in $\simeq_\sigma$.

5 Macros and pseudo-macros

In practice, many formulas contain non-ground clauses that can be seen as *macro definitions*. These clauses have the following form: $g(\overline{x}) \simeq t[\overline{x}]$, where g does not occur in $t[\overline{x}]$. For example, the macro $g(x_1) \simeq x_1 + c$ is not in any of the fragments described so far. The simplest way to handle a macro $g(\overline{x}) \simeq t[\overline{x}]$ is to remove it

from F, and replace every term of the form $g(\overline{s})$ with $t[\overline{s}]$. Clearly, the resultant formula is equisatisfiable to F. More generally, we say g is a *pseudo-macro* defined by the non-ground clauses $D_g = \{C_1[\overline{x}], \ldots, C_n[\overline{x}]\}$ if all clauses in D_g contain $g(\overline{x})$, and are trivially satisfied (i.e., are equivalent to *true*) by replacing $g(\overline{x})$ with a term $t_g[\overline{x}]$.

Example 6 (Pseudo-Macro). The function symbol g is a pseudo-macro in the following example. Note that replacing $g(x_1)$ with 0 trivially satisfies the first two clauses.

$$g(x_1) \geq 0 \vee f(g(x_1)) \simeq x_1, \quad g(x_1) \geq 0 \vee h(g(x_1)) \leq g(x_1), \quad g(a) < 0$$

Many different heuristics may be used to find pseudo-macros. For example, it is clear that g is a pseudo-macro if $D_g = \{C_1[\overline{x}] \vee g(\overline{x}) \bowtie t_g[\overline{x}], \ldots, C_n[\overline{x}] \vee g(\overline{x}) \bowtie t_g[\overline{x}]\}$, where $\bowtie$ is $\simeq$, $\leq$, or $\geq$. From hereafter, we assume some heuristic was used to select the pseudo-macros g, their definitions D_g, and the terms $t_g[\overline{x}]$. We now describe how to incorporate pseudo-macros in our framework. A clause C is *regular* if C is not in any set D_g. First we observe that a pseudo-macro g may occur in regular clauses C if none of its arguments is a variable. Intuitively, a pseudo-macro g is treated as an interpreted function symbol in regular clauses. The rules for generating constraints from regular clauses are unchanged. For clauses $C_k[\overline{x}]$ in D_g, we use slightly different rules. The main difference is the rule for a variable x_i occurring as an argument of f.

x_i is an argument of f in C_k	*Set constraint*
$f = g$	$S_{k,i} = A_{f,j}$
$f \neq g$	$S_{k,i} \subseteq A_{f,j}$

The construction of M^π is also slightly modified. If g is a pseudo-macro, then $M^\pi(g)$ is defined in the following way

$$\begin{array}{ll} M^\pi(g)(\overline{v}) = M[\![g(\overline{s})]\!] & \text{if } g(\overline{s}) \in F^*, \text{ and } M[\![\overline{s}]\!] = \overline{v} \\ \quad = M\{\overline{x} \mapsto \overline{v}\}[\![t_g[\overline{x}]]\!] & \text{otherwise} \end{array}$$

The proof that F and F^* are still equisatisfiable, when F contains pseudo-macros, is a direct consequence of Theorem 1 and the definition above.

6 Implementation

Some of the ideas described in this paper were already implemented in the Z3 theorem prover submitted to the SMT 2008 competition[4]. The extensions for many-sorted logic, and pseudo-macros defined by multiple clauses were not implemented yet.

We would like to make it clear that Z3's performance is not a consequence of the theory or heuristics described on this paper. On the other hand, the techniques proposed here increased Z3's precision. For example, Z3 was the only

[4] http://www.smtcomp.org

theorem prover in the competition that produced the correct answers for satisfiable problems in the divisions for quantified formulas. Z3 detected that 33 benchmarks with quantifiers were satisfiable; 3 of them were *Almost Uninterpreted Formulas* (AUF), and 30 were AUF+*pseudo-macros*.

For some applications, it is desirable not only to know whether a formula is satisfiable, but also, what a satisfying model is. In general, it is very challenging to capture the structure of an arbitrary first-order model. We have a more modest goal: we only want to describe models for the decidable fragments described in this paper. We also propose a heuristic to minimize the number of instantiations. The basic idea is to use "candidate" models to guide quantifier instantiation. A similar idea is used in the theorem prover Equinox [17] for pure first-order logic.

Model representation Assume T is a Σ-theory, Σ' includes the signature Σ, and F is a set of Σ'-clauses. In our implementation, a "model" is essentially a function that maps a Σ-structure $\mathcal{T}$ that satisfies T, into an expanded Σ'-structure M that satisfies $T \cup F$. Our "models" also come equipped with a set of formulas R that restricts the class of Σ-structures that satisfy T. If T is the empty theory, then R is just a cardinality constraint on the size of the universe. When needed, we use fresh constant symbols $k_1, \ldots, k_n$ to name the elements in $|M|$. We also use R to restrict the interpretation of under-specified interpreted function symbols such as: division and modulo [14]. In our implementation, the interpretation of an uninterpreted symbol s in $\Sigma' \setminus \Sigma$ is an expression $I_s[\overline{x}]$, which contains only interpreted symbols and the fresh constants $k_1, \ldots, k_n$. For uninterpreted constants c, $I_c[\overline{x}]$ is a ground term I_c, and $M[\![c]\!] = \mathcal{T}[\![I_c]\!]$. When I_c is ground we say it is a *value*. For uninterpreted function and predicate symbols, the term $I_s[\overline{x}]$ should be viewed as a *lambda expression*, where for all $\overline{v}$ in $|M|^n$, $M(f)(\overline{v}) = \mathcal{T}\{\overline{x} \mapsto \overline{v}\}[\![I_f[\overline{x}]]\!]$. We assume the construct $ite(\phi, t_1, t_2)$ (the if-then-else construct for terms) is available in our language.

Example 7 (Model representation). Let F be the following four clauses.

$$\neg(5 \leq x_1) \vee f(x_1) < 0, \quad f(a) \simeq 0, \quad f(b) \simeq 1, \quad a < 2$$

These clauses are satisfiable, and Z3 generates the following model.

$$a \mapsto 0, \quad b \mapsto 2, \quad f \mapsto ite(x_1 < 2, 0, ite(x_1 < 5, 1, -1))$$

Note that SMT solvers can be used to model check any clause $C_k[\overline{x}]$ in F. Let $C_k^I[\overline{x}]$ be the clause obtained from $C_k[\overline{x}]$ by replacing any term $f(\overline{s})$ with $I_f[\overline{s}]$, when f is uninterpreted. Thus, a model satisfies $C_k[\overline{x}]$ if and only if $R \wedge \neg C_k^I[\overline{w}]$ is unsatisfiable, where $\overline{w}$ is a tuple of fresh constant symbols. For example, in the previous example, the first clause is satisfied by the model above because the following formula is unsatisfiable.

$$5 \leq w_1 \wedge \neg(ite(w_1 < 2, 0, ite(w_1 < 5, 1, -1)) < 0)$$

It is straightforward to construct the term I_c for uninterpreted constants. For uninterpreted function symbols, we construct $I_f[\overline{x}]$ based on the definition of $M^\pi(f)$ using the *ite* construct.

We say M^n is a *candidate model* for F if it is a model for a finite subset F^n of F^*. The set of ground terms $A^n_{f,j}$ contains all j-th arguments of terms $f(\overline{s})$ in F^n. The *candidate interpretation* $I_f[\overline{x}]$ for f is defined using the set $A^n_{f,i}$ instead of the set $A_{f,i}$. If $A^n_{f,i}$ is empty, then $I_f[\overline{x}] = t_f[\overline{x}]$ if f is a pseudo-macro, and $I_f[\overline{x}]$ is an arbitrary constant function otherwise.

Model-based quantifier instantiation (MBQI) Let $t_1 \prec_{k,j} t_2$ be a total order on the terms in $S_{k,j}$ such that $t_1 \prec_{k,j} t_2$ whenever there is an n such that $t_1 \in S^n_{k,j}$ and $t_2 \notin S^n_{k,j}$. Let $\pi^{-1}_{k,j}(v_j, M)$ be a function that maps a value v_j to the least (with respect to $\prec_{k,j}$) ground term r_j in $S_{k,j}$ such that $\pi_{k,j}(v_j) = M[\![r_j]\!]$. As before, we use $\pi^{-1}_k(\overline{v}, M)$ to denote the tuple $\langle \pi^{-1}_{k,1}(v_1, M), \ldots, \pi^{-1}_{k,n}(v_n, M)\rangle$. Instead of generating the fair enumeration of F^*, we guide quantifier instantiation using the model checking procedure described above, and the following procedure.

$\varphi :=$ set of ground clauses in F
loop
  if φ is unsatisfiable **return** unsat
  create the candidate model M^n
  ok := true
  foreach non-ground clause $C_k[\overline{x}]$ in F
    create $C^I_k[\overline{w}]$ using M^n
    if $R \wedge \neg C^I_k[\overline{w}]$ is satisfiable
      let M_k be the model for $R \wedge \neg C^I_k[\overline{w}]$, **and** $\overline{v}$ be $M_k(\overline{w})$.
      $\varphi := \varphi \cup C_k[\pi^{-1}_k(\overline{v}, M^n)]$
      ok := false
  if ok **return** sat

Heuristics Heuristic quantifier instantiation based on E-matching generates a subset of the instances in F^*. An advantage of E-matching is that it can be used incrementally. In [5, 3] it was observed that incremental and eager quantifier instantiation (EQI) is more efficient, in software verification benchmarks, than lazy quantifier instantiation (LQI). In this way, MBQI does not substitute E-matching in Z3, but complements it. MBQI increases the number of benchmarks that Z3 can solve. The prototype of Z3 submitted to SMT-COMP'08 still uses E-matching, and only applies MBQI after a candidate model is produced. In SMT-COMP'08, 22 benchmarks were proved to be unsatisfiable by Z3 using MBQI, and a prover solely based on e-matching would fail on these benchmarks. Another important heuristic used is relevancy propagation [3]. Relevancy propagation keeps track of which truth assignments are essential for determining satisfiability of a formula. Only terms that are marked as relevant are considered for E-matching and constructing candidate models.

7 Related work

The fragment that contains arithmetical literals and the associated projection functions resemble much in spirit the array property fragment and its projection function proposed in [12], which is the original motivation for this paper. It is obvious that our fragments subsume the array property fragment, since we support nested array reads, offsets on indices, and pseudo-macros. As proved in [12], nested array reads and offsets on indices will in general make the formula undecidable. However, we show that for certain cases containing nested array reads and offsets, a complete decision procedure is possible as long as the set F^* is finite. In [18] a logic called LIA is proposed, in which modulo equalities, difference constraints, and non-nested array reads are allowed. The decidability of LIA is proved by employing a customized counter Büchi automata. Compared with LIA, our fragments allow propositional combination of any theory constraints and nested array reads. For certain cases containing offsets on array indices, our procedure will result in an infinite set of instantiations, while a decision procedure of LIA will terminate.

In [19, 20, 11] procedures based on stratified vocabularies are presented. These procedures are in the context of many-sorted logic. A vocabulary is stratified if there is a function *level* from sorts to naturals, and for every function $f\colon \sigma_1 \times \ldots \times \sigma_n \rightarrow \sigma$, $\mathit{level}(\sigma) < \mathit{level}(\sigma_i)$. Our method can decide a broader class of problems. For example, these methods fail if there is a single function $f\colon \sigma \rightarrow \sigma$, and cannot handle simple examples such as $f(x) = b \wedge f(a) = a$. In [21] local theories and local extensions are studied; they propose a complete instantiation procedure for certain types of quantified formulas. One major difference is that our method can provide models for satisfiable cases.

In our approach, if T is the empty theory, then Theorem 1 can be viewed as a frugal version of the standard Herbrand theorem, and the universe does not necessarily become infinite in the presence of function symbols.

8 Conclusion

We proposed several new useful fragments of first order logic modulo theories that have complete instantiation. We showed how to construct models for satisfiable formulas in these fragments. We also described undecidable fragments and discussed the conditions under which a refutationally complete procedure exists. We discussed the difference between a theory as a deductively closed set of sentences, and as a class of intended structures. We used model-based quantifier instantiation to prioritize quantifier instantiation. Some of ideas in this paper have been implemented in Z3 2.0. In the last SMT competition, Z3 was the only prover that solved satisfiable quantified formulas. Future work includes investigation of more heuristics to prioritize instantiations, and more decidable fragments. For instance, our approach cannot handle a clause containing the term $f(x_1 + x_2)$, where x_1 and x_2 are universally quantified variables.

Acknowledgments We'd like to thank Tom Ball, Clark Barrett, Margus Veanes, Cesare Tinelli, and the anonymous reviewers for reading early drafts of this paper, and providing helpful feedback.

References

1. Halpern, J.Y.: Presburger Arithmetic with unary predicates is Π^1_1 Complete. Journal of Symbolic Logic **56** (1991) 637–642
2. Deharbe, D., Ranise, S.: Satisfiability solving for software verification. International Journal on Software Tools Technology Transfer (2008) to appear.
3. de Moura, L.M., Bjørner, N.: Efficient E-Matching for SMT Solvers. In: CADE. (2007)
4. Stickel, M.E.: Automated deduction by theory resolution. Journal of Automated Reasoning **1** (1985) 333–355
5. Ge, Y., Barrett, C., Tinelli, C.: Solving quantified verification conditions using satisfiability modulo theories. In: CADE. (2007)
6. Flanagan, C., Joshi, R., Saxe, J.B.: Theorem proving using lazy proof explication. In: CAV. (2003)
7. Dutertre, B., de Moura, L.: A Fast Linear-Arithmetic Solver for DPLL(T). In: CAV'06. LNCS 4144, Springer-Verlag (2006) 81–94
8. Detlefs, D., Nelson, G., Saxe, J.B.: Simplify: a theorem prover for program checking. J. ACM **52** (2005)
9. Barnett, M., Chang, B.Y.E., DeLine, R., Jacobs, B., Leino, K.R.M.: Boogie: A modular reusable verifier for object-oriented programs. In: FMCO. (2005)
10. Flanagan, C., Leino, K.R.M., Lillibridge, M., Nelson, G., Saxe, J.B., Stata, R.: Extended static checking for java. In: PLDI '02, ACM (2002)
11. Abadi, A., Rabinovich, A., Sagiv, M.: Decidable fragments of many-sorted logic. In: LPAR. (2007)
12. Bradley, A.R., Manna, Z., Sipma, H.B.: What's decidable about arrays? In: VMCAI. (2006)
13. Aiken, A.: Set Constraints: Results, Applications, and Future Directions. In: Second Workshop on the Principles and Practice of Constraint Programming. (1994)
14. Ge, Y., de Moura, L.: Complete instantiation for quantified SMT formulas . Technical report, Microsoft Research (2009)
15. Barrett, C., Tinelli, C.: CVC3. In: CAV '07. (2007)
16. de Moura, L., Bjørner, N.: Z3: An Efficient SMT Solver. In: TACAS 08. (2008)
17. Claessen, K.: Equinox, a new theorem prover for full first-order logic with equality. Presentation at Dagstuhl Seminar 05431 on Deduction and Applications (2005)
18. Habermehl, P., Iosif, R., Vojnar, T.: What else is decidable about integer arrays? In: FoSSaCS. (2008)
19. Fontaine, P., Gribomont, E.P.: Decidability of invariant validation for paramaterized systems. In: TACAS. (2003)
20. Arons, T., Pnueli, A., Ruah, S., Xu, J., Zuck, L.D.: Parameterized verification with automatically computed inductive assertions. In: CAV. (2001)
21. Ihlemann, C., Jacobs, S., Sofronie-Stokkermans, V.: On local reasoning in verification. In: TACAS. (2008)