Finite Quantification in Hierarchic Theorem Proving

Peter Baumgartner[1] Joshua Bax[1] Uwe Waldmann[2]

NICTA* and Australian National University, Canberra, Australia
MPI für Informatik, Saarbrücken, Germany

Abstract. Many applications of automated deduction require reasoning in first-order logic modulo background theories, in particular some form of integer arithmetic. A major unsolved research challenge is to design theorem provers that are "reasonably complete" even in the presence of free function symbols ranging into a background theory sort. In this paper we consider the case when all variables occurring below such function symbols are quantified over a finite subset of their domains. We present a non-naive decision procedure for background theories extended this way on top of black-box decision procedures for the EA-fragment of the background theory. In its core, it employs a *model-guided* instantiation strategy for obtaining pure background formulas that are equi-satisfiable with the original formula. Unlike traditional finite model finders, it avoids exhaustive instantiation and, hence, is expected to scale better with the size of the domains. Our main results in this paper are a correctness proof and first experimental results.

1 Introduction

Many applications of automated deduction require reasoning in first-order logic modulo background theories, in particular some form of integer arithmetic. A major unsolved research challenge is to design theorem provers that are "reasonably complete" for quantified formulas, in particular in presence of free function symbols ranging into a background theory sort ("free BG-sorted operators", for short). Such formulas arise frequently when reasoning on data structures with specific properties, e.g., *symmetric* arrays over integers and *sorted* lists over integers. Modelling such data structures is easy when full quantification and free integer-sorted function symbols are available to axiomatize the array access function and the list head function respectively.

Unfortunately, (refutationally) complete theorem proving in the presence of free BG-sorted operators is intractable in general. For instance, just adding one free predicate symbol to linear integer arithmetic results in a Π^1_1-hard validity problem [12]. Theorem proving approaches hence have to circumvent this problem in one way or the other. On the one hand, SMT-solvers [18] generally use instantiation heuristics [10, 16] for reducing the input problem to a quantifier-free one, and these are complete only in rather restricted cases [11]. On the other hand, approaches rooted in first-order theorem proving either are incomplete; do not accept free BG-sorted operators at all [13, 21, 9, 5] or, are complete only for certain fragments or under certain conditions [3, 1, 14, 6, 7].

In practice, lack of completeness is a major concern in, e.g., software verification applications, which frequently require disproving non-valid proof obligations. In such

* NICTA is funded by the Australian Government through the Department of Communications and the Australian Research Council through the ICT Centre of Excellence Program.

cases, incomplete theorem provers run out of resources or report "unknown" instead of detecting non-validity. We address this problem by working with quantification over *finite* segments of the background sorts, e.g., the integers. Our underlying methodology assumes that from a user's point of view, data structures over the integers can often be supplanted by data structures over reasonably large finite segments of the integers, say, from −Maxint to +Maxint, as good-enough approximations. As no other restrictions apply, our method should be widely applicable in practice. Our method is also refutationally sound wrt. the standard semantics. That is, if our algorithm determines unsatisfiability wrt. finite domains, the given clause set is also unsatisfiable wrt. unbounded domains. Because of that, our approach can be seen as an extension of current quantifier instantiation heuristics by being able to determine satisfiability wrt. finite domains.

If all quantifiers range over finite domains, decidability can be recovered in a trivial way by exhaustive instantiation and calling a suitable SMT-solver afterwards. Of course, this naive approach does not scale with the domain size and cannot be expected to work well in practice. This problem has often been observed in the context of finite-model finding [22, 23, 15, 8, 4, 20, 19]. While our method is also based on instantiation, it is (often) far less prolific than the naive method.

More precisely, our method accepts as input a set of *finitely quantified clauses*. A clause is finitely quantified if every variable occurring below a free BG-sorted operator is quantified over a finite segment of its domain. The core idea is to give the free BG-sorted operators a *default* interpretation that is then stepwise refined. This default interpretation maps every free BG-sorted operator to a constant function, and refinements are done by finding exceptions to that in a conflict-driven way. After each refinement, the given clause set is transformed into a certain form whose satisfiability can be decided by existing reasoners in a black-box fashion. Suitable reasoners are, e.g., theorem provers implementing hierarchic superposition [3, 7] and, with one more simple transformation step, SMT-solvers for the EA-fragment of the background theory. The procedure stops after finitely many (hopefully few) refinement steps, either with a representation of a model or a set of ground instances obtained from exceptions which demonstrates the unsatisfiability of the given clause set.

We preview our method with an example. Let N be the following clause set:

(1) $\mathsf{read}(\mathsf{write}(a, i, x), i) \approx x$ (4) $1 \leq \mathsf{m} \wedge \mathsf{m} < 1000$
(2) $\mathsf{read}(\mathsf{write}(a, i, x), j) \approx \mathsf{read}(a, j) \vee i \approx j$ (5) $\mathsf{read}(\mathsf{a}, \mathsf{m}) < \mathsf{read}(\mathsf{a}, \mathsf{m} + 1)$
(3) $\mathsf{read}(\mathsf{a}, i) \leq \mathsf{read}(\mathsf{a}, j) \vee \neg(i < j) \vee i \notin [1..1000^i] \vee j \notin [1..1000^j]$

where $t \in [l..h]$ abbreviates the formula $l \leq t \wedge t \leq h$ for any integer-sorted terms t, l and h. Variables are typeset in italics, e.g, x, and operators in sans-serif, e.g., read, a and m. The axioms (1) and (2) are the standard axioms for integer-sorted arrays with integer indices. The axiom (3) states that the array a is sorted within the domain [1..1000] for i and j. Annotating the upper bounds as 1000^i and 1000^j facilitates replacing them with different values for a given variable, see below. The clauses (4) constrains the integer constant m to the stated range. The task is to confirm that N is satisfiable.

In order to check satisfiability with hierarchic superposition, the input clause set has to be *sufficiently complete* (cf. Section 2). In the example, sufficient completeness means that in every model of (1)-(5) wrt. pure first-order logic every ground read-term must be equal to some background term. With the axioms (1) and (2) every write-term inside of a read-term can be eliminated, and so the only critical terms are applications of read to the array constant a. The clauses (3) and (5) constrain the interpretation of terms of the form $\mathsf{read}(\mathsf{a}, t)$ but do not enforce sufficient completeness. Achieving sufficient completeness for *ground* clauses like (5) is easy, one just needs to add "definitions" like (5b) $\mathsf{read}(\mathsf{a}, \mathsf{m}) \approx \mathsf{n}_0$ and (5c) $\mathsf{read}(\mathsf{a}, \mathsf{m} + 1) \approx \mathsf{n}_1$ where n_0 and n_1 are fresh integer-sorted parameters (symbolic constants) and replace the clause (5) by (5a) $\mathsf{n}_0 < \mathsf{n}_1$. Indeed, our transformation does all that (and so does our earlier calculus in [7]).

The more difficult part concerns the non-ground clause (3). Our procedure generalizes the above mechanism of introducing definitions and applying them to the non-ground case (see Section 3). For that, it uses a candidate model which initially is the *default interpretation* that maps all read-terms of a particular shape to the *same* arbitrary symbolic constant. This results in the following transformation of clause (3):

$$(3a)\ \ \mathsf{n}_3 \leq \mathsf{n}_4 \vee \neg(i < j) \vee i \notin [1..1000^i] \vee j \notin [1..1000^j]$$
$$(3b)\ \ \mathsf{read}(\mathsf{a}, i) \approx \mathsf{n}_3 \vee i \notin [1..1000^i] \quad (3c)\ \ \mathsf{read}(\mathsf{a}, j) \approx \mathsf{n}_4 \vee j \notin [1..1000^j]$$

Clauses (3b) and (3c) are the definitions for the default interpretation, one per occurrence of a read-term in (3), and clause (3a) is clause (3) after applying these definitions.

The new clause set $N_1 = \{(1), (2), (3a)–(3c), (4), (5a)–(5c)\}$ now needs to be checked for satisfiability. Because the clause set N_1 is sufficiently complete and hierarchic superposition decides the underlying fragment, we get a definite result.

The clause set N_1 is in fact unsatisfiable. Because this only means that N is not satisfied using the current model candidate, the search for a model needs to continue. This is done by refining the default interpretation at a critical point that is responsible for unsatisfiability. Our algorithm determines that point as an adjacent one to a maximal sub-domain that results in satisfiability. In the example, this is the sub-domain $[1..999^i]$ for the variable i and the point is 1000. That is, the set N_2 obtained from N_1 by replacing everywhere 999^i by 1000^i is satisfiable, while adding back 1000 to $[1..999^i]$ makes it unsatisfiable again. The refinement then is done by excluding the point 1000 from the default interpretation and providing a separate definition for it. The corresponding transformation of clause (3) hence looks as follows :

$$(3a1)\ \ \mathsf{n}_{31} \leq \mathsf{n}_4 \vee \neg(i < j) \vee i \notin [1..1000^i] \setminus \{1000\} \vee j \notin [1..1000^j]$$
$$(3a2)\ \ \mathsf{n}_{32} \leq \mathsf{n}_4 \vee \neg(1000 < j) \vee j \notin [1..1000^j]$$
$$(3b1)\ \ \mathsf{read}(\mathsf{a}, i) \approx \mathsf{n}_{31} \vee i \notin [1..1000^i] \setminus \{1000\} \quad (3c)\ \ \mathsf{read}(\mathsf{a}, j) \approx \mathsf{n}_4 \vee j \notin [1..1000^j]$$
$$(3b2)\ \ \mathsf{read}(\mathsf{a}, 1000) \approx \mathsf{n}_{32}$$

Clauses (3b1) and (3b2) provide the modified definitions and clauses (3a1) and (3a2) are the correspondingly rewritten versions of (3). Let $N_3 = \{(1), (2), (3a1)–(3c), (4), (5a)–(5c)\}$ be the result of the current transformation step.

The clause set N_3 is still unsatisfiable. In the next round, the new upper bounds required for the clauses in N_3 to have satisfiability are 999^j and 1000^i. Transforming clause (3) wrt. the points 1000 for j and 1000 for i from the previous step gives:

$$(3a1)\ \ \mathsf{n}_{31} \leq \mathsf{n}_{41} \vee \neg(i < j) \vee i \notin [1..1000^i] \setminus \{1000\} \vee j \notin [1..1000^j] \setminus \{1000\}$$
$$(3a2)\ \ \mathsf{n}_{32} \leq \mathsf{n}_{41} \vee \neg(1000 < j) \vee j \notin [1..1000^j] \setminus \{1000\}$$
$$(3a3)\ \ \mathsf{n}_{31} \leq \mathsf{n}_{42} \vee \neg(i < 1000) \vee i \notin [1..1000^j] \setminus \{1000\}$$
$$(3a4)\ \ \mathsf{n}_{32} \leq \mathsf{n}_{42} \vee \neg(1000 < 1000)$$
$$(3b1)\ \ \mathsf{read}(\mathsf{a}, i) \approx \mathsf{n}_{31} \vee i \notin [1..1000^i] \setminus \{1000\} \quad (3b2)\ \ \mathsf{read}(\mathsf{a}, 1000) \approx \mathsf{n}_{32}$$
$$(3c1)\ \ \mathsf{read}(\mathsf{a}, j) \approx \mathsf{n}_{41} \vee j \notin [1..1000^j] \setminus \{1000\} \quad (3c2)\ \ \mathsf{read}(\mathsf{a}, 1000) \approx \mathsf{n}_{42}$$

Let $N_4 = \{(1), (2), (3a1) - (3c2), (4), (5a) - (5c)\}$ be the result of the current transformation step. This time, N_4 is satisfiable, and so is N, with the same models. If I is any such model we have $I(\mathsf{m}) = 999$, $I(\mathsf{read}(\mathsf{a}, i)) = k$, for some integer k and all $i = 1..999$, and $I(\mathsf{read}(\mathsf{a}, 1000)) = l$ for some integer $l > k$. (We present the general model finding procedure and its correctness results in Section 4.)

The example is solved after two iterations of transformation steps. In general, each transformation step needs $O(m \cdot \log(n))$ prover calls to determine the sub-intervals and the next point as explained above, where m is the number of variables in the given clause set after making clauses variable-disjoint and n is the size of the largest domain. In total, with $m = 2$ and $n = 1000$ this accounts for $2 \cdot (m \cdot \log(n)) \leq 40$ theorem prover calls, however each one rather simple. By contrast, the full ground instantiation of the clauses (3)-(5) has a size of $n^m = 10^6$ which, in general, grows too quickly for current theorem provers or SMT solvers. In the worst case, though, our method also requires full ground instantiation (but is not worse). This happens when the default interpretation is unsuitable for the whole domain, so that separate definitions are needed for all points to establish (un)satisfiability. In Section 5 we report on first experimental results.

Related Work

Related work comes from several directions. Procedures for computing models of first-order logic formulas *without background theories* have a long tradition in automated reasoning. MACE-style model finding [8] utilizes translation into propositional SAT or into EPR [4] for deciding satisfiability wrt. a given candidate domain size k; SEM-style model finding [22, 23, 15] utilizes constraint solving techniques, again wrt. k. The main problem is scalability wrt. both the domain size k and the number of variables in the input clause set, which severely limits the applicability of both styles in practice. Recently, Reynolds, Tinelli, Goel, Krstić, Deters and Barrett proposed a finite model finding procedure in the SMT framework that addresses this problem by on-demand instantiation techniques [20, 19]. This way, their work is conceptually related to ours, but, unlike ours, they allow quantification only over variables ranging into the free sort. An extension for quantifying variables over background domains such as the integers does not seem straightforward and is left as future work in [20].

Heuristic instantiation is the state of the art technique for handling quantified formulas in SMT-solvers [10, 16]. These heuristics perform impressively well in practice, but in general are incomplete even for pure first-order logic. Ge and deMoura [11] propose a technique where the ground terms used for instantiation come from solving certain set constraints. They obtain completeness results for the fragment where every variable occurs only as an argument of a free function or predicate symbol. Interestingly, they also use the notion of a default interpretation in a similar way as we do. However, even with certain extensions their approach remains incomparable to ours. For example, terms like $\mathsf{f}(x + y)$ are disallowed, but are acceptable in our approach when x and y are finitely quantified.

Regarding related work in first-order theorem proving, the problem we are considering has been tackled in the framework of the hierarchic superposition calculus [3]. Weidenbach and Kruglow [14] consider the case when all background-sorted terms are ground, similarly to our calculus in [7]. In [6] we have identified a certain syntactic fragment that enables complete reasoning.

2 Hierarchic Theorem Proving

Hierarchic superposition [3, 7] is a calculus for automated reasoning in a hierarchic combination of first-order logic and some background theory, for instance some form of arithmetic. We consider the following scenario:[1]

We assume that we have a *background ("BG")* prover that accepts as input a set of clauses over a *BG signature* $\Sigma_B = (\Xi_B, \Omega_B)$, where Ξ_B is a set of *BG sorts* and Ω_B is a set of *BG operators*. Terms/clauses over Σ_B and BG-sorted variables are called *BG terms/clauses*. The BG prover decides the satisfiability of Σ_B-clause sets w. r. t. a *BG specification*, that is, a class of term-generated Σ_B-interpretations (called *BG models*) that is closed under isomorphisms. We assume that Ω_B contains a set of distinguished constant symbols $\Omega_B^D \subseteq \Omega_B$ that has the property that any two distinct $d_1, d_2 \in \Omega_B^D$ are interpreted by different elements in every BG model. We refer to these constant symbols as *(BG) domain elements*. We also assume that Σ_B contains infinitely many *parameters*, that is, additional constants that may be interpreted freely by arbitrary elements of the appropriate domain. In examples we use $\{0, 1, 2, \dots\}$ to denote BG domain elements, $\{+, -, <, \leq\}$ to denote (non-parameter) BG operators, and the possibly subscripted letters $\{x, y\}$ and $\{\alpha, \beta\}$ to denote variables and parameters, respectively. We assume that the BG specification is the class of all models of linear integer arithmetic (LIA).

The *foreground ("FG")* theorem prover accepts as input clauses over a signature $\Sigma = (\Xi, \Omega)$, where $\Xi_B \subseteq \Xi$ and $\Omega_B \subseteq \Omega$. The sorts in $\Xi_F = \Xi \setminus \Xi_B$ and the operator symbols in $\Omega_F = \Omega \setminus \Omega_B$ are called *FG sorts* and *FG operators*. The intended semantics is that of *conservative extensions of the BG specification*, i. e., Σ-interpretations whose restriction to Σ_B is a model of the BG specification. Below we refer to satisfiability in this sense as *$\mathcal{B}$-satisfiability*.

We use $\{\mathsf{a}, \mathsf{b}, \mathsf{c}, \mathsf{f}, \mathsf{g}\}$ to denote FG operators. A Σ-term is an *FG term* if it is not a BG term, that is, if it contains at least one FG operator or FG variable (and analogously for equations, literals, or clauses). We emphasize that for an FG operator $\mathsf{f} : \xi_1 \dots \xi_n \to \xi_0$ in Ω_F any of the ξ_i may be a BG sort. Consequently, FG terms may have BG sorts. Every FG operator f with a BG range sort $\xi_0 \in \Xi_B$ is called a *free BG-sorted (FG) operator*.

After abstracting out BG terms other than BG domain elements and variables that occur as subterms of FG terms,[2] the FG prover saturates the set of Σ-clauses using the inference rules of hierarchic superposition, such as, e. g.,

$$\text{Negative superposition} \quad \frac{l \approx r \lor C \qquad s[u] \not\approx t \lor D}{\mathrm{abstr}((s[r] \not\approx t \lor C \lor D)\sigma)}$$

if (i) neither l nor u is a BG term, (ii) u is not a variable, (iii) σ is an mgu of l and u, (iv) σ maps all BG variables to BG terms, (v) $r\sigma \not\succeq l\sigma$, (vi) $(l \approx r)\sigma$ is strictly maximal in $(l \approx r \lor C)\sigma$, (vii) the first premise does not have selected literals, (viii) $t\sigma \not\succeq s\sigma$, and (ix) if the second premise has selected literals, then $s \not\approx t$ is selected in the second premise, otherwise $(s \not\approx t)\sigma$ is maximal in $(s \not\approx t \lor D)\sigma$.

---

[1] Due to a lack of space, we can only give a brief overview of the calculus and of the semantics of hierarchic specifications. We refer to [7] for the details.

[2] *Abstracting out* a term t that occurs in a clause $C[t]$ means replacing $C[t]$ by $x \not\approx t \lor C[x]$ for a new variable x.

These differ from the standard superposition inference rules [2] mainly in that only the FG parts of clauses are overlapped and that any BG clauses derived during the saturation are instead passed to the BG prover. The BG prover implements an inference rule

$$\text{Close}\ \frac{C_1 \quad \cdots \quad C_n}{\square}$$ if $C_1, \ldots, C_n$ are BG clauses and $\{C_1, \ldots, C_n\}$ is unsatisfiable w. r. t. the BG specification.

As soon as one of the two provers detects a contradiction, the input clause set has been shown to be $\mathcal{B}$-unsatisfiable.

There are two requirements for the refutational completeness of hierarchic superposition. The first one is *sufficient completeness*: We must be able to prove that every ground BG-sorted FG term is equal to some BG term. Sufficient completeness of a set of Σ-clauses is a property that is not even recursively enumerable. For certain classes of Σ-clause sets, however, it is possible to establish a variant of sufficient completeness automatically [14, 7]: If all BG-sorted FG terms in the input are ground, it suffices to show that each BG-sorted FG term *in the input* is equal to some BG term. This can be achieved by adding a *definition* $\alpha_t \approx t$ for every BG-sorted FG term t occurring in a clause $C[t]$, where α_t is a new parameter (BG constant); afterwards $C[t]$ can be replaced by $C[\alpha_t]$.

Since we can only pass *finite* clause sets to a BG prover, there is a second requirement for refutational completeness, namely the compactness of the BG specification. A specification is called *compact*, if every set of formulas that is unsatisfiable w. r. t. the specification has a finite unsatisfiable subset.

3 Finite Domain Transformation

We are interested in refutationally complete hierarchic theorem proving in the presence of free BG-sorted FG operators. Unfortunately, just adding one free predicate symbol to linear integer arithmetic results in a Π^1_1-hard validity problem. To circumvent this problem, we work with a modified semantics and introduce a concept of finite quantification of BG variables. This allows us to remove all free BG-sorted FG operators by a *finite domain transformation*, introduced next, and use existing reasoning methods as decision procedures on the result.

Let $\xi \in \Xi_{\mathrm{B}}$ be a BG sort. By a *finite ξ-domain Δ* we mean any possibly empty finite set $\{d_1, \ldots, d_n\} \subseteq \Omega^D_{\mathrm{B}}$ of ξ-sorted domain elements d_i. Set membership in Δ can be expressed by a BG formula $\mathcal{F}_\Delta[x]$ in one free ξ-sorted variable x whose extension is exactly the set Δ, in every $\mathcal{B}$-interpretation. One can always take $\mathcal{F}_\Delta[x] = x \approx d_1 \vee \cdots \vee x \approx d_n$, but if supported by the BG logic, as in the case of integer arithmetic, it may be advantageous to use "compact" representations like $\mathcal{F}_\Delta[x] = 1 \leq x \wedge x \leq 20$ instead.

We use set-theoretic expressions for finite ξ-domains, in particular of the form $\Delta \setminus \Gamma$, where Γ is a finite set of domain elements of the proper sort. In the previous example, e.g., $\mathcal{F}_{\Delta\setminus\{3,5\}}[x] = 1 \leq x \wedge x \leq 20 \wedge x \not\approx 3 \wedge x \not\approx 5$. Instead of $\mathcal{F}_\Delta[x]$ and $\mathcal{F}_{\Delta\setminus\Gamma}[x]$ we generally write $x \in \Delta$ and $x \in \Delta \setminus \Gamma$, respectively, and $x \notin \Delta$ and $x \notin \Delta \setminus \Gamma$ for their negations. We call these expressions *domain predicates* and treat them as literals in clauses instead of expanding them.

Definition 3.1. *A* finitely quantified clause *is a Σ-clause of the form $D \vee x_1 \notin \Delta_{x_1} \vee \cdots \vee x_n \notin \Delta_{x_n}$ such that D does not contain domain predicates, $n \geq 0$, $x_i \neq x_j$ for $1 \leq i < j \leq n$, and every variable occurring below a free BG-sorted operator in D is among $x_1, \ldots, x_n$.*

For example, $\mathsf{f}(x + 1) > \alpha + y \vee y > 0 \vee x \notin [1..1000]$ is finitely quantified.

Example 3.2. Let N consist of the following two finitely quantified clauses:

(C_1) $\mathsf{f}(x_1) > x_1 \vee x_1 \notin [1..1000]$
(C_2) $\mathsf{f}(x_2 + 3) < 10 \vee \neg(x_2 > 2) \vee x_2 \notin [1..1000]$

We formally have $\Delta_{x_1} = \Delta_{x_2} = [1..1000]$, and in C_1 the pseudo-literal $x_1 \notin [1..1000]$ is short for $\neg(1 \leq x_1 \leq 1000)$. □

Where $\mathbf{x} = (x_1, \ldots, x_n)$, let $\Delta_{\mathbf{x}}$ denote the $\mathbf{x}$-indexed list $(\Delta_{x_1}, \ldots, \Delta_{x_n})$ of sets of domain elements. We extend usual set operations pointwise to $\mathbf{x}$-indexed lists $\Pi_{\mathbf{x}}$ and $\Delta_{\mathbf{x}}$ of sets of domain elements. For instance $\Pi_{\mathbf{x}} \subseteq \Delta_{\mathbf{x}}$ iff $\Pi_x \subseteq \Delta_x$, for each $x \in \mathbf{x}$.

We are going to define the earlier mentioned finite domain transformation for evaluating finitely quantified clauses under a given interpretation. It takes as input a finitely quantified clause $C[\Delta_{\mathbf{x}}]$ and sets of points $\Pi_{\mathbf{x}}$ that provide possible exceptions to interpreting the free BG-sorted operators as the constant function on the domains $\Delta_{\mathbf{x}}$ as specified by the default interpretation.

Definition 3.3 (Finite Domain Transformation). *Let $C[\Delta_{\mathbf{x}}] = D \vee x_1 \notin \Delta_{x_1} \vee \cdots \vee x_n \notin \Delta_{x_n}$ be a finitely quantified clause and $\Pi_{\mathbf{x}} \subseteq \Delta_{\mathbf{x}}$ a list of sets of domain elements.*

Let $\mathrm{Cls}_C := \emptyset$ and $\mathrm{Def}_C := \emptyset$ be initially empty sets of Σ-clauses. For every partition $\{y_1, \ldots, y_k\} \uplus \{z_1, \ldots, z_l\}$ of $\{x_1, \ldots, x_n\}$ do the following:

For all substitutions $\gamma = [z_1 \mapsto d_1, \ldots z_l \mapsto d_l]$ such that $d_m \in \Pi_{z_m}$:

1. *Let $E := D\gamma$*
2. *While E has the form $E[t]$ where t is a minimal term with a free BG-sorted operator at the top-level do the following:*
   (a) *Let α be a fresh parameter*
   (b) *Add to Def_C the clause $t \approx \alpha \vee y_1 \notin \Delta_{y_1} \setminus \Pi_{y_1} \vee \cdots \vee y_k \notin \Delta_{y_k} \setminus \Pi_{y_k}$*
   (c) *Set $E := E[\alpha]$*
3. *Add to Cls_C the clause $E \vee y_1 \notin \Delta_{y_1} \setminus \Pi_{y_1} \vee \cdots \vee y_k \notin \Delta_{y_k} \setminus \Pi_{y_k}$*

The result is the pair $FD(C, \Pi_{\mathbf{x}}) = (\mathrm{Cls}_C, \mathrm{Def}_C)$, the finite domain transformation *of C.*

By the minimality of t in (2) we mean that no proper subterm of t is built with a free BG-sorted operator. The finite domain transformation removes from the given finitely quantified clause C every occurrence of a term t built with some free BG-sorted symbol. Recall from Definition 3.1 that all variables in t are among $\mathbf{x} = (x_1, \ldots, x_n)$. The removal of t distinguishes whether x_i is interpreted as an element of $\Delta_i \setminus \Pi_i$ or as an element $d_i \in \Pi_i$. This is done in all possible ways by exhaustive partitioning of the variables $\boldsymbol{x}$ and exhausting the substitution γ for all possible assignments for x_i. The set $\Delta_i \setminus \Pi_i$ specifies those domain elements for which the interpretation of t is undistinguished, and the set Π_i specifies those domain elements for which the interpretation of t is distinguished, by taking different parameters α per substitution γ. In step (b) corresponding definitions for t are put into Def_C. Step (c) applies these definitions to the current clause E.

On complexity: the result $\mathrm{FD}(C, \Pi_{\mathbf{x}})$ contains $O(|\boldsymbol{x}|^{|\Pi_{\mathbf{x}}|+1})$ clauses. This is, because for every $x_i \in \boldsymbol{x}$ a choice is made for either instantiating x_i exhaustively with all elements from $\Pi_{\mathbf{x}}$ if $x_i \in \{z_1, \ldots, z_n\}$, or otherwise not doing so, which explains the " + 1". (Extracting out subterms does not affect the complexity.) In the worst case $\Pi_{\mathbf{x}} = \Delta_{\mathbf{x}}$ and all clauses stemming from the latter case are tautological. The complexity in this case is $O(|\boldsymbol{x}|^{|\Pi_{\mathbf{x}}|})$, which is the same as with ground-instantiation based MACE-style model finders.

Example 3.2 (continued). Let $\Pi_{(x_1)} = (\{9\})$. Then $\mathrm{FD}(C_1, \Pi_{(x_1)})$ consists of the clauses

$$
\begin{array}{ll}
(C_{11})\ \ \alpha_1 > x_1 \vee x_1 \notin [1..1000] \setminus \{9\} & (C_{13})\ \ \alpha_2 > 9 \\
(C_{12})\ \ \mathsf{f}(x_1) \approx \alpha_1 \vee x_1 \notin [1..1000] \setminus \{9\} & (C_{14})\ \ \mathsf{f}(9) \approx \alpha_2
\end{array}
$$

where $\mathrm{Cls}_{C_1} = \{C_{11}, C_{13}\}$ and $\mathrm{Def}_{C_1} = \{C_{12}, C_{14}\}$. The left clauses stem from partitioning $\{x_1\}$ as $\{y_1\} \uplus \emptyset$, and the right clauses from $\emptyset \uplus \{z_1\}$. There are two occurrences of $\Delta_{x_1} = [1..1000]$. □

There are no restrictions on nesting free BG-sorted operators, although none of our examples shows that. For example, a literal like $\mathsf{f}(x + \mathsf{g}(y, \beta)) > f(y) + y$ is perfectly acceptable. The possible nesting of free BG-sorted operators necessitates the while-loop in step (2) in Definition 3.3; removing all of them in a single step is not possible.

The sets of domain elements Δ_x occurring in clauses in $\mathrm{FD}(C, \Pi_{\mathbf{x}})$ are all within pseudo-literals of the form $x \notin \Delta_x \setminus \Pi_x$. Hence, both Cls_C and Def_C are of the form $\mathrm{Cls}_C[\Delta_{\mathbf{x}}]$ and $\mathrm{Def}_C[\Delta_{\mathbf{x}}]$. Moreover, in $\mathrm{FD}(C, \Pi_{\mathbf{x}})$, every free BG-sorted operator f occurs only in a clause of the form $\mathsf{f}(t_1, \ldots, t_n) \approx \alpha \vee D$ in Def_C where no t_i and no literal in D contains any free BG-sorted operator.

The finite domain transformation is generalized to clause sets by taking the union of the finite domain transformations applied to its members. More precisely, let $N = \{C_1[\Delta_{\mathbf{x}_1}], \ldots, C_m[\Delta_{\mathbf{x}_m}]\}$ be a finite set of finitely quantified clauses. Let us assume that the clauses in N have been renamed apart, so that the lists of variables $\mathbf{x}_i$ are pairwise disjoint, for all $i = 1..m$. By definition, each $\mathbf{x}_i$ consists of pairwise different variables, too. This allows us to take $\mathbf{x}$ as the concatenation of all $\mathbf{x}_i$'s and to write $\Delta_{\mathbf{x}}$ for the concatenation of all $\Delta_{\mathbf{x}_m}$'s. The clause set N hence is of the form $N[\Delta_{\mathbf{x}}]$. Now let $(\mathrm{Cls}_{C_i}, \mathrm{Def}_{C_i}) = \mathrm{FD}(C_i, \Pi_{\mathbf{x}})$ and define $\mathrm{FD}(N, \Pi_{\mathbf{x}}) = (\mathrm{Cls}_N, \mathrm{Def}_N)$ where $\mathrm{Cls}_N = \bigcup_{i=1..m} \mathrm{Cls}_{C_i}$ and $\mathrm{Def}_N = \bigcup_{i=1..m} \mathrm{Def}_{C_i}$.

Below, we usually denote $\mathrm{FD}(N, \Pi_{\mathbf{x}})$ as a single clause set $M[\Delta_{\mathbf{x}}] = \mathrm{Cls}_N \cup \mathrm{Def}_N$. The following result follows immediately:

Proposition 3.5. *Let $N[\Delta_{\mathbf{x}}]$ be a set of finitely quantified clauses and $\Pi_{\mathbf{x}} \subseteq \Delta_{\mathbf{x}}$. Then $FD(N, \Pi_{\mathbf{x}})$ is sufficiently complete.*

Proposition 3.5 is one of the ingredients that allows us to argue for hierarchic superposition [7] as a decision procedure for $\mathcal{B}$-satisfiability of the clause sets $\mathrm{FD}(C, \Pi_{\mathbf{x}})$. We also need a termination argument for derivations (compactness, cf. Section 2, is unproblematic then). This is easy, for instance, in the absence of non-ground FG-sorted operators only finitely many superposition steps exist and all of these are between the clauses in

Def_C, and then only at the top-level – that is, between the literals $\mathsf{f}(t_1, \ldots, t_n) \approx \alpha$. Alternatively one can use SMT-solvers after removing all free BG-operators by exhaustive application of a superposition-like inference rule that from premises $\mathsf{f}(t_1, \ldots, t_n) \approx \alpha \vee D$ and $\mathsf{f}(s_1, \ldots, s_n) \approx \beta \vee E$ derives the clause $s_1 \not\approx t_1 \vee \cdots \vee s_n \not\approx t_n \vee \alpha \approx \beta \vee D \vee E$. In general, hierarchic superposition can be used if it is guaranteed to terminate on $\mathrm{FD}(C, \Pi_{\mathbf{x}})$. This applies, e.g., to the example in the introduction.

The notation $M[\Delta_{\mathbf{x}}]$ makes it easy to modify the sets Δ_x in pseudo-literals in clauses in M. More precisely, if $\Delta_{\mathbf{x}} = (\ldots, \Delta_x, \ldots)$ for some $x \in \mathbf{x}$ and Γ is a set of domain elements with the same sort as x, we denote by $\Delta_{\mathbf{x}}[x \mapsto \Gamma]$ the update of $\Delta_{\mathbf{x}}$ at index x by Γ, i.e., the list $(\ldots, \Gamma_x, \ldots)$. Correspondingly, $C[\Delta_{\mathbf{x}}[x \mapsto \Gamma]]$ is the clause that is obtained from $C[\Delta_{\mathbf{x}}]$ be replacing Δ_x by Γ_x everywhere. For clause sets $N[\Delta_{\mathbf{x}}]$ we define $N[\Delta_{\mathbf{x}}[x \mapsto \Gamma]]$ analogously.

Example 3.2 (continued). The clause set N is of the form $N[\Delta_{\mathbf{x}}]$ where $\mathbf{x} = (x_1, x_2)$ and $\Delta_{x_1} = \Delta_{x_2} = [1..1000]$. Now let $\Pi_{\mathbf{x}} = (\{9\}, \{6\})$. Then $M[\Pi_{\mathbf{x}}] = \mathrm{FD}(N, \Pi_{\mathbf{x}}) = (\mathrm{Cls}_{C_1} \cup \mathrm{Cls}_{C_2}) \cup (\mathrm{Def}_{C_1} \cup \mathrm{Def}_{C_2})$ where $\mathrm{Cls}_{C_2} = \{C_{21}, C_{23}\}$, $\mathrm{Def}_{C_2} = \{C_{22}, C_{24}\}$ and

(C_{21}) $\alpha_3 < 10 \vee \neg(x_2 > 2) \vee x_2 \notin [1..1000] \setminus \{6\}$ $\quad$ (C_{23}) $\alpha_4 < 10 \vee \neg(6 > 2)$

(C_{22}) $\mathsf{f}(x_2 + 3) \approx \alpha_3 \vee x_2 \notin [1..1000] \setminus \{6\}$ $\quad$ (C_{24}) $\mathsf{f}(6 + 3) \approx \alpha_4$

The clause set $M[\Delta_{\mathbf{x}}[x_2 \mapsto \emptyset]] = M[(\{9\}, \emptyset)]$ is obtained by replacing the two occurrences of $\Delta_{x_2} = [1..1000]$ in C_{21} and C_{22} by the empty interval []. ◻

We conclude this section with some lemmas that will be needed in the proof of the main correctness result, Theorem 4.2 below. In each of them, $N[\Delta_{\mathbf{x}}]$ is a set of finitely quantified clauses, $\Pi_{\mathbf{x}} \subseteq \Delta_{\mathbf{x}}$, $(\mathrm{Cls}_N, \mathrm{Def}_N) = \mathrm{FD}(N, \Pi_{\mathbf{x}})$, and $M = \mathrm{Cls}_N \cup \mathrm{Def}_N$.

Lemma 3.7. $\mathrm{Cls}_N \cup \mathrm{Def}_N$ *is $\mathcal{B}$-satisfiable iff $N \cup \mathrm{Def}_N$ is $\mathcal{B}$-satisfiable.*

Proof. For the if-direction assume that $N \cup \mathrm{Def}_N$ is $\mathcal{B}$-satisfiable. It suffices to show that $N \cup \mathrm{Cls}_N \cup \mathrm{Def}_N$ is $\mathcal{B}$-satisfiable. Observe that all clauses in Cls_N can be seen to be obtained by paramodulation inferences from clauses in $N \cup \mathrm{Def}_N$, which are all logical consequences of $N \cup \mathrm{Def}_N$.

For the only-if direction assume that $\mathrm{Cls}_N \cup \mathrm{Def}_N$ is $\mathcal{B}$-satisfiable. The definitions in Def_N are exhaustive in the sense that any instance C of a finitely quantified clause in N obtained by ground instantiation with domain elements is congruent with some clause in Cls_N obtained by paramodulation with clauses in Def_N. This entails that $N \cup \mathrm{Cls}_N \cup \mathrm{Def}_N$ is $\mathcal{B}$-satisfiable, and hence so is $N \cup \mathrm{Def}_N$. ◻

Lemma 3.8. *If $M[\emptyset_{\mathbf{x}}]$ is $\mathcal{B}$-unsatisfiable then N and N' are $\mathcal{B}$-unsatisfiable, where N' is obtained from N by removing from all clauses all domain predicates.*

Proof. Assume that $M[\emptyset_{\mathbf{x}}]$ is $\mathcal{B}$-unsatisfiable. Every clause in $M[\Delta_{\mathbf{x}}]$ that contains a pseudo-literal of the form $x \notin \Delta_x \setminus \Pi_x$, for some $x \in \mathbf{x}$, becomes a tautology in $M[\emptyset_{\mathbf{x}}]$ after replacing $x \notin \Delta_x \setminus \Pi_x$ by $x \notin \emptyset \setminus \Pi_x$. Deleting all these tautologies leaves us with a ($\mathcal{B}$-unsatisfiable) set $M' \subseteq M[\emptyset_{\mathbf{x}}]$. All clauses in M' are either ground definitions in Def_N of the form $t \approx \alpha$ (cf. Definition 3.3), or clauses in Cls_N that are obtained by (repeated) paramodulation of the sub-clause D of a clause $C \in N$ (cf. again Definition 3.3) such that all instantiated domain predicates in the instance $C\gamma$ are satisfied. Clearly, adding such definitions to N preserves $\mathcal{B}$-satisfiability. The $\mathcal{B}$-unsatisfiability of both N and N' then follows from the soundness of paramodulation. ◻

Lemma 3.9. *Let $\Gamma_{\mathbf{x}}$ be a vector of sets of domain elements of the proper sorts. For every $x \in \mathbf{x}$ and $d \in \Pi_x$, if $M[\Gamma_{\mathbf{x}}]$ is $\mathcal{B}$-satisfiable then $M[\Gamma_{\mathbf{x}}[x \mapsto \Gamma_x \cup \{d\}]]$ is $\mathcal{B}$-satisfiable.*

Proof. All occurrences of Γ_x in clauses in $M[\Gamma_{\mathbf{x}}]$ are within pseudo-literals of the form $x \notin \Gamma_x \setminus \Pi_x$. We are given $d \in \Pi_x$. It follows trivially that $\Gamma_x \setminus \Pi_x$ and $(\Gamma_x \cup \{d\}) \setminus \Pi_x$ are the same sets, which immediately entails the claim. □

Example 3.2 (continued). Let $M[\Delta_{(x_1)}] = \mathrm{FD}(C_1, \Pi_{(x_1)})$ from above. Let $\Gamma_{(x_1)} = ([5..500])$ and $d = 9$. Then $M[\Gamma_{(x_1)}[x_1 \mapsto \Gamma_{x_1} \cup \{d\}]]$ consists of the clauses

$$\begin{array}{llll} (C'_{11}) & \alpha_1 > x_1 \lor x_1 \notin ([5..500] \cup \{9\}) \setminus \{9\} & (C_{13}) & \alpha_2 > 9 \\ (C'_{12}) & \mathsf{f}(x_1) \approx \alpha_1 \lor x_1 \notin ([5..500] \cup \{9\}) \setminus \{9\} & (C_{14}) & \mathsf{f}(9) \approx \alpha_2 \end{array}$$

Lemma 3.9 requires $d \in \Pi_x$. Adding d to Γ_x does not change anything, as d is again removed from $\Gamma_x \cup \{d\}$: the sets $([5..500] \cup \{9\}) \setminus \{9\}$ and $[5..500] \setminus \{9\}$ are the same. □

4 Checking Satisfiability

Next we define a procedure checkSAT for checking the $\mathcal{B}$-satisfiability of sets of finitely quantified clauses. It repeatedly applies the finite domain transformation wrt. growing sets of exception points. It stops if a transformed set has been found that is either $\mathcal{B}$-satisfiable or serves to demonstrate $\mathcal{B}$-unsatisfiability.

algorithm checkSAT($N[\Delta_{\mathbf{x}}]$)
// returns "$\mathcal{B}$-satisfiable" or "$\mathcal{B}$-unsatisfiable"
var $\Pi_{\mathbf{x}} := \emptyset_{\mathbf{x}}$ // The current set of exceptions
while true {
  let $M = \mathrm{FD}(N, \Pi_{\mathbf{x}})$
  if M is $\mathcal{B}$-satisfiable **return** "$\mathcal{B}$-satisfiable" // justified by Lemma 3.7
  if $M[\emptyset_{\mathbf{x}}]$ is $\mathcal{B}$-unsatisfiable **return** "$\mathcal{B}$-unsatisfiable" // justified by Lemma 3.8
  let $(x, d) = \mathrm{find}(M)$
  $\Pi_{\mathbf{x}} := \Pi_{\mathbf{x}}[x \mapsto \Pi_x \cup \{d\}]$
}

algorithm find($M[\Delta_{\mathbf{x}}]$)
// returns a pair (x, d) such that $x \in \mathbf{x}$ and $d \in \Delta_x \setminus \Pi_x$
let $(x_1, \ldots, x_n) = \mathbf{x}$
for $i = 1$ **to** n {
  if $M[\emptyset_{(x_1,\ldots,x_i)} \cdot \Delta_{(x_{i+1},\ldots,x_n)}]$ is $\mathcal{B}$-satisfiable {
    let $\Gamma \subseteq \Delta_{x_i}$ and $d \in \Gamma$ such that
      $M[\emptyset_{(x_1,\ldots,x_{i-1})} \cdot \Gamma_{x_i} \cdot \Delta_{(x_{i+1},\ldots,x_n)}]$ is $\mathcal{B}$-unsatisfiable and
      $M[\emptyset_{(x_1,\ldots,x_{i-1})} \cdot (\Gamma \setminus \{d\})_{x_i} \cdot \Delta_{(x_{i+1},\ldots,x_n)}]$ is $\mathcal{B}$-satisfiable // see text
    return (x_i, d) // from Lemma 3.9 it follows $d \in \Delta_x \setminus \Pi_x$ as claimed
    }
}

We tacitly assume that the $\mathcal{B}$-satisfiability tests in checkSAT and find are effective. This is always the case, e.g., if there are no FG operators other than free BG-sorted operators and the EA-fragment of the background theory is decidable.

Let us go through the run of checkSAT(N), where $N = \{C_1, C_2\}$ from Example 3.2. Let $\Pi^1_{\mathbf{x}} = (\emptyset, \emptyset)$ be the initially empty set of exceptions set in line 3. For $M^1 = \mathrm{FD}(N, \Pi^1_{\mathbf{x}})$ in line 5 none of the termination cases applies, hence find is called. The condition in the for-loop in find is satisfied for $i = 1$. In line 6, a suitable set Γ is the interval $[1..9]$ and $d = 9$, as $M^1[([1..9], \Delta_{x_2})]$ is $\mathcal{B}$-unsatisfiable and $M^1[([1..9] \setminus \{9\}, \Delta_{x_2})]$ is $\mathcal{B}$-unsatisfiable. The call of find(M^1) hence returns the pair $(x_1, 9)$. (In the proof of Lemma 4.1 below we show how Γ and $d \in \Gamma$ can be found efficiently by binary search in the case of (linear) integer arithmetic.)

The updated set $\Pi^2_{\mathbf{x}}$ in checkSAT now is $(\{9\}, \emptyset)$ and we get $M^2[\Delta_{\mathbf{x}}] = \mathrm{FD}(N, \Pi^2_{\mathbf{x}})$ in the next iteration. Again, the termination tests do not apply and find(M^2) is called. This time $M^2[(\emptyset, \Delta_{x_2})]$ is $\mathcal{B}$-unsatisfiable and the result of find(M^2) is $(x_2, 6)$.

The updated set $\Pi^3_{\mathbf{x}}$ hence is $(\{9\}, \{6\})$ and $M^3[\Delta_{\mathbf{x}}] = \mathrm{FD}(N, \Pi^3_{\mathbf{x}})$ consists of the clauses C_{11}–C_{14} and C_{21}–C_{24} already shown above. In the next iteration, the set $M^3[\emptyset_{\mathbf{x}}]$ is built, which is obtained by replacing the sets $\Delta_{x_1} = \Delta_{x_2} = [1..1000]$ everywhere by the empty interval []:

(C'_{11}) $\alpha_1 > x_1 \vee x_1 \notin [\,] \setminus \{9\}$	(C_{13}) $\alpha_2 > 9$
(C'_{12}) $\mathsf{f}(x_1) \approx \alpha_1 \vee x_1 \notin [\,] \setminus \{9\}$	(C_{14}) $\mathsf{f}(9) \approx \alpha_2$
(C'_{21}) $\alpha_3 < 10 \vee \neg(x_2 > 2) \vee x_2 \notin [\,] \setminus \{6\}$	(C_{23}) $\alpha_4 < 10 \vee \neg(6 > 2)$
(C'_{22}) $\mathsf{f}(x_2 + 3) \approx \alpha_3 \vee x_2 \notin [\,] \setminus \{6\}$	(C_{24}) $\mathsf{f}(6 + 3) \approx \alpha_4$

By construction, all clauses affected by the replacement are tautological. Yet, the set $M^3[\emptyset_{\mathbf{x}}]$ is $\mathcal{B}$-unsatisfiable, which can be seen easily from the clauses in the right column. The algorithm returns "$\mathcal{B}$-unsatisfiable". This is indeed correct, as, by construction, the remaining non-tautological clauses contain and use definitions for *ground* instances of the f-terms only. Because of that, our method is sound wrt. $\mathcal{B}$-unsatisfiability even for non-finitely quantified clause sets as expressed in Lemma 3.8 above.

Notice that find searches for the set Γ wrt. the whole set $M = \mathrm{FD}(N, \Pi_{\mathbf{x}}) = \mathrm{Cls}_N \cup \mathrm{Def}_N$. It would be tempting to fix Def_N and search only wrt. Cls_N (or vice versa) but this would be unsound. An example for that is the clause set $N = \{f(x) \geq 0 \vee x \notin \Delta,\ f(3) \approx 3,\ f(4) \approx 4\}$, where $\Delta = [0..1000]$. Using the default interpretation we get $\mathrm{Cls}_N = \{\alpha_1 \geq 0 \vee x \notin \Delta,\ \alpha_2 \approx 3,\ \alpha_3 \approx 4\}$ and $\mathrm{Def}_N = \{f(x) \approx \alpha_1 \vee x \notin \Delta,\ f(3) \approx \alpha_2,\ f(4) \approx \alpha_3\}$. While $\mathrm{Cls}_N[\emptyset] \cup \mathrm{Def}_N$ is $\mathcal{B}$-unsatisfiable, N is $\mathcal{B}$-satisfiable. Hence the procedure in that form would be unsound.

Lemma 4.1. *Whenever* find *is called from* checkSAT *on line 8 then the if-clause in the for-loop in* find *is executed for some i, and* find *returns a pair (x_i, d) such that $x_i \in \mathbf{x}$ and $d \in \Delta_{x_i} \setminus \Pi_{x_i}$.*

Proof. Assume find($M[\Delta_{\mathbf{x}}]$) is executed and that $\mathbf{x}$ is of the form $(x_1, \ldots, x_n)$. Because the test in line 7 in checkSAT has not applied it follows that the condition in line 5 in find is satisfied for some i in $1, \ldots, n$. Among all these values, the if-clause is executed for the least one. That is, $M[\emptyset_{(x_1,\ldots,x_i)} \cdot \Delta_{(x_{i+1},\ldots,x_n)}]$ is $\mathcal{B}$-satisfiable and $M[\emptyset_{(x_1,\ldots,x_j)} \cdot \Delta_{(x_{j+1},\ldots,x_n)}]$ is $\mathcal{B}$-unsatisfiable, for all j with $1 \leq j < i \leq n$. Because $i \geq 1$ we can rewrite the former and obtain that $M[\emptyset_{(x_1,\ldots,x_{i-1})} \cdot \emptyset_{x_i} \cdot \Delta_{(x_{i+1},\ldots,x_n)}]$ is $\mathcal{B}$-satisfiable. Furthermore, $M[\emptyset_{(x_1,\ldots,x_{i-1})} \cdot$

$\Delta_{x_i} \cdot \Delta_{(x_{i+1},\ldots,x_n)}]$ is $\mathcal{B}$-unsatisfiable: if $i = 1$ this follows from the fact that the test in line 6 in checkSAT has not applied, and if $i > 1$ this follows from the minimality of i. This shows that Γ and a $d \in \Gamma$ exists as claimed in lines 7 and 8.

In our main application of integer arithmetic the set Γ and $d \in \Gamma$ can be determined efficiently, as follows: We assume the set Δ_{x_i} is an interval of the form $[l..u]$ for some numbers l and u with $l < u$. From the above it follows there is a maximal number u' with $l < u' \leq u$ such that $\Gamma := [l..u']$ is as claimed. The number u' can be determined by binary search in the interval $[l + 1..u]$. By maximality, u' is the desired element d. □

For termination of checkSAT, instead of determining the pair (x, d) in line 11 by the call to find, one could choose any (x, d) such that the current set $\Pi_{\mathbf{x}}$ grows. An advantage of using find, however, is that the relevant ground instances of the clauses $C_1[x_1]$ and $C_2[x_2]$, which are $C_1[9]$ and $C_2[6]$, have been found through semantic guidance by refining the default interpretation in only two steps.

In general terms, checkSAT/find realizes a *heuristic* that tries to search for a model by deviating from the current interpretation only when a conflict arises. The conflict is identified by the point d for the variable x_i in Line 8 of find. The next round of checkSAT continues with the correspondingly updated current interpretation by adding d to Π_{x_i}, which may stop now with "satisfiable", "unsatisfiable" or continue the search.

We summarize the essential properties of checkSAT in our main result as follows.

Theorem 4.2 (Correctness of checkSAT). *For any set N of finitely quantified clauses,* checkSAT(N) *terminates with the correct result "$\mathcal{B}$-satisfiable" or "$\mathcal{B}$-unsatisfiable" for N. Moreover, in case of "$\mathcal{B}$-unsatisfiable" the non-domain restricted version of N is $\mathcal{B}$-unsatisfiable, which is obtained from N by removing from all clauses all domain predicates.*

Proof. Termination follows from the fact that find always returns some pair (x, d) such that $x \in \mathbf{x}$ and $d \in \Delta_x \setminus \Pi_x$, as shown in Lemma 4.1. Hence, the set Π_x grows monotonically in line 12 in checkSAT and there are only finitely many elements in Δ_x available for that. Correctness follows from the lemmas in Section 3 as referenced in the comments in checkSAT. □

5 Experimental Results

We have implemented the checkSAT/find algorithm on top of the hierarchic superposition prover Beagle [7].[3] The implementation is prototypical and currently serves only to try out the ideas in the paper. Table 1 summarizes the experiments we carried out.

We have tried six problems, some of them with varying domain sizes. The problems (1) and (6) are $\mathcal{B}$-unsatisfiable, the others $\mathcal{B}$-satisfiable. The "Problem" column contains the individual clause sets. The column "$|\Delta|$" gives the size of the finite domains uniformly used in the problem clauses, e.g., $|\Delta| = 50$ means the range $[1..50]$. The column "#Iter" is the number of while-loop iterations in checkSAT needed to solve the problem for the given Δ. The column "#TP" is the number of theorem prover calls (Beagle calls) stemming from the various $\mathcal{B}$-satisfiability checks in checkSAT/find. Finally, "Time" is

[3] http://users.cecs.anu.edu.au/ baumgart/systems/beagle/

#	Problem	$\lvert\varDelta\rvert$	#Iter	#TP	Time
1	$\mathsf{f}(x) > 1 + y \lor y < 0 \lor x \notin \varDelta$	any	1	1	<1
2	$\mathsf{g}(x) \approx x \lor \mathsf{g}(x) \approx x + 1 \lor \neg(x \geq 0)$	10	9	32	5.5
	$\mathsf{g}(x) \approx -x \lor \neg(x < 0)$	20	20	86	55
	$\mathsf{f}(x) < \mathsf{g}(x) \lor x \notin \varDelta$				
3	$\mathsf{f}(x_1, x_2, x_3, x_4) > x_1 + x_2 + x_3 + x_4 \lor$ $x_1 \notin \varDelta \lor x_2 \notin \varDelta \lor x_3 \notin \varDelta \lor x_4 \notin \varDelta$	any	1	1	<1

	4- see caption			5- see Section 1			6- see Example 3.2			6alt- see text		
$\lvert\varDelta\rvert$	#Iter	#TP	Time	#Iter	#TP	Time	#Iter	#TP	Time	#Iter	#TP	Time
10	2	5	<1	3	15	2.3	3	12	<1	5	25	1.5
20	2	6	<1	3	17	2.6	3	14	<1	15	87	4.4
50	2	8	<1	3	19	2.8	3	19	1.1	34	239	23
100	2	9	<1	3	21	2.8	3	21	1.1	59	456	181
200	2	10	<1	3	23	2.8	3	23	1.2			
500	2	11	<1	3	25	2.9	3	24	1.2			
1000	2	12	<1	3	27	3.0	3	26	1.3			
2000	2	13	<1	3	29	3.0	3	28	1.4			
5000	2	15	<1	3	33	3.5	3	32	1.5			

Table 1. Experimental results. Problem 4 is $\{\mathsf{f}(x) \not\approx x \lor x \notin \varDelta,\ \mathsf{f}(5) \approx 8,\ \mathsf{f}(8) \approx 5\}$

the total CPU time needed to solve the problem. All experiments were carried out on a Linux desktop with a quad-core Intel i7 cpu running at 2.8 GHz. For comparison, we have also run Microsoft's SMT-solver Z3 [17], version 4.1, on our examples, using the obvious formula representation of the domains $\varDelta$.

Some comments on the individual problems. Problem (1) is trivially solved, for any $\varDelta$. In fact, the default interpretation is sufficient for that. Notice that the variable y is not finitely quantified (and does not need to be). Z3 reports "unknown" on problem (1), but, surprisingly it solves the essentially same problem $\mathsf{f}(x) > y \lor y < 0$ quickly. Problem (2) is meant to showcase our algorithm in conjunction with Beagle's theorem proving capabilities. The function symbol g is "sufficiently complete" defined by the first two clauses, and only the third clause containing the function symbol f needs finite quantification. Z3 could not solve this problem within three minutes. We devised problem (3) to get some insight into Z3's capabilities on the problems we are interested in. While it is trivial for our approach, Z3 seems to instantiate the clause in problem (3). Clearly, there is a scalability issue here, as for about $\lvert\varDelta\rvert > 60$ the problem becomes unsolvable in reasonable time.

As a side note, we found Z3s performance impressive, and it could solve problems (4)–(6) in very short time. Indeed, we plan to integrate Z3 in our approach and expect much better performance on many problems (Beagle's theory reasoning component is a rather slow implementation of Cooper's quantifier elimination algorithm.)

Problem (4) is a simple test of the default interpretation/exception mechanism. Problem (5) is the one in the Introduction, and problem (6) is our running example.

The problems (4), (5) and (6) scale very well, as expected. The first two are proven satisfiable using the default interpretation and a fixed number of exception points. In problem (4) these are easily discovered from the problem and in (5) the exceptions are quickly discovered by the search. Similarly, in problem (6) the definition for $\mathsf{f}(9)$ is found quickly, which is the only one needed to establish unsatisfiability. However, this requires to search first the domain of x_1, then x_2 (cf. Example 3.2). With the other way round we obtain much worse scaling behavior, cf. the entry "6alt" in Table 1.

6 Conclusions

We have presented a method for deciding hierarchic satisfiability, or satisfiability modulo theories, of first-order clause sets where all variables are quantified over finite subsets of background domains. The method tries to construct a model by stepwise amending a default interpretation in a conflict-driven way by utilizing a decision procedure for the EA-fragment of the background theory. It may also terminate with a set of ground instances witnessing that no model exists. For space reasons and for clarity we have focused in this paper on the basic principles and leave extensions for future work. Here are some ideas.

Richer input language: One important extension concerns foreground-sorted variables and operators, like the array-sorted variable a and the write-operator in clauses (1) and (2) in the introduction. In the example we got away without further modifications because the axioms (1) and (2) do not pose problems for sufficient completeness and for termination of hierarchic superposition. The question is under which conditions this is possible in general. One could also try to enumerate finite segments of the foreground domains in a Herbrand fashion, similarly as with background domains.

Our method can also be applied to certain richer syntactic fragments that require a full-fledged theorem prover for hierarchic specifications instead of a decision procedure for the background theory. However, this would "reverse" the common architecture by invoking that foreground reasoner from within an outer loop. This is problematic, however, because the foreground reasoner might not terminate or be incomplete. To fix that, it should be possible, under certain conditions, to instead integrate the checkSAT as an inference rule into, say, hierarchic superposition and apply it only to finitely quantified clauses as defined above. (This would directly generalize the Define-rule in [7].)

Alternative default interpretation: Taking the constant function as the default interpretation for free BG-sorted operators is not always a good choice. For example, for the clause $\mathsf{f}(x) \approx x \vee x \notin [1..1000]$ our method needs to amend the default interpretation at every point. Fortunately, any interpretation can be used as a default, and the identity function as the default interpretation for f leads immediately to a model. (On the other hand, in this example f is already sufficiently defined and could possibly be excepted from the transformation in the first place.)

Bernays-Schönfinkel fragment: The hierarchic superposition calculus can immediately be instantiated with, say, an instance-based method for deciding background theories that are given as a set of EPR-clauses. Our method, or the extensions above, could possibly be used to integrate arithmetic reasoners, instance-based methods and superposition in a beneficial way.

References

1. E. Althaus, E. Kruglov, and C. Weidenbach. Superposition modulo linear arithmetic SUP(LA). In S. Ghilardi and R. Sebastiani, eds., *FroCos*, 2009, *LNCS 5749*. Springer.
2. L. Bachmair and H. Ganzinger. Rewrite-based equational theorem proving with selection and simplification. *Journal of Logic and Computation*, 4(3):217–247, 1994.
3. L. Bachmair, H. Ganzinger, and U. Waldmann. Refutational theorem proving for hierarchic first-order theories. *Appl. Algebra Eng. Commun. Comput*, 5:193–212, 1994.
4. P. Baumgartner, A. Fuchs, H. de Nivelle, and C. Tinelli. Computing finite models by reduction to function-free clause logic. *Journal of Applied Logic*, 7(1):58–74, 2009.
5. P. Baumgartner and C. Tinelli. Model evolution with equality modulo built-in theories. In N. Bjoerner and V. Sofronie-Stokkermans, eds., *CADE-23*, 2011, *LNAI 6803*, Springer.
6. P. Baumgartner and U. Waldmann. Hierarchic superposition: Completeness without compactness. In M. Kosta and T. Sturm, eds., *MACIS*, 2013.
7. P. Baumgartner and U. Waldmann. Hierarchic superposition with weak abstraction. In M. P. Bonacina, ed., *CADE-24*, 2013, *LNAI 7898*, Springer.
8. K. Claessen and N. Sörensson. New techniques that improve MACE-style finite model building. In P. Baumgartner and C. G. Fermüller, eds., *CADE-19 Workshop: Model Computation – Principles, Algorithms, Applications*, 2003.
9. H. Ganzinger and K. Korovin. Theory instantiation. In M. Hermann, ed., *LPAR'06*, 2006, *LNCS 4246*, Springer.
10. Y. Ge, C. Barrett, and C. Tinelli. Solving quantified verification conditions using satisfiability modulo theories. In F. Pfenning, ed., *CADE-21*, 2007, *LNCS 4603*, Springer.
11. Y. Ge and L. M. de Moura. Complete instantiation for quantified formulas in satisfiabiliby modulo theories. In A. Bouajjani and O. Maler, eds., *CAV*, 2009, *LNCS 5643*, Springer.
12. J. Halpern. Presburger Arithmetic With Unary Predicates is Π^1_1-Complete. *Journal of Symbolic Logic*, 56(2):637–642, 1991.
13. K. Korovin and A. Voronkov. Integrating linear arithmetic into superposition calculus. In J. Duparc and T. A. Henzinger, eds., *CSL'07*, 2007, *LNCS 4646*, Springer.
14. E. Kruglov and C. Weidenbach. Superposition decides the first-order logic fragment over ground theories. *Mathematics in Computer Science*, pp. 1–30, 2012.
15. W. McCune. Mace4 reference manual and guide. Technical Report ANL/MCS-TM-264, Argonne National Laboratory, 2003.
16. L. M. de Moura and N. Bjørner. Efficient E-matching for SMT solvers. In F. Pfenning, ed., *CADE-21*, 2007, *LNCS 4603*, Springer.
17. L. M. de Moura and N. Bjørner. Z3: An efficient SMT solver. In C. R. Ramakrishnan and J. Rehof, eds., *TACAS*, 2008, *LNCS 4963*, Springer.
18. R. Nieuwenhuis, A. Oliveras, and C. Tinelli. Solving SAT and SAT modulo theories: from an abstract Davis-Putnam-Logemann-Loveland Procedure to DPLL(T). *Journal of the ACM*, 53(6):937–977, 2006.
19. A. Reynolds, C. Tinelli, A. Goel, and S. Krstić. Finite model finding in SMT. In N. Sharygina and H. Veith, eds., *CAV*, 2013, *LNCS 8044*, Springer.
20. A. Reynolds, C. Tinelli, A. Goel, S. Krstić, M. Deters, and C. Barrett. Quantifier instantiation techniques for finite model finding in SMT. In M. P. Bonacina, ed., *CADE-13*, 2013, *LNCS 7898*, Springer.
21. P. Rümmer. A constraint sequent calculus for first-order logic with linear integer arithmetic. In I. Cervesato, H. Veith, and A. Voronkov, eds., *LPAR'08*, 2008, *LNAI 5330*, Springer.
22. J. Slaney. Finder (finite domain enumerator): Notes and guide. Technical Report TR-ARP-1/92, Australian National University, Automated Reasoning Project, Canberra, 1992.
23. J. Zhang and H. Zhang. SEM: a system for enumerating models. In C. Mellish, ed., *IJCAI-95*, 1995, Morgan Kaufmann.