# Quantifier Instantiation Techniques for Finite Model Finding in SMT[⋆]

Andrew Reynolds[1], Cesare Tinelli[1], Amit Goel[2],
Sava Krstić[2], Morgan Deters[3], and Clark Barrett[3]

[1] Department of Computer Science, The University of Iowa
[2] Strategic CAD Labs, Intel Corporation
[3] New York University

**Abstract.** SMT-based applications increasingly rely on SMT solvers being able to deal with quantified formulas. Current work shows that for formulas with quantifiers over uninterpreted sorts counter-models can be obtained by integrating a finite model finding capability into the architecture of a modern SMT solver. We examine various strategies for on-demand quantifier instantiation in this setting. Here, completeness can be achieved by consider all ground instances over the finite domain of each quantifier. However, exhaustive instantiation quickly becomes unfeasible with larger domain sizes. We propose instantiation strategies to identify and consider only a selection of ground instances that suffices to determine the satisfiability of the input formula. We also examine heuristic quantifier instantiation techniques such as $E$-matching for the purpose of accelerating the search. We give experimental evidence that our approach is practical for use in industrial applications and is competitive with other approaches.

## 1 Introduction

Solvers for satisfiability modulo theories (SMT) are concerned with the problem of determining the satisfiability of a set of formulas in some first order theory $T$, which is possibly the combination of several sub-theories. SMT solvers use sophisticated and very effective techniques for deciding the satisfiability of ground formulas. While some of them can reason about quantified formulas, they do that using incomplete methods. Hence they often report "unknown" when they fail, after some predetermined amount of effort, to prove a quantified input formula unsatisfiable. For many client applications, however, it is very useful to know when such formulas are indeed satisfiable. Current SMT solvers are able to produce models of satisfiable quantified formulas only in fairly restricted cases [8], which limits their scope and usefulness. To address this limitation, in previous work we have developed a general method for efficient finite model finding in SMT [13]. More precisely, since SMT solvers work in sorted logics with both built-in and *free* ("uninterpreted") sorts, the method looks for models that interpret the latter as finite domains—and so is restricted to SMT formulas with quantifiers ranging only over the free sorts.

Like finite model finders for standard first-order logic, our method is based on checking universal quantifiers exhaustively over candidate models with increasingly

[⋆] The work of the first two authors was partially funded by a grant from Intel Corporation.

large domains for the free sorts, until an actual model is found. It contrasts with previous approaches for not relying on the explicit introduction of *domain constants* for the free sorts, as done by MACE-style model finders [6], and for being able to reason modulo more theories than just the theory of equality, contrary to SEM-style model finders [15]. The model finder described in [13] incorporates into a general architecture used by many SMT solvers [12] an efficient mechanism for deciding the satisfiability of a set of ground SMT formulas under finite cardinality constraints for the free sorts. This is used to find first a *candidate model*, a model $\mathcal{M}$ of a set of ground formulas generated from the input formula $\varphi$. To check that $\mathcal{M}$ satisfies $\varphi$ as well, the model finder then checks, by exhaustive instantiation, that all the ground instances of $\varphi$ over the universe of $\mathcal{M}$ are satisfied by $\mathcal{M}$. When this check fails, the model finder looks for a new candidate model, possibly under extended cardinality bounds for the free sorts.

**Contribution** The contribution of this paper consists in two major improvements to work described in [13]: (1) a method for constructing and representing candidate models efficiently and (2) a model-based instantiation approach that avoids the explicit generation and checking of all the ground instances of the input formula. The two are strictly related since the new instantiation approach takes advantage of the way the model is represented to identify entire sets of instances that do not need to be considered.

**Related work** The data structure we use to represent candidate models is inspired by the *context* data structure introduced in the Model Evolution calculus [2]. The way we construct these models is similar to the generalization mechanism of the Inst-Gen calculus [7]. An instance generation approach similar to ours is taken by [10] for the local theory extensions method. There, the number of generated instances is reduced by finding an unsatisfiable core of relevant ground literals that are in conflict with a candidate model. A different model-based instantiation approach is followed by the Z3 SMT solver [8] where the solver itself is used as an oracle for checking the satisfiability of candidate models.

**Formal preliminaries** We work in the context of many-sorted first-order logic with equality. A (many-sorted) *signature* $\Sigma$ consists of a set $\Sigma^{\mathrm{s}} \subseteq S$ of sort symbols and a set $\Sigma^{\mathrm{f}}$ of *(sorted) function symbols*, $f^{S_1 \cdots S_n S}$, where $n \geq 0$ and $S_1, \ldots, S_n, S \in \Sigma^{\mathrm{s}}$. We drop the sort superscript from function symbols when it is clear from context or unimportant. Without loss of generality, we use equality, denoted by $\approx$, as the only predicate symbol.

Given a signature $\Sigma$, well-sorted terms, atoms, literals, clauses, and formulas are defined as usual, and referred to respectively as $\Sigma$-*terms*, $\Sigma$-*atoms* and so on. A *ground term (resp. formula)* is a $\Sigma$-term (resp. formula) with no variables. A $\Sigma$-*sentence* is a $\Sigma$-formula with no free variables. Where $\mathbf{x} = (x_1, \ldots, x_n)$ is tuple of sorted variables we write $\forall \mathbf{x}\, \varphi$ as an abbreviation of $\forall x_1 \cdots \forall x_n\, \varphi$. A $\Sigma$-formula is *universal* if it has the form $\forall \mathbf{x}\, \varphi$ where $\varphi$ is a quantifier-free formula.

A $\Sigma$-*structure* $\mathcal{M}$ maps each $S \in \Sigma^{\mathrm{s}}$ to a non-empty set $S^{\mathcal{M}}$, the *domain* of $S$ in $\mathcal{M}$, and each $f^{S_1 \cdots S_n S} \in \Sigma^{\mathrm{f}}$ to a total function $f^{\mathcal{M}} : S_1^{\mathcal{M}} \times \cdots \times S_n^{\mathcal{M}} \to S^{\mathcal{M}}$. A satisfiability relation $\models$ between $\Sigma$-structures and $\Sigma$-sentences is defined as usual. A $\Sigma$-structure $\mathcal{M}$ *satisfies* (or *is a model of*) a $\Sigma$-sentence $\varphi$ if $\mathcal{M} \models \varphi$. Entailment between (sets of) sentences, also denoted by $\models$, is also defined as usual.

Given a set $G$ of ground formulas, let $\mathbf{T}_G$ be the set of all terms occurring in $G$. A set $A$ of equalities and disequalities between terms in $\mathbf{T}_G$ is a *(complete) arrangement for $G$* if $A$ is satisfiable and for all $s,t \in \mathbf{T}_G$ of the same sort, $s \approx t$ or $s \not\approx t$ is in $A$. An arrangement $A$ for $G$ *satisfies* $G$ if $A \models G$. A set $E \subseteq \{s \approx t \mid s,t \in \mathbf{T}_G\}$ is a *congruence (for $G$)* if it is closed under entailment: for all $s,t \in \mathbf{T}_G$, $E \models s \approx t$ iff $s \approx t \in E$. The *congruence closure $E^*$ of $E$ (wrt. $G$)* is the smallest congruence for $G$ that includes $E$. By construction, $E^*$ is an equivalence relation over $\mathbf{T}_G$. For any such relation, we will assume as fixed for every sort $S$ with terms in $G$, a set $\mathbf{V}^S = \{v_1^S, \ldots, v_{n_S}^S\}$ consisting of an arbitrary representatives for each equivalence class in $E^*$'s over terms of sort $S$. We call $\mathbf{V}^S$ the set of *$S$-values for $E^*$* and say that terms of sort $S_i$ *have value $v_i^S$ in $E^*$*. For each term $t$ we denote by $\mathsf{v}_{E^*}(t)$ its value in $E^*$. It can be shown (see, e.g., [1]) that $E$ is satisfied by a structure $\mathcal{M}$ that interprets each sort $S$ as $\mathbf{V}^S$. We call $\mathcal{M}$ a *normal model* of $E$. By reducing $G$ to disjunctive normal form it is easy to show that $G$ is satisfiable iff it is satisfied by a normal model of some set $E$ of equalities over $\mathbf{T}_G$.

A congruence $E^*$ over $\mathbf{T}_G$ can be uniquely extended to the arrangement $E^* \cup \{s \not\approx t \mid s \approx t \notin E^*\}$ for $G$. Moreover, that arrangement satisfies $G$ whenever $E^* \models G$. So in the paper *we will often identify congruences and their associated arrangements*.

A substitution $\sigma$ is a mapping from variables to terms of the same sort, such that the set $\{x \mid x\sigma \neq x\}$, the *domain* of $\sigma$, is finite. Unifying substitutions, most general unifiers (mug's), and term variants are defined as usual. Let $\preceq$ be the usual instantiation (quasi-)ordering between terms/atoms: $s \preceq t$ iff $s\sigma = t$ for some substitution $\sigma$. If $T$ is a set of terms and $t$ a term, a *most-specific generalization of $t$ in $T$* is any term $s \in T$ such that (1) $s \preceq t$ and (2) for all $s' \in T$ with $s \preceq s' \preceq t$, $s'$ is a variant of $s$.

## 2 Quantifier Instantiation for Finite Model Finding

Our finite model finding method has been developed so that it can be tightly integrated into a multi-theory version of the DPLL($T$) architecture [12]. A description of our model finder in terms of that architecture is provided in [13]. For the purposes of this paper, it is enough to give a high-level, stand-alone description of the basic model finding procedure restricted to general satisfiability problems (with no theories).

We can ignore the background theory $T$ in this paper because any set of ground formulas generated by the model finder can be purified Nelson-Oppen-style into one set $F_T$ of formulas built only with symbols of $T$ and free constants, and one set $F$ built only with free symbols. After adding to $F_T$ and $F$ a suitable set of equality constraints over their shared free constants, as prescribed by the Nelson-Oppen combination method, if $F_T$ and $F$ are satisfiable their respective models can be always amalgamated into a model of the original problem. Given our restriction on the quantifiers of the input problem, the model finder never needs to instantiate over the sorts of the theory $T$. So we can focus here just on finding models for non-theory formulas.

Without loss of generality, we consider only input problems that are the union $G \cup Q$ of a set $G$ of ground $\Sigma$-formulas and a set $Q$ of non-ground universal $\Sigma$-sentences for some finite signature $\Sigma$. Moreover, $G$ contains a term of sort $S$ for each $S \in \Sigma^{\mathrm{s}}$. We fix $G$, $Q$ and $\Sigma$ as above for the rest of the section.

**Basic model finding procedure** A basic version of the procedure, which is parametrized by a *quantifier instantiation heuristic* $\mathcal{H}$, works as follows:

1. if $G$ is unsatisfiable return "unsat"; else, find a satisfying arrangement $E^*$ for $G$
2. for each sort $S \in \Sigma^{\mathrm{s}}$, let $\mathbf{V}_S$ be the set of $S$-values of $E^*$; let $\mathbf{V} = \bigcup_{S \in \Sigma^{\mathrm{s}}} \mathbf{V}_S$
3. using $\mathcal{H}$ choose a set $I_{\mathbf{x}}$ of *valuations*, substitutions from $\mathbf{x}$ to $\mathbf{V}$, for each $\forall \mathbf{x}\, \varphi \in Q$
4. if the union of all sets $I_{\mathbf{x}}$ is empty, return "sat"; otherwise, for each $\forall \mathbf{x} \varphi \in Q$, add the instances $\{\varphi\sigma \mid \sigma \in I_{\mathbf{x}}\}$ to $G$, and go to Step 1

Step 1 is achieved in our model finder with a novel satisfiability solver, also described in [13], for ground formulas with finite cardinality constraints (FCC) on their sorts. If $G$ is satisfiable, the FCC solver finds a model of $G$ where each sort has a domain of minimal size. As in other model finders, this is done to minimize the number of possible instances of the formulas in $Q$. Step 2 is a by-product of the congruence closure procedure used by the FCC solver: to construct each $\mathbf{V}_S$ it is enough to collect the representatives of the congruence classes computed for $S$. We provide more details on this in Section 2.1. The heuristic $\mathcal{H}$ should be such that whenever $\bigcup_{\forall \mathbf{x} \varphi \in Q} I_{\mathbf{x}}$ is empty $\mathcal{M}$ satisfies $Q$ as well. We discuss how the sets $I_{\mathbf{x}}$ are constructed in Section 2.2.

We represent normal models using the following data structure parametrized by the sets of $S$-values $\mathbf{V}_S$ for some arrangement $A$.

**Definition 1 (Defining map).** *Let $f^{S_1 \cdots S_n S} \in \Sigma^{\mathrm{f}}$ and let $x_1, \ldots, x_n$ be distinct variables of respective sort $S_1 \ldots, S_n$. A* defining map *for $f$ is a finite set $\Delta_f$ of well-sorted (directed) equations of the form $f(t_1, \ldots, t_n) \approx v$ with $v \in \mathbf{V}_S$ and $t_i \in \{x_i\} \cup \mathbf{V}_{S_i}$ for $i = 1, \ldots, n$, satisfying the following requirements.*

1. *If $t_1 \approx v_1, t_2 \approx v_2 \in \Delta_f$ with $v_1 \neq v_2$ and $t_1$ and $t_2$ have an mgu $\sigma$, then $\sigma$ is non-empty and $t_1\sigma = v \in \Delta_f$ for some $v$.*
2. *$f(x_1, \ldots, x_n) \approx v \in \Delta_f$ for some $v$.*

*A $\Sigma$-map is a set $\Delta = \bigcup_{f \in \Sigma^{\mathrm{f}}} \Delta_f$ where each $\Delta_f$ is a defining map for $f$.*

By construction of $\Delta$, every *flat term*, i.e., every $\Sigma$-term $t = f(v_1, \ldots, v_n)$ with $v_1, \ldots, v_n \in \bigcup_S \mathbf{V}_S$, has exactly one most specific generalization $s$ among the left-hand sides of equalities in $\Delta_f$. The existence of $s$ is guaranteed by Point 2 in Definition 1; its uniqueness by Point 1. The *value of $t$ in* $\Delta$ is the value $v$ in the (unique) equality $s \approx v \in \Delta_f$.

Intuitively, a $\Sigma$-map $\Delta$ represents a normal model $\mathcal{M}$ where each sort $S$ is interpreted as the term set $\mathbf{V}_S$ and each function symbol $f^{S_1 \cdots S_n S}$ is interpreted as the function $f^{\mathcal{M}}$ mapping every $(v_1, \ldots, v_n) \in S_1^{\mathcal{M}} \times \cdots \times S_n^{\mathcal{M}}$ to the value of $f(v_1, \ldots, v_n)$ in $\Delta$.

**Proposition 1.** *Let $\Delta$ be a $\Sigma$-map.*

1. *$\Delta$ induces a unique $\Sigma$-structure $\mathcal{M}_\Delta$ modulo isomorphism.*
2. *The satisfiability of universal $\Sigma$-sentences in $\mathcal{M}_\Delta$ is decidable.*
3. *Every normal model of $G$ is induced by a $\Sigma$-map.*

We omit the simple proof of this proposition. For Point 2, we just observe that ground terms can be evaluated in $\mathcal{M}_\Delta$ bottom-up by computing the value in $\Delta$ of flat terms $f(v_1,\ldots,v_n)$. That evaluation allows one to decide ground satisfiability in $\mathcal{M}_\Delta$ in the obvious way. Since every domain of $\mathcal{M}_\Delta$ is finite, the ground satisfiability procedure can be extended to universal $\Sigma$-sentences by exhaustive instantiation of their quantifiers by all values of the corresponding sort.

We rely on normal models constructed from $\Sigma$-maps to be able to check the satisfiability of our problem $G \cup Q$ without having to generate all ground instances of its quantified formulas.

### 2.1 Constructing Normal Models

Given an arrangement $A$ for the ground portion $G$ of our input problem we wish to construct a normal model $\mathcal{M}$ of $G$ that satisfies the quantified portion $Q$ as well. We will refer to $\mathcal{M}$ as a *candidate model (of $G \cup Q$)*. We do this in concrete by building a $\Sigma$-map from $A$ following a strategy that tries to maximize the number of satisfied ground instances of formulas in $Q$. For each function symbol $f$ in $\Sigma$, we start building its defining map $\Delta_f$ by putting in $\Delta_f$ the equality $f(v_1,\ldots,v_n) \approx v_0$ for each term $t_0$ of the form $f(t_1,\ldots,t_n)$ in $G$ where $v_i = \mathsf{v}_A(t_i)$ for $i = 0,\ldots,n$.

Collecting these equalities may produce only a partial definition for $f$. To complete it so that the corresponding $\Sigma$-structure satisfies $G$, one can use arbitrary output values for the remaining input tuples. Previous approaches such as the model-based quantifier instantiation approach implemented in the Z3 SMT solver [8] choose the same default values for all input tuples unconstrained by $A$. Such choices may lead to an infinite series of model checking steps and subsequent instantiations of $Q$ if the *wrong* default values are chosen. We too use default values, but we select them in a more disciplined way, inspired by the instantiation heuristics used in the iProver theorem prover [11]. The main idea is to use the valuation of certain ground terms to guide the selection of default values for function symbols.

Similarly to iProver, we attempt to lift the model of a *ground abstraction* of quantified formulas. We first associate to each sort $S$ a distinguished ground $\Sigma$-term $e^S$ of $G$, which we will write ambiguously here just as $e$ when convenient. Let $\sigma_e$ be the substitution mapping all variables of sort $S$ to $e^S$ for each sort $S$. For all $f^{S_1\cdots S_n S} \in \Sigma^{\mathrm{f}}$, fix $n$ distinct variables $x_1,\ldots,x_n$ of respective sort $S_1 \ldots, S_n$. Then, for all ground $\Sigma$-terms $f(t_1,\ldots,t_n)$, let

$$f(t_1,\ldots,t_n)^\forall = f(u_1,\ldots,u_n)$$

where $u_i = x_i$ if $t_i = e$, and $u_i = \mathsf{v}_A(t_i)$ otherwise, for $i = 1,\ldots,n$. To guide the construction of a $\Sigma$-map, instead of starting with $G$ we start with $\widehat{G} = G \cup \{\varphi\sigma_e \mid \forall \mathbf{y}\, \varphi \in Q\}$ .

Once we find a satisfying arrangement $A$ for $\widehat{G}$, we look at the values it gives to the terms containing the distinguished terms $e$ in order to determine the choice of default values for the function symbols. As a very simple example, suppose $Q = \{\forall y\, f(g(y)) \approx h(a,y)\}$, $\widehat{G} = G \cup \{f(g(e)) \approx h(a,e)\}$, and suppose $A$ is a satisfying arrangement for $\widehat{G}$ such that $\mathsf{v}_A(g(e)) = v$ and $\mathsf{v}_A(h(a,e)) = u$. To complete the defining map $\Delta_g$ for $g$ we use $v$ as the *default* value for $g$, that is, we add the equation $g(x_1) \approx v$ to $\Delta_g$. Similarly, we add the equation $h(a,x_2) \approx u$ to $\Delta_h$. The rationale for this choice is that, together with $f(v) \approx u$ in $\Delta_f$, this will guarantee in this case that the corresponding normal model satisfies $Q$. Of course, this heuristic is not always successful in finding a satisfying model for $Q$ right away. We describe later the corrective measures we take to find a *better* model, that is, one that falsifies fewer ground instances of formulas in $Q$.

The general procedure for constructing a $\Sigma$-map is the following.

**Model construction procedure** Assuming that $\widehat{G} = G \cup \{\varphi\sigma_e \mid \forall \mathbf{y}\, \varphi \in Q\}$ is satisfiable, let $A$ be a satisfying arrangement for it.

1. select a subset $T$ of $\mathbf{T}_{\widehat{G}}$.
2. for each $f \in \Sigma^{\mathrm{f}}$,
   (a) let $D_1 = \{f(\mathsf{v}_A(t_1),\dots,\mathsf{v}_A(t_n)) \approx \mathsf{v}_A(t) \mid t \in \mathbf{T}_{\widehat{G}}, t = f(t_1,\dots,t_n)\}$
   (b) let $D_2 = \{f(t_1,\dots,t_n)^{\forall} \approx \mathsf{v}_A(t) \mid t \in T, t = f(t_1,\dots,t_n)\}$
   (c) let $\Delta_f = D_1 \cup D_2$ and let $\{t_i \approx v_i\}_{0 \le i \le m}$ be an arbitrary enumeration of $\Delta_f$; for all $t_i \approx v_i$, $t_j \approx v_j$ that are unifiable with mgu $\sigma$, if $t_i\sigma$ does not already occur as a left-hand side in $\Delta_f$, add $t_i\sigma \approx v_i$ to $\Delta_f$
   (d) unless $f(x_1,\dots,x_n)$ already occurs as a left-hand side in $\Delta_f$, add $f(x_1,\dots,x_n) \approx v$ for some arbitrary value $v$ of the same sort
3. let $\Delta = \bigcup_{f \in \Sigma^{\mathrm{f}}} \Delta_f$ ■

**Proposition 2.** *The set $\Delta$ constructed by the procedure above is a $\Sigma$-map. Moreover, the $\Sigma$-structure $\mathcal{M}$ induced by $\Delta$ is a normal model of $\widehat{G}$.*

The first step of the model construction procedure is intended to choose a selection of terms containing the distinguished terms $e$. This selection is driven by the arrangement $A$ itself and the way it satisfies the formulas of $G$. It is currently defined as follows.

Let $A$ be a satisfying arrangement for $\widehat{G}$. For all $\psi = \forall \mathbf{y}\, \varphi \in Q$, a ground formula $\varphi'$ is *selectable for* $\psi$ if $A \models \varphi'$ and $\varphi' \models \varphi\sigma_e$. We have a strategy that chooses a selectable formula $\mathsf{sel}(\varphi)$ for each $\psi = \forall \mathbf{y}\, \varphi \in Q$ and then selects all terms in $\mathsf{sel}(\varphi)$. The set $T$ in Step 1 of the model construction procedure is the collection of all these selected terms. The formula $\mathsf{sel}(\varphi)$ is extracted from $\varphi\sigma_e$ itself. For formulas $\varphi$ in CNF it is simply a conjuction of literals, with each literal coming from a conjunct of $\varphi\sigma_e$.

*Example 1.* Say $Q = \{\forall y\, (f(y) \not\approx g(y) \vee h(y) \not\approx b)\}$ and

$$\widehat{G} = \{g(b) \approx a,\, h(a) \approx b,\, h(b) \approx b,\, a \approx f(a)\} \cup \{f(a) \not\approx g(a) \vee h(a) \not\approx b\}$$

where all terms have the same sort and $e = a$. The congruence closure $E^*$ of the set $E$ of equalities in $\widehat{G}$ extends to an arrangement $A$ that satisfies $f(a) \not\approx g(a)$. With $A$ we would select $f(a) \not\approx g(a)$ for $Q$'s only formula, with selected terms $f(a)$ and $g(a)$.

Assuming the values of $A$ are $\{a, b, g(a)\}$, a $\Sigma$-map constructed from $A$ could be

$$\begin{aligned}\Delta = \{a \approx a\} \cup \{b \approx b\} \cup \{g(a) \approx g(a),\, g(b) \approx a,\, g(x_1) \approx g(a)\} \cup \\ \{h(a) \approx b,\, h(b) \approx b,\, h(x_1) \approx b\} \cup \{f(a) \approx a,\, f(x_1) \approx a\}\;.\end{aligned}$$

The $\Sigma$-structure induced by $\Delta$ *almost* satisfies $Q$. It does not for falsifying the instance $f(b) \not\approx g(b) \vee h(b) \not\approx b$. Adding that instance to $\widehat{G}$, we can construct an arrangement like $A$ but with the additional value $f(b)$. This can lead to a $\Sigma$-map $\Delta' = \Delta \cup \{f(b) \approx f(b)\}$ whose induced $\Sigma$-structure does satisfy $G \cup Q$. ■

```
proc eval$(\Delta,t,\sigma)$ $\equiv$ match $t$ with
    | $f(t_1,\ldots,t_n)$ $\rightarrow$ for $j = 1,\ldots,n$ let $(v_j,X_j) = \mathsf{eval}(\Delta,t_j,\sigma)$
                               choose a critical argument subset $C$ of $\{1,\ldots,n\}$
                               return $(f^{\mathcal{M}_\Delta}(v_1,\ldots,v_n), \bigcup_{i\in C} X_i)$
    | $x$ $\rightarrow$ return $(\sigma(x), \{x\})$
```

**Fig. 1.** The eval procedure. $\mathcal{M}_\Delta$ is the model induced by $\Delta$.

### 2.2 Checking Models

As mentioned earlier, once we have a candidate model, i.e., a normal model $\mathcal{M}$ satisfying the ground formulas in $G$, a straightforward way to check that it satisfies the quantified formulas in $Q$ is to check *all* of their ground instances over the finitely many values $\mathbf{V}$ of $\mathcal{M}$. Since a universal formula with $n$ quantified variables each ranging over a domain of size at least $k$ has at least $n^k$ such instances, this is feasible in practice only when both $n$ and $k$ are small.

To increase the scalability of our model finding method we have developed a technique that identifies entire sets of instances satisfiable in $\mathcal{M}$ without actually generating and checking those instances individually. Since the technique is based on the model $\mathcal{M}$ (actually, on the $\Sigma$-map that represents $\mathcal{M}$), we will refer to it as the *model-based approach*, as opposed to the *naive approach* consisting of generating and checking every possible ground instances.

The main idea of the model-based approach is to determine the satisfiability in $\mathcal{M}$ of some ground instance $\varphi\sigma$ of a quantified formula $\forall\mathbf{x}\,\varphi \in Q$, generalize $\varphi\sigma$ to a whole set of $F$ of instances equisatisfiable with $\varphi\sigma$ in $\mathcal{M}$, and then look for further instances only outside that set. The set $F$ is computed by identifying which variables of $\varphi$ actually matter in determining the satisfiability of $\varphi\sigma$. Technically, for each $\psi = \forall\mathbf{x}\,\varphi \in Q$, valuation $\sigma = \{\mathbf{x} \mapsto \mathbf{v}\}$ into $\mathbf{V}$, and ground instance $\varphi' = \varphi\sigma$ of $\psi$, if $\mathcal{M} \models \varphi'$ we compute a partition of $\mathbf{x}$ into $\mathbf{x}_1$ and $\mathbf{x}_2$ and a corresponding partition of $\mathbf{v}$ into $\mathbf{v}_1$ and $\mathbf{v}_2$ such that $\mathcal{M} \models \forall\mathbf{x}_2\,\varphi\{\mathbf{x}_1 \mapsto \mathbf{v}_1\}$; similarly, if $\mathcal{M} \not\models \neg\varphi'$ we compute a partition such that $\mathcal{M} \not\models \forall\mathbf{x}_2\,\neg\varphi\{\mathbf{x}_1 \mapsto \mathbf{v}_1\}$. In either case, we then know that all ground instances of $\varphi\{\mathbf{x}_1 \mapsto \mathbf{v}_1\}$ over $\mathbf{V}$ are equisatisfiable with $\varphi'$ in $\mathcal{M}$, and so it is enough to consider just $\varphi'$ in lieu of all them. We will refer to the elements of $\mathbf{x}_1$ above as a set of *critical variables for* $\varphi$ *(under* $\sigma$*)*—although strictly speaking this is a misnomer as we do not insist that $\mathbf{x}_1$ be minimal.

**Checking and generalizing ground instances** Treating quantifier-free formulas as Boolean terms (which evaluate to either true or false in a $\Sigma$-structure depending on whether they are satisfied by the model or not), we developed a general procedure that, given the $\Sigma$-map of a candidate model $\mathcal{M}$, a term $t$, and a valuation $\sigma$ of $t$'s variables, computes and returns both the value of $t\sigma$ in $\mathcal{M}$ and a set of critical variables for $\sigma$.

The procedure, defined recursively over the input term and assuming a prefix form for the logical operators as well, is sketched in Figure 1. When evaluating a non-variable term $f(t_1,\ldots,t_n)$, eval determines a *critical argument subset* $C$ for it. This is a subset of $\{1,\ldots,n\}$ such that the term $f(s_1,\ldots,s_n)$ denotes a constant function in $\mathcal{M}$ where each $s_i$ is the value computed by eval for $t_i$ if $i \in C$, and is a unique variable otherwise. If $f$

is a logical symbol, the choice of $C$ is dictated by the symbol's semantics. For instance, for $\approx(t_1,t_2)$, $C$ is $\{1,2\}$; for $\vee(t_1,\dots,t_n)$, it is $\{1,\dots,n\}$ if the disjunction evaluates to false; otherwise, it is $\{i\}$ if $t_i$ is the one with the best set $X_i$ of critical variables among the elements of $\{t_1,\dots,t_n\}$ that evaluate to true, where "best" is defined in term of another heuristic measure. If $f$ is a function symbol of $\Sigma$, eval computes $C$ by first constructing a custom index data structure for interpreting applications of $f$ to values. The key feature of this data structure is that it uses information on the sets $X_1,\dots X_n$ to choose an evaluation order for the arguments of $f$.

For space constraints, we give just a concrete example of how the choice of $C$ is made. Say eval, given the term $t = f(g(x,y,z),v_2,h(x))$, computes the values $v_1,v_2,v_3$ and the critical variable sets $\{x,y,z\}$, $\emptyset$, $\{x\}$ for the three arguments of $f$, respectively. With those sets, it will use the evaluation order $(2,3,1)$ for those arguments—meaning that the second argument is evaluated first, then the third, etc. Using the index data structure, it will first determine if $f(x_1,v_2,x_3)$ has a constant interpretation in $\mathcal{M}$. If so, then the evaluation depends on no variables and the returned set of critical variables for $t$ will be $\emptyset$. Otherwise, if $f(x_1,v_2,v_3)$ has a constant interpretation in $M$, then the evaluation depends on $\{x\}$, or else it depends on the entire variable set $\{x,y,z\}$.

The next example gives more details on the whole process of generalizing a ground instance to a set of ground instances equisatisfiable with it in the given model.

*Example 2.* Let $Q = \{\forall y \forall z\, f(z) \approx g(y,b) \vee h(y,z) \not\approx b\}$ and $\widehat{G} = \{f(a) \approx a, f(b) \approx b, g(a,a) \approx b, h(a,a) \approx b, f(a) \approx g(a,b) \vee h(a,a) \not\approx b\}$ where $a$ is the only distinguished ground term. Consider a $\Sigma$-map $\Delta$ constructed as in Example 1 and containing the following defining maps:

$$\Delta_g = \{g(a,b) \approx a,\ g(a,a) \approx b,\ g(x_1,b) \approx a,\ g(x_1,x_2) \approx b\}$$

$$\Delta_f = \{f(b) \approx b,\ f(a) \approx a,\ f(x_1) \approx a\} \qquad \Delta_h = \{h(a,a) \approx b,\ h(x_1,x_2) \approx b\}$$

The table below shows the bottom-up calculation performed by eval on the formula $\varphi = f(z) \approx g(y,b) \vee h(y,z) \not\approx b$ with $\Delta$ above and $\sigma = \{y \mapsto a, z \mapsto a\}$.

| input | output | critical arg. subset |
|---|---|---|
| $z$ | $(a,\{z\})$ | // |
| $y$ | $(a,\{y\})$ | // |
| $b$ | $(b,\emptyset)$ | // |
| $f(z)$ | $(a,\{z\})$ | $\{1\}$ |
| $g(y,b)$ | $(a,\emptyset)$ | $\{2\}$ |

| input | output | critical arg. subset |
|---|---|---|
| $h(y,z)$ | $(b,\emptyset)$ | $\emptyset$ |
| $f(z) \approx g(y,b)$ | (true, $\{z\}$) | $\{1,2\}$ |
| $h(y,z) \not\approx b$ | (false, $\emptyset$) | $\emptyset$ |
| $f(z) \approx g(y,b) \vee h(y,z) \not\approx b$ | (true, $\{z\}$) | $\{1\}$ |

For most entries in the table the evaluation is straightforward. For a more interesting case, consider the evaluation of $g(y,b)$. First, the arguments of $g$ are evaluated, respectively to $(a,\{y\})$ and $(b,\emptyset)$, but with evaluation order $(2,1)$. After evaluating $y$ to $b$, using an indexing data structure built from $\Delta_g$ for the evaluation order $(2,1)$, eval is able to quickly determine that the term $g(x_1,b)$ has constant value $a$ for all $x_1$. Hence it returns an empty set of critical variables for $g(y,b)$.

Similarly, the fact that eval returns (true, $\{z\}$) for the original input formula $\varphi$ and the valuation $\sigma = \{y \mapsto a, z \mapsto a\}$, means that it was able to determine that all ground instances of $\varphi\{z \mapsto a\} = (f(a) \approx g(y,b) \vee h(y,a) \not\approx b)$, not just the instance $\varphi\sigma$, are satisfied in $\mathcal{M}$. Our model finder can then use this information to completely avoid generating and checking those instances. ■

```
**proc** choose_instances$(\Delta, \varphi, \mathbf{x})$ $\equiv$
  $I_{\mathbf{x}} := \emptyset$;  $t_{next} := \mathbf{v}_{min}$ where $\mathbf{v}_{min}$ is the minimum of $\mathbf{V}_{\mathbf{x}}$
  **do**
    $t := t_{next}$
    $(v, \{x_{i_1}, \dots, x_{i_m}\}) :=$ eval$(\Delta, \varphi, \{\mathbf{x} \mapsto t\})$
    **if** $v =$ false **then** $I_{\mathbf{x}} := I_{\mathbf{x}} \cup \{\{\mathbf{x} \mapsto t\}\}$
    $t_{next} :=$ next$_i(t)$ where $i$ is the minimum of $\{i_1, \dots, i_m, n+1\}$
  **while** $t_{next} \neq t$
  **return** $I_{\mathbf{x}}$
```

**Fig. 2.** The choose_instances procedure. We assume that $\mathbf{x} = (x_1, \dots, x_n)$.

**Collecting ground instances** For any given quantified formula $\psi$, the eval procedure allows us to identify a set of instances over $\mathbf{V}$ that can be represented by a single one, as far as satisfiability in the candidate model $\mathcal{M}$ is concerned. The next question then is how to generate a set $I$ of instances that together represent *all* instances of $\psi$ over $\mathbf{V}$ that are falsified by $\mathcal{M}$. This kind of exhaustiveness is crucial because it allows us to conclude correctly that $\mathcal{M} \models \psi$ by just checking that $I$ is empty.

We present a procedure that relies on eval for computing the set $I$ above or, rather, a set of valuations for generating the elements of $I$ from $\psi$. The procedure is fairly unsophisticated and quite conservative in its choice of representative instances, which makes it very simple to implement and prove correct. Its main shortcoming is that it does not take full advantage of the information provided by eval, and so may end up producing more representative instances than needed in many cases. The development of a more selective procedure is left to future work.

Let $\psi = \forall \mathbf{x}\, \varphi \in Q$ with $\mathbf{x} = (x_1, \dots, x_n)$. For $i = 1, \dots, n$, let $S_i$ be the sort of $x_i$ and let $\mathbf{V}_{\mathbf{x}} = \mathbf{V}_{S_1} \times \cdots \times \mathbf{V}_{S_n}$. For each $S \in \{S_1, \dots, S_n\}$, let $<_S$ be an arbitrary total ordering over the values $\mathbf{V}_S$ of sort $S$. Let $<$ be the *reversed lexicographic*[4] extension of these orderings to the tuples in $\mathbf{V}_{\mathbf{x}}$ and observe that $\mathbf{V}_{\mathbf{x}}$ is totally ordered by $<$.

For every $\mathbf{v} = (v_1, \dots, v_n) \in \mathbf{V}_{\mathbf{x}}$ let $\mathbf{v}[i]$ denote the $i^{th}$ element of $\mathbf{v}$ and let next$_i(\mathbf{v})$ denote the smallest tuple $\mathbf{u}$ wrt. $<$ such that $\mathbf{v}[j] <_{S_j} \mathbf{u}[j]$ for some $j \geq i$, if such tuple exists, and denote $\mathbf{v}$ itself otherwise (including when $i > n$). For instance, with $n = 3$, $S_1 = S_2 = S_3$ and $\mathbf{V}_{S_1} = \{a, b\}$ with $a <_{S_1} b$, we have that next$_2(a,a,a) = (a,b,a)$, next$_2(a,b,a) = (a,a,b)$, next$_3(a,a,b) = (a,a,b)$, and next$_3(a,b,b) = (a,b,b)$. Note that $\mathbf{v} \leq$ next$_i(\mathbf{v})$ for all $\mathbf{v}$.

The instantiation heuristic $\mathcal{H}$ used in the model finding procedure presented in Section 2 is implemented by the procedure choose_instances described in Figure 2, which takes in a quantifier-free formula $\varphi$ with variables $\mathbf{x}$ and returns a set $I_{\mathbf{x}}$ of valuations $\sigma$ for $\mathbf{x}$ such that $\mathcal{M} \not\models \varphi\sigma$. At each execution of its loop the procedure implicitly determines with eval a set of $I$ of instances of $\varphi$ that are equisatisfiable with $\varphi\{\mathbf{x} \mapsto \mathbf{v}\}$ in $\mathcal{M}$, where $\mathbf{v}$ is the tuple stored in the program variable $t$. The next value $t_{next}$ for $t$ is a greater tuple chosen to maintain the invariant that all the tuples between $t$ and $t_{next}$ generate instances of $\varphi$ that are in $I$. To see that, it suffices to observe that these tuples differ from

[4] This is defined similarly to the standard lexicographic extension except that the last component of a tuple is the most significant one, then the last but one, and so on.

$t$ only in positions that correspond to non-critical variables of $\varphi$, namely those before position $i$ where $x_i$ is the first critical variable of $\varphi$ in the enumeration $x_1, \ldots, x_n$. This observation is the main argument in the proof of the following result.

**Proposition 3.** *Let $\mathbf{v}_0, \ldots, \mathbf{v}_m$ be all values successively taken by the variable $t$ in the loop of* choose_instances. *Let $v_{max}$ be the maximum element of $\mathbf{V}_\mathbf{x}$. Then for all $i = 1, \ldots, m$,*

1. *$\mathbf{v}_{i-1} < \mathbf{v}_i$,*
2. *for all $\mathbf{u}$ with $\mathbf{v}_{i-1} \leq \mathbf{u} < \mathbf{v}_i$, $\mathcal{M} \models \varphi\{\mathbf{x} \mapsto \mathbf{u}\}$ iff $\mathcal{M} \models \varphi\{\mathbf{x} \mapsto \mathbf{v}_{i-1}\}$,*
3. *for all $\mathbf{u}$ with $\mathbf{v}_m \leq \mathbf{u} \leq \mathbf{v}_{max}$, $\mathcal{M} \models \varphi\{\mathbf{x} \mapsto \mathbf{u}\}$ iff $\mathcal{M} \models \varphi\{\mathbf{x} \mapsto \mathbf{v}_m\}$.*

For this proposition it follows immediately that $\mathcal{M} \models \forall \mathbf{x}\, \varphi$ if and only if the set $I_\mathbf{x}$ returned by choose_instances$(\Delta, \varphi, \mathbf{x})$ is empty.

We remark that, for our model finding purposes, there is no need for the procedure choose_instances to compute the full set $I_\mathbf{x}$ once it contains at least one valuation. Any non-empty subset would suffice to trigger a (more incremental) revision of the current candidate model $\mathcal{M}$. That said, our current implementation does compute the whole set and adds all the corresponding instances to $Q$ before recomputing another model for it. Our initial experiments show that computing and using one valuation at a time is significantly worse for overall performance than computing and using the full set $I_\mathbf{x}$.

### 2.3 Enhancements Based on Heuristic Instantiation

Many SMT solvers rely on heuristic instantiation methods for finding unsatisfiable instances for quantified formulas. These methods typically use *E-matching* techniques [3] to generate heuristically relevant instances, which are based on matching distinguished terms, called *triggers*, with ground terms in the problems. We found that $E$-matching can be helpful in our model finder as well, even for satisfiable problems.

**Enhanced model finding procedure** Our original heuristic $\mathcal{H}$ from Section 2 for quantifier instantiation can be enhanced with $E$-matching to a heuristic $\mathcal{H}'$ as follows:

1. choose a set of triggers $T_\psi$ for each $\psi \in Q$, and return valuations based on $E$-matching for $(T_\psi, G)$
2. if no such instances exist, apply the original $\mathcal{H}$.

Applying $E$-matching helps the model finder detect the unsatisfiability of its input formulas more promptly in cases where a conflict is easily identifiable. Furthermore, it may also accelerate the discovery of a model for satisfiable input problems, since the instances it generates can help rule out bad choices of candidate models more quickly.

Recall that in the basic model finding procedure, quantifier instantiation is applied *after* finding a model of the ground formulas $G$ of minimal size. By waiting to apply quantifier instantiation until after model minimization, we also avoid pitfalls common to $E$-matching-based procedures such as, for instance, *matching loops* where certain terms get generated at every instantiation round. Since only a finite number of terms exist for a given cardinality bound on a sort, our approach guarantees that $E$-matching will eventually rule out the given bound, or terminate with no instances produced.

Most $E$-matching techniques generate triggers automatically, but in fairly uninformed ways, typically choosing every applicable term (or set of terms) in a quantified formula $\psi$ as a trigger. In our model finding method, the selection heuristic described in Section 2.1 can be used as a criterion for trigger generation by using first as triggers the terms that were selected for the construction of the current candidate mode. The intuition is that if we are basing the satisfiability of $\psi$ on the default values given for a function symbol $f$, then we need only be concerned with possible exceptions to those defaults. Our current implementation follows this criterion.

## 3 Experimental Results

The model finding method introduced in [13] is implemented within the CVC4 SMT solver. For the present work, we implemented the naive and the model-based approach for quantifier instantiation as alternative configurations of CVC4's finite model finder. We ran experimental comparisons for these approaches on three sets of benchmarks.

First we considered formulas derived from verification conditions generated by DVF [9], a tool used at Intel for verifying properties of security protocols and design architectures, among other applications, comparing configurations of the model finder against CVC4 in native mode (i.e., not using the model finder) and Z3 version 4.1, which we previously found to be the best SMT solver besides CVC4 on these benchmarks [13]. Second, we considered benchmarks from the latest version of the TPTP library (5.4.0), comparing against various automated theorem provers and model finders for first order logic, as well as the two SMT solvers above. Third, we considered a set of SMT benchmarks translated from proof obligations generated by the Isabelle prover, comparing again with CVC4 in native mode and Z3.[5]

In all experiments we used revision 4751 of CVC4 1.0, both in native mode (indicated here as **cvc4**) and in finite model finding mode. The default configuration of the latter (**cvc4+f**) applies naive quantifier instantiation as described in Section 2.2, and no heuristic instantiation. The other model finding configurations use either model-based quantifier instantiation as described in Section 2.2 (**cvc4+fm**), or just heuristic quantifier instantiation as described in Section 2.3 (**cvc4+fi**), or both (**cvc4+fmi**).

The first set of experiments was run on a Linux machine with an 8-core 2.60GHz Intel® Xeon® E5-2670 processor. The second and third on a cluster of 5 identical Linux machines with a 2.4 GHz AMD Opteron 250s and 2 GB of available memory.

**Intel benchmarks** We considered 3 of the 5 classes of benchmarks from [13]; the other two are uninteresting as their problems can be solved quickly by **cvc4+f**. The **agree** class is from [14] while the **apg** and **bmk** classes are verification conditions internal to Intel. The benchmarks contain a variety of SMT theories, including arithmetic, arrays, datatypes, free functions over free sorts and built-in sorts, but with quantifiers limited to free sorts. Both unsatisfiable and satisfiable benchmarks were considered, the latter produced by manually removing necessary assumptions from verification conditions. The results are summarized in Figure 3 for various configurations of CVC4 and for Z3.

[5] The finite model finder, detailed results, and the non-proprietary benchmarks discussed in this section are available at `http://cvc4.cs.nyu.edu/experiments/CADE24-2013/`.

|  | **Sat** |  |  |  |  |  | **Unsat** |  |  |  |  |  |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
|  | **agree**<br>(15) |  | **apg**<br>(17) |  | **bmk**<br>(31) |  | **agree**<br>(139) |  | **apg**<br>(124) |  | **bmk**<br>(83) |  |
| Solver | solved | time | solved | time | solved | time | solved | time | solved | time | solved | time |
| **z3** | 0 | 0 | 0 | 0 | 0 | 0 | **139** | **3.5** | **124** | **9.0** | 83 | 2.5 |
| **cvc4** | 0 | 0 | 0 | 0 | 0 | 0 | 135 | 2.9 | 124 | 10.0 | **83** | **1.7** |
| **cvc4+f** | 15 | 13.7 | 16 | 199.4 | 30 | 1200.1 | 127 | 3772.4 | 118 | 2243.3 | 81 | 1496.5 |
| **cvc4+fi** | **15** | **12.3** | 17 | 492.2 | 30 | 829.5 | 139 | 185.8 | 122 | 338.8 | 83 | 656.7 |
| **cvc4+fm** | 15 | 21.3 | **17** | **209.9** | 31 | 374.2 | 122 | 5007.5 | 120 | 1114.9 | 81 | 827.3 |
| **cvc4+fmi** | 15 | 13.6 | 17 | 220.6 | **31** | **175.5** | 139 | 183.4 | 122 | 336.6 | 83 | 664.9 |

**Fig. 3.** Results for DVF benchmarks. All times are in seconds. Best performances are in bold font.

We show results for the 412 problems from the previous study that were non-trivial for CVC4's model finder.[6] All configurations had a 600s timeout per problem.

For the satisfiable benchmarks, CVC4's model finder is the only tool able to solve at least one. Additionally, through use of model-based quantifier instantiation, it is now able to solve all of them within the timeout. Moreover, the best configuration of the model finder, **cvc4+fmi**, solves each benchmark within 60s.

For the unsatisfiable benchmarks, Z3 is the overall winner, solving all of them within the timeout. Pairing heuristic quantifier instantiation with finite model finding (configurations with **cvc4+*i***) is beneficial, as it even solves four problems that **cvc4** cannot solve. We found that each unsatisfiable problem can be solved by either **cvc4** or **cvc4+fmi**, and in less than 3s. Configuration **cvc4+fmi** solves all unsatisfiable benchmarks within 900s, suggesting that CVC4's model finder makes consistent progress towards answering unsatisfiable on provable DVF verification conditions. Also, that configuration is an order of magnitude faster than **cvc4+f** on unsatisfiable benchmarks solved by each of them. From the perspective of verification tools, the results here seem promising. A feasible strategy for discharging a verification condition would be to first use an SMT solver hoping that it can quickly find it unsatisfiable with $E$-matching techniques; and then resort to finite model finding if needed to either answer unsatisfiable, or produce a model representing a concrete counterexample for the verification condition.

**TPTP benchmarks** For these benchmarks we also compared against Paradox [6] and iProver [11]. Paradox is a MACE-style model finder that uses preprocessing optimizations such as sort inference and clause splitting, among others, and then encodes to SAT the original problem together with increasingly looser constraints on the size of the model. iProver is an automated theorem prover based in the Inst-Gen calculus that can also run in finite model finding mode (**iprover-fm**). In that mode, it incrementally bounds model sizes in a manner similar to MACE-style model finding. However, it encodes the whole problem into the EPR fragment, for which it is a decision procedure. Since these two tools are limited to classical first-order logic with equality, we considered only the unsorted first-order benchmarks of TPTP.

The results for a 30s timeout per benchmark, are shown in Figure 4. CVC4's model finder with exhaustive instantiation (**cvc4+f**) can find 975 benchmarks to be satisfiable. That number goes up to 1025 with model-based quantifier instantiation (**cvc4+fm**). While better than Z3, which finds 888 satisfiable benchmarks, our model finder still

[6] The rest are solved in less than 0.5s by all configurations of the model finder.

| | **paradox** | **iprover** | **iprover-fm** | **z3** | **cvc4** | **cvc4+f** | **cvc4+fm** | **cvc4+fmi** |
|---|---|---|---|---|---|---|---|---|
| **Sat** | **1344** | 995 | 1231 | 888 | 33 | 975 | 1025 | 955 |
| **Unsat** | 1272 | 5556 | 383 | **5934** | 5295 | 2633 | 2754 | 3028 |

**Fig. 4.** Results for 15561 benchmarks taken from the TPTP library, with a 30s timeout. Of these benchmarks, 1995 are known to be satistiable, and 12586 are known to be unsatisfiable.

trails the overall performance of the other provers on these problems. Paradox, the best here, finds 1344 satisfiable benchmarks. We attribute this to the fact that we have implemented none of the advanced preprocessing techniques, such as sort inference and clause splitting, that have been shown to be critical for finding finite models of TPTP benchmarks. Nevertheless, CVC4's model finder is capable solving a handful of benchmarks that neither Paradox nor iProver can solve. In particular, it solves two satisfiable benchmarks with 1.0 difficulty rate, which means that no known ATP system had solved these problems when version of 5.4.0 of the TPTP library was released.

Figure 4 shows also results for unsatisfiable problems. Although these results are not comparable to those achieved by state-of-art theorem provers, such as Vampire and E, we note that Z3 solves the most benchmarks, 5924. Interestingly, an additional 35 unsatisfiable problems with difficulty rating 1.0 were found in this study by Z3, **cvc4**, iProver and **cvc4+fmi**, which respectively solve 21, 6, 4, and 1 of these uniquely.

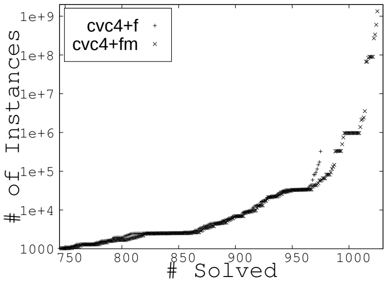

**Fig. 5.** Comparison of satisfiable problems found with and without model-based quantifier instantiation. A point $(x, y)$ on this graph says the configuration solves $x$ benchmarks each with a total of at most $y$ ground problems of quantified formulas.

To further evaluate the impact of model-based quantifier instantiation on our model finder, we recorded statistics on the domain size of quantified formulas in benchmarks solved by its various configurations. We measured the total number of instances for all quantified formulas occurring in a problem (a quantified formula over $n$ variables each with domain size $k$ has $n^k$ instances). For a problem with $d$ total instances, the configuration **cvc4+f** must explicitly generate these $d$ instances, while a model-based configuration may avoid doing so. For these experiments, **cvc4+f** was only able to solve 4 problems having more than 100k instances, the maximum having around 325k instances. On the other hand, **cvc4+fm** was capable of solving 41 problems having more than 100k instances, with the largest having more than 1.3 billion instances. This information is plotted in Figure 5, showing how the model-based instantiation approach improves the scalability of our model finder and allows it to solve benchmarks where exhaustive instantiation is clearly infeasible. We stress that model finders such as Para-

| Sat | **Arrow_Order** | **FFT** | **FTA** | **Hoare** | **NS_Shared** | **QEpres** | **StrongNorm** | **TwoSquares** | **TypeSafe** | **TOTAL** |
|---|---|---|---|---|---|---|---|---|---|---|
| **z3** | 3 | 19 | 24 | 46 | 10 | 49 | 1 | 17 | 11 | 180 |
| **cvc4** | 0 | 9 | 0 | 0 | 0 | 0 | 0 | 8 | 0 | 17 |
| **cvc4+f** | 22 | 138 | 172 | 153 | 56 | 79 | **12** | 59 | 69 | 760 |
| **cvc4+fm** | **26** | 139 | 171 | 151 | 49 | 80 | **12** | 59 | 69 | 756 |
| **cvc4+fmi** | **26** | **151** | **174** | **159** | **60** | **81** | **12** | **60** | **78** | **801** |

| Unsat | **Arrow_Order** | **FFT** | **FTA** | **Hoare** | **NS_Shared** | **QEpres** | **StrongNorm** | **TwoSquares** | **TypeSafe** | **TOTAL** |
|---|---|---|---|---|---|---|---|---|---|---|
| **z3** | **261** | **224** | **765** | **497** | **135** | 236 | **240** | 451 | **325** | **3134** |
| **cvc4** | 199 | 217 | 682 | 456 | 97 | **244** | 231 | **486** | 239 | 2851 |
| **cvc4+f** | 120 | 99 | 298 | 214 | 36 | 105 | 84 | 316 | 132 | 1404 |
| **cvc4+fm** | 102 | 91 | 330 | 246 | 26 | 117 | 80 | 310 | 128 | 1430 |
| **cvc4+fmi** | 155 | 170 | 467 | 328 | 42 | 161 | 97 | 411 | 188 | 2019 |

**Fig. 6.** Results for Isabelle Benchmarks. Numbers of problems solved within 30s.

dox have other ways of handling the explosion in the number of instances, namely by minimizing the number of variables per clause. We expect that coupling these techniques with the model-based techniques used here will lead to additional improvements in the scalability of our model finder.

**Isabelle benchmarks** Recent work has shown that SMT solvers, in particular Z3, are effective at discharging Isabelle proof obligations whose encoding can be represented with theories [5]. Model finding can be useful in Isabelle for debugging and for brute-force proof minimization [4]. More generally, it is useful to interactive theorem provers that are based on heuristically selecting a set of relevant background axioms which might be sufficient to prove a conjecture. In this case, a model finder could be used to quickly identify axiom sets that are not large enough for a given conjecture.

We considered a set of 13,041 benchmarks generated from Isabelle and kindly provided by Sascha Böhme. The benchmarks in this set correspond to both provable and unprovable conjectures.[7] Most of them contain quantifiers, and a significant portion contain integer arithmetic. For many, quantifiers are limited to the free sorts, thus making our finite model finding approach applicable. Since CVC4 does not yet have support for non-linear arithmetic, we report results only for the 11,187 benchmarks that do not contain non-linear arithmetic constraints. Additionally, CVC4 ignored various hints (such as weight values) that were given to Z3 for quantifier instantiation.

The results are shown in Figure 6. In these experiments, using E-matching accelerates the search for models, as **cvc4+fmi** finds more satisfiable problems (810) than both **cvc4+f** (760) and **cvc4+fm** (756). All configurations of CVC4's model finder find many more satisfiable problems than Z3, which finds only 180 of them overall. For unsatisfiable problems, Z3 is the overall winner, solving 3,134 of them, followed by the **cvc4** configuration with 2,851. Interestingly, while **cvc4+fmi** solves only 2,019 unsatisfiable benchmarks, 244 of them are not solved by Z3, and 164 are not solved by **cvc4**.

## 4 Conclusion

We have introduced a few quantifier instantiation techniques for finite model finding in SMT which drastically improve the scalability of our basic model finding procedure and

[7] It is our understanding that these benchmarks are a superset of those discussed in [5].

are useful in various applications. Our experiments show that our model-based quantifier instantiation approach is useful for finding models where exhaustive instantiation is infeasible, and can be improved further by integrating heuristic instantiation in it, especially for unsatisfiable problems.

Future research includes improvements to the instance generation technique in Section 2.2, and further generalizing the approach to the construction of models for built-in theories. We are currently investigating ways to modify the selection heuristics of Section 2.1 to generate candidate models (in some fragments) of the theory of arrays. We plan to investigate further approaches for finding models of formulas with quantifiers ranging over built-in domains such as the integers.

**Acknowledgements** We would like thank Sascha Böhme for proving the Isabelle benchmarks and François Bobot for his help in writing a TPTP front end for CVC4.

## References

1. F. Baader and T. Nipkow. *Term Rewriting and All That*. Cambridge University Press, 1998.
2. P. Baumgartner and C. Tinelli. The Model Evolution calculus as a first-order DPLL method. *Artificial Intelligence*, 172:591–632, 2008.
3. N. Bjørner and L. de Moura. Efficient E-matching for SMT solvers. In *Proceedings of CADE-21*, volume 4603 of *LNCS*, pages 183–198. Springer, 2007.
4. J. C. Blanchette. Personal communication, 2013.
5. J. C. Blanchette, S. Böhme, and L. C. Paulson. Extending Sledgehammer with SMT solvers. In *Proceedings of CADE-23*, volume 6803 of *LNCS*, pages 116–130. Springer, 2011.
6. K. Claessen and N. Sörensson. New techniques that improve MACE-style finite model building. In *CADE-19 Workshop: Model Computation – Principles, Algorithms, Applications*, pages 11–27, 2003.
7. H. Ganzinger and K. Korovin. New directions in instantiation-based theorem proving. In *Proceedings of LICS'03*, pages 55–64. IEEE Computer Society, 2003.
8. Y. Ge and L. de Moura. Complete instantiation for quantified formulas in sat. modulo theories. In *Proceedings of CAV'09*, volume 5643 of *LNCS*, pages 306–320. Springer, 2009.
9. A. Goel, S. Krstic, and R. L. M. Tuttle. SMT-based system verification with DVF. In *Proceedings of SMT'12*, 2012.
10. S. Jacobs. Incremental instance generation in local reasoning. In *Proceedings of CAV'09*, volume 5643 of *LNCS*, pages 368–382. Springer, 2009.
11. K. Korovin. iProver – an instantiation-based theorem prover for first-order logic. In *Proceedings of IJCAR'08*, volume 5195 of *LNCS*, pages 292–298. Springer, 2008.
12. S. Krstić and A. Goel. Architecting solvers for SAT modulo theories: Nelson-Oppen with DPLL. In *Proceeding of FroCoS'07*, volume 4720 of *LNCS*, pages 1–27. Springer, 2007.
13. A. Reynolds, C. Tinelli, A. Goel, and S. Krstić. Finite model finding in SMT. In *Proceedings of CAV'13*, LNCS. Springer, 2013. (Accepted).
14. M. R. Tuttle and A. Goel. Protocol proof checking simplified with SMT. In *Proceedings of NCA'12*, pages 195–202. IEEE Computer Society, 2012.
15. J. Zhang and H. Zhang. SEM: a system for enumerating models. In *Proceedings of IJCAI'95*, pages 298–303, 1995.