Deciding Local Theory Extensions via E-matching

Kshitij Bansal[1], Andrew Reynolds[2], Tim King[3],
Clark Barrett[1], and Thomas Wies[1]

[1] NYU
[2] EPFL
[3] Verimag

To the memory of Morgan Deters

Abstract. Satisfiability Modulo Theories (SMT) solvers incorporate decision procedures for theories of data types that commonly occur in software. This makes them important tools for automating verification problems. A limitation frequently encountered is that verification problems are often not fully expressible in the theories supported natively by the solvers. Many solvers allow the specification of application-specific theories as quantified axioms, but their handling is incomplete outside of narrow special cases.
In this work, we show how SMT solvers can be used to obtain complete decision procedures for local theory extensions, an important class of theories that are decidable using finite instantiation of axioms. We present an algorithm that uses E-matching to generate instances incrementally during the search, significantly reducing the number of generated instances compared to eager instantiation strategies. We have used two SMT solvers to implement this algorithm and conducted an extensive experimental evaluation on benchmarks derived from verification conditions for heap-manipulating programs. We believe that our results are of interest to both the users of SMT solvers as well as their developers.

1 Introduction

Satisfiability Modulo Theories (SMT) solvers are a cornerstone of today's verification technology. Common applications of SMT include checking verification conditions in deductive verification [14, 26], computing program abstractions in software model checking [1, 9, 27], and synthesizing code fragments in software synthesis [5, 6]. Ultimately, all these tasks can be reduced to satisfiability of formulas in certain first-order theories that model the semantics of prevalent data types and software constructs, such as integers, bitvectors, and arrays. The appeal of SMT solvers is that they implement decision procedures for efficiently reasoning about formulas in these theories. Thus, they can often be used off the shelf as automated back-end solvers in verification tools.

Some verification tasks involve reasoning about universally quantified formulas, which goes beyond the capabilities of the solvers' core decision procedures. Typical examples include verification of programs with complex data structures and concurrency, yielding formulas that quantify over unbounded sets of memory locations or thread identifiers. From a logical perspective, these quantified formulas can be thought of as axioms of application-specific theories. In practice, such theories often remain

within decidable fragments of first-order logic [2,7,9,23]. However, their narrow scope (which is typically restricted to a specific program) does not justify the implementation of a dedicated decision procedure inside the SMT solver. Instead, many solvers allow theory axioms to be specified directly in the input constraints. The solver then provides a quantifier module that is designed to heuristically instantiate these axioms. These heuristics are in general incomplete and the user is given little control over the instance generation. Thus, even if there exists a finite instantiation strategy that yields a decision procedure for a specific set of axioms, the communication of strategies and tactics to SMT solvers is a challenge [12]. Further, the user cannot communicate the completeness of such a strategy. In this situation, the user is left with two alternatives: either she gives up on completeness, which may lead to usability issues in the verification tool, or she implements her own instantiation engine as a preprocessor to the SMT solver, leading to duplication of effort and reduced solver performance.

The contributions of this paper are two-fold. First, we provide a better understanding of how complete decision procedures for application-specific theories can be realized with the quantifier modules that are implemented in SMT solvers. Second, we explore several extensions of the capabilities of these modules to better serve the needs of verification tool developers. The focus of our exploration is on *local theory extensions* [21,36]. A theory extension extends a given base theory with additional symbols and axioms. Local theory extensions are a class of such extensions that can be decided using finite quantifier instantiation of the extension axioms. This class is attractive because it is characterized by proof and model-theoretic properties that abstract from the intricacies of specific quantifier instantiation techniques [15,20,36]. Also, many well-known theories that are important in verification but not commonly supported by SMT solvers are in fact local theory extensions, even if they have not been presented as such in the literature. Examples include the array property fragment [8], the theory of reachability in linked lists [25,32], and the theories of finite sets [39] and multisets [38].

We present a general decision procedure for local theory extensions that relies on E-matching, one of the core components of the quantifier modules in SMT solvers. We have implemented our decision procedure using the SMT solvers CVC4 [3] and Z3 [11] and applied it to a large set of SMT benchmarks coming from the deductive software verification tool GRASShopper [29,31]. These benchmarks use a hierarchical combination of local theory extensions to encode verification conditions that express correctness properties of programs manipulating complex heap-allocated data structures. Guided by our experiments, we developed generic optimizations in CVC4 that improve the performance of our base-line decision procedure. Some of these optimizations required us to implement extensions in the solver's quantifier module. We believe that our results are of interest to both the users of SMT solvers as well as their developers. For users we provide simple ways of realizing complete decision procedures for application-specific theories with today's SMT solvers. For developers we provide interesting insights that can help them further improve the completeness and performance of today's quantifier instantiation modules.

Related work. Sofronie-Stokkermans [36] introduced local theory extensions as a generalization of locality in equational theories [15, 18]. Further generalizations include Psi-local theories [21], which can describe arbitrary theory extensions that admit finite

quantifier instantiation. The formalization of our algorithm targets local theory extensions, but we briefly describe how it can be generalized to handle Psi-locality. The original decision procedure for local theory extensions presented in [36], which is implemented in H-Pilot [22], eagerly generates all instances of extension axioms upfront, before the base theory solver is called. As we show in our experiments, eager instantiation is prohibitively expensive for many local theory extensions that are of interest in verification because it results in a high degree polynomial blowup in the problem size.

In [24], Swen Jacobs proposed an incremental instantiation algorithm for local theory extensions. The algorithm is a variant of model-based quantifier instantiation (MBQI). It uses the base theory solver to incrementally generate partial models from which relevant axiom instances are extracted. The algorithm was implemented as a plug-in to Z3 and experiments showed that it helps to reduce the overall number of axiom instances that need to be considered. However, the benchmarks were artificially generated. Jacob's algorithm is orthogonal to ours as the focus of this paper is on how to use SMT solvers for deciding local theory extensions without adding new substantial functionality to the solvers. A combination with this approach is feasible as we discuss in more detail below.

Other variants of MBQI include its use in the context of finite model finding [33], and the algorithm described in [17], which is implemented in Z3. This algorithm is complete for the so-called almost uninterpreted fragment of first-order logic. While this fragment is not sufficiently expressive for the local theory extensions that appear in our benchmarks, it includes important fragments such as Effectively Propositional Logic (EPR). In fact, we have also experimented with a hybrid approach that uses our E-matching-based algorithm to reduce the benchmarks first to EPR and then solves them with Z3's MBQI algorithm.

E-matching was first described in [28], and since has been implemented in various SMT solvers [10, 16]. In practice, user-provided *triggers* can be given as hints for finer grained control over quantifier instantiations in these implementations. More recent work [13] has made progress towards formalizing the semantics of triggers for the purposes of specifying decision procedures for a number of theories. A more general but incomplete technique [34] addresses the prohibitively large number of instantiations produced by E-matching by prioritizing instantiations that lead to ground conflicts.

2 Example

We start our discussion with a simple example that illustrates the basic idea behind local theory extensions. Consider the following set of ground literals

$$G = \{a + b = 1, f(a) + f(b) = 0\}.$$

We interpret G in the theory of linear integer arithmetic and a monotonically increasing function $f : \mathbb{Z} \to \mathbb{Z}$. One satisfying assignment for G is:

$$a = 0, b = 1, f(x) = \{-1 \text{ if } x \leq 0, 1 \text{ if } x > 0\}. \tag{1}$$

We now explain how we can use an SMT solver to conclude that G is indeed satisfiable in the above theory.

SMT solvers commonly provide inbuilt decision procedures for common theories such as the theory of linear integer arithmetic (`LIA`) and the theory of equality over uninterpreted functions (`UF`). However, they do not natively support the theory of monotone functions. The standard way to enforce f to be monotonic is to axiomatize this property,

$$K = \forall x, y.\, x \leq y \implies f(x) \leq f(y), \tag{2}$$

and then let the SMT solver check if $G \cup \{K\}$ is satisfiable via a reduction to its natively supported theories. In our example, the reduction target is the combination of `LIA` and `UF`, which we refer to as the *base theory*, denoted by $\mathcal{T}_0$. We refer to the axiom K as a *theory extension* of the base theory and to the function symbol f as an *extension symbol*.

Most SMT solvers divide the work of deciding ground formulas G in a base theory $\mathcal{T}_0$ and axioms $\mathcal{K}$ of theory extensions between different modules. A quantifier module looks for substitutions to the variables within an axiom K, x and y, to some ground terms, t_1 and t_2. We denote such a substitution as $\sigma = \{x \mapsto t_1, y \mapsto t_2\}$ and the instance of an axiom K with respect to this substitution as $K\sigma$. The quantifier module iteratively adds the generated ground instances $K\sigma$ as lemmas to G until the base theory solver derives a contradiction. However, if G is satisfiable, as in our case, then the quantifier module does not know when to stop generating instances of K, and the solver may diverge, effectively enumerating an infinite model of G.

For a local theory extension, we can syntactically restrict the instances $K\sigma$ that need to be considered before concluding that G is satisfiable to a finite set of candidates. More precisely, a theory extension is called *local* if in order to decide satisfiability of $G \cup \{K\}$, it is sufficient to consider only those instances $K\sigma$ in which all ground terms already occur in G and K. The monotonicity axiom K is a local theory extension of $\mathcal{T}_0$. The local instances of K and G are:

$$\begin{aligned}
K\sigma_1 = a \leq b &\implies f(a) \leq f(b) \text{ where } \sigma_1 = \{x \mapsto a, y \mapsto b\},\\
K\sigma_2 = b \leq a &\implies f(b) \leq f(a) \text{ where } \sigma_2 = \{x \mapsto b, y \mapsto a\},\\
K\sigma_3 = a \leq a &\implies f(a) \leq f(a) \text{ where } \sigma_3 = \{x \mapsto a, y \mapsto a\}, \text{ and}\\
K\sigma_4 = b \leq b &\implies f(b) \leq f(b) \text{ where } \sigma_4 = \{x \mapsto b, y \mapsto b\}.
\end{aligned}$$

Note that we do not need to instantiate x and y with other ground terms in G, such as 0 and 1. Adding the above instances to G yields

$$G' = G \cup \{K\sigma_1, K\sigma_2, K\sigma_3, K\sigma_4\}.$$

which is satisfiable in the base theory. Since K is a local theory extension, we can immediately conclude that $G \cup \{K\}$ is also satisfiable.

Recognizing Local Theory Extensions. There are two useful characterizations of local theory extensions that can help users of SMT solvers in designing axiomatization that are local. The first one is model-theoretic [15, 36]. Consider again the set of ground clauses G'. When checking satisfiability of G' in the base theory, the SMT solver may produce the following model:

$$a = 0, b = 1, f(x) = \{-1 \text{ if } x = 0, 1 \text{ if } x = 1, \text{-1 otherwise}\}. \tag{3}$$

This is not a model of the original $G \cup \{K\}$. However, if we restrict the interpretation of the extension symbol f in this model to the ground terms in $G \cup \{K\}$, we obtain the *partial model*

$$a = 0, b = 1, f(x) = \{-1 \text{ if } x = 0, 1 \text{ if } x = 1, \text{undefined otherwise}\}. \tag{4}$$

This partial model can now be embedded into the model (1) of the theory extension. If such embeddings of partial models of G' to total models of $G \cup \{K\}$ always exist for all sets of ground literals G, then K is a local theory extension of $\mathcal{T}_0$. The second characterization of local theory extensions is proof-theoretic and states that a set of axioms is a local theory extension if it is saturated under (ordered) resolution [4]. This characterization can be used to automatically compute local theory extensions from non-local ones [20].

Note that the locality property depends both on the base theory as well as the specific axiomatization of the theory extension. For example, the following axiomatization of a monotone function f over the integers, which is logically equivalent to equation (2) in $\mathcal{T}_0$, is not local:

$$K = \forall x.\, f(x) \leq f(x+1) \ .$$

Similarly, if we replace all inequalities in equation (2) by strict inequalities, then the extension is no longer local for the base theory $\mathcal{T}_0$. However, if we replace $\mathcal{T}_0$ by a theory in which $\leq$ is a dense order (such as in linear real arithmetic), then the strict version of the monotonicity axiom is again a local theory extension.

In the next two sections, we show how we can use the existing technology implemented in quantifier modules of SMT solvers to decide local theory extensions. In particular, we show how E-matching can be used to further reduce the number of axiom instances that need to be considered before we can conclude that a given set of ground literals G is satisfiable.

3 Preliminaries

Sorted first-order logic. We present our problem in sorted first-order logic with equality. A *signature* Σ is a tuple $(\mathsf{Sorts}, \Omega, \Pi)$, where Sorts is a countable set of sorts and Ω and Π are countable sets of function and predicate symbols, respectively. Each function symbol $f \in \Omega$ has an associated arity $n \geq 0$ and associated sort $s_1 \times \cdots \times s_n \to s_0$ with $s_i \in \mathsf{Sorts}$ for all $i \leq n$. Function symbols of arity 0 are called *constant symbols*. Similarly, predicate symbols $P \in \Pi$ have an arity $n \geq 0$ and sort $s_1 \times \cdots \times s_n$. We assume dedicated equality symbols $\approx_s \in \Pi$ with the sort $s \times s$ for all sorts $s \in \mathsf{Sorts}$, though we typically drop the explicit subscript. Terms are built from the function symbols in Ω and (sorted) variables taken from a countably infinite set X that is disjoint from Ω. We denote by $t : s$ that term t has sort s.

A Σ-atom A is of the form $P(t_1, \ldots, t_n)$ where $P \in \Pi$ is a predicate symbol of sort $s_1 \times \cdots \times s_n$ and the t_i are terms with $t_i : s_i$. A *Σ-formula* F is either a Σ-atom A, $\neg F_1$, $F_1 \wedge F_2$, $F_1 \vee F_2$, or $\forall x : s.F_1$ where F_1 and F_2 are Σ-formulas. A *Σ-literal* L is either A or $\neg A$ for a Σ-atom A. A *Σ-clause* C is a disjunction of Σ-literals. A Σ-term, atom, or formula is said to be *ground*, if no variable appears in it. For a set of formulas $\mathcal{K}$, we denote by $\mathsf{st}(\mathcal{K})$ the set of all ground subterms that appear in $\mathcal{K}$.

A Σ-sentence is a Σ-formula with no free variables where the free variables of a formula are defined in the standard fashion. We typically omit Σ if it is clear from the context.

Structures. Given a signature $\Sigma = (\mathsf{Sorts}, \Omega, \Pi)$, a Σ-*structure* M is a function that maps each sort $s \in \mathsf{Sorts}$ to a non-empty set $M(s)$, each function symbol $f \in \Omega$ of sort $s_1 \times \cdots \times s_n \to s_0$ to a function $M(f) : M(s_1) \times \cdots \times M(s_n) \to M(s_0)$, and each predicate symbol $P \in \Pi$ of sort $s_1 \times \cdots \times s_n$ to a relation $M(s_1) \times \cdots \times M(s_n)$. We assume that all structures M interpret each symbol $\approx_s$ by the equality relation on $M(s)$. For a Σ-structure M where Σ extends a signature Σ_0 with additional sorts and function symbols, we write $M|_{\Sigma_0}$ for the Σ_0-structure obtained by restricting M to Σ_0.

Given a structure M and a *variable assignment* $\nu : X \to M$, the evaluation $t^{M,\nu}$ of a term t in M, ν is defined as usual. For a structure M and an atom A of the form $P(t_1, \ldots, t_n)$, (M, ν) satisfies A iff $(t_1^{M,\nu}, \ldots, t_n^{M,\nu}) \in M(P)$. This is written as $(M, \nu) \models A$. From this satisfaction relation of atoms and Σ-structures, we can derive the standard notions of the satisfiability of a formula, satisfying a set of formulas $(M, \nu) \models \{F_i\}$, validity $\models F$, and entailment $F_1 \models F_2$. If a Σ-structure M satisfies a Σ-sentence F, we call M a model of F.

Theories and theory extensions. A *theory* $\mathcal{T}$ over signature Σ is a set of Σ-structures. We call a Σ-sentence K an *axiom* if it is the universal closure of a Σ-clause, and we denote a set of Σ-axioms as $\mathcal{K}$. We consider theories $\mathcal{T}$ defined as a class of Σ-structures that are models of a given set of Σ-sentences $\mathcal{K}$.

Let $\Sigma_0 = (\mathsf{Sorts}_0, \Omega_0, \Pi)$ be a signature and assume that the signature $\Sigma_1 = (\mathsf{Sorts}_0 \cup \mathsf{Sorts}_e, \Omega_0 \cup \Omega_e, \Pi)$ extends Σ_0 by new sorts Sorts_e and function symbols Ω_e. We call the elements of Ω_e *extension symbols* and terms starting with extension symbols *extension terms*. Given a Σ_0-theory $\mathcal{T}_0$ and Σ_1-axioms $\mathcal{K}_e$, we call $(\mathcal{T}_0, \mathcal{K}_e, \mathcal{T}_1)$ the *theory extension* of $\mathcal{T}_0$ with $\mathcal{K}_e$, where $\mathcal{T}_1$ is the set of all Σ_1-structures M that are models of $\mathcal{K}_e$ and whose reducts $M|_{\Sigma_0}$ are in $\mathcal{T}_0$. We often identify the theory extension with the theory $\mathcal{T}_1$.

4 Problem

We formally define the problem of satisfiability modulo theory and the notion of local theory extensions in this section.

Let $\mathcal{T}$ be a theory over signature Σ. Given a Σ-formula ϕ, we say ϕ is satisfiable modulo $\mathcal{T}$ if there exists a structure M in $\mathcal{T}$ and an assignment ν of the variables in ϕ such that $(M, \nu) \models \phi$. We define the ground satisfiability modulo theory problem as the corresponding decision problem for quantifier-free formulas.

Problem 1 (Ground satisfiability problem for Σ-theory $\mathcal{T}$).

input: A quantifier-free Σ-formula ϕ.
output: sat if ϕ is satisfiable modulo $\mathcal{T}$, unsat otherwise.

We say the satisfiability problem for $\mathcal{T}$ is *decidable* if there exists a procedure for the above problem which always terminates with sat or unsat. We write entailment modulo a theory as $\phi \models_{\mathcal{T}} \psi$.

We say an axiom of a theory extension is *linear* if all the variables occur under at most one extension term. We say it is *flat* if there there is no nesting of terms containing variables. It is easy to linearize and flatten the axioms by using additional variables and equality. As an example, $\forall x.\phi$ with $f(x)$ and $f(g(x))$ as terms in F may be written as

$$\forall xyz.x \approx y \wedge z \approx g(y) \implies F'$$

where F' is obtained from F by replacing $f(g(x))$ with $f(z)$. For the remainder of the paper, we assume that all extension axioms $\mathcal{K}_e$ are flat and linear. For the simplicity of the presentation, we assume that if a variable appears below a function symbol then that symbol must be an extension symbol.

Definition 2 (Local theory extensions). *A theory extension* $(\mathcal{T}_0, \mathcal{K}_e, \mathcal{T}_1)$ *is* local *if for any set of ground* Σ_1*-literals* G*:* G *is satisfiable modulo* $\mathcal{T}_1$ *if and only if* $G \cup \mathcal{K}_e[G]$ *is satisfiable modulo* $\mathcal{T}_0$ *extended with free function symbols. Here* $\mathcal{K}_e[G]$ *is the set of instances of* $\mathcal{K}_e$ *where the subterms of the instantiation are all subterms of* G *or* $\mathcal{K}_e$ *(in other words, they do not introduce new terms).*

For simplicity, in the rest of this paper, we work with theories $\mathcal{T}_0$ which have decision procedures for not just $\mathcal{T}_0$ but also $\mathcal{T}_0$ extended with free function symbols. Thus, we sometimes talk of satisfiability of a Σ_1-formula with respect a Σ_0-theory $\mathcal{T}_0$, to mean satisfiability in the $\mathcal{T}_0$ with the extension symbols in Σ_1 treated as free function symbols. In terms of SMT, we only talk of extensions of theories containing uninterpreted functions (`UF`).

A naive decision procedure for ground SMT of a local theory extension $\mathcal{T}_1$ is thus to generate all possible instances of the axioms $\mathcal{K}_e$ that do not introduce new ground terms, thereby reducing to the ground SMT problem of $\mathcal{T}_0$ extended with free functions.

Hierarchical extensions. Note that local theory extensions can be stacked to form hierarchies $((\ldots((\mathcal{T}_0, \mathcal{K}_1, \mathcal{T}_1), \mathcal{K}_2, \mathcal{T}_2), \ldots), \mathcal{K}_n, \mathcal{T}_n)$. Such a hierarchical arrangement of extension axioms is often useful to modularize locality proofs. In such cases, the condition that variables are only allowed to occur below extension symbols (of the current extension) can be relaxed to any extension symbol of the current level or below. The resulting theory extension can be decided by composing procedures for the individual extensions. Alternatively, one can use a monolithic decision procedure for the resulting theory $\mathcal{T}_n$, which can also be viewed as a single local theory extension $(\mathcal{T}_0, \mathcal{K}_1 \cup \cdots \cup \mathcal{K}_n, \mathcal{T}_n)$. In our experimental evaluation, which involved hierarchical extensions, we followed the latter approach.

5 Algorithm

In this section, we describe a decision procedure for a local theory extension, say $(\mathcal{T}_0, \mathcal{K}_e, \mathcal{T}_1)$, which can be easily implemented in most SMT solvers with quantifier instantiation support. We describe our procedure $\mathfrak{D}_{\mathcal{T}_1}$ as a theory module in a typical

SMT solver architecture. For simplicity, we separate out the interaction between theory solver and core SMT solver. We describe the procedure abstractly as taking as input:

- the original formula ϕ,
- a set of extension axioms $\mathcal{K}_e$,
- a set of instantiations of axioms that have already been made, Z, and
- a set of $\mathcal{T}_0$ satisfiable ground literals G such that $G \models \phi \wedge (\bigwedge_{\psi \in Z} \psi)$, and
- a set equalities $E \subseteq G$ between terms.

It either returns

- sat, denoting that G is $\mathcal{T}_1$ satisfiable; or
- a new set of instantiations of the axioms, Z'.

For completeness, we describe briefly the way we envisage the interaction mechanism of this module in a DPLL(T) SMT solver. Let the input problem be ϕ. The SAT solver along with the theory solvers for $\mathcal{T}_0$ will find a subset of literals G from $\phi \wedge (\bigwedge_{\psi \in Z} \psi)$ such that its conjunction is satisfiable modulo $\mathcal{T}_0$. If no such satisfying assignment exists, the SMT solver stops with unsat. One can think of G as being simply the literals in ϕ on the SAT solver trail. G will be sent to $\mathfrak{D}_{\mathcal{T}_1}$ along with information known about equalities between terms. The set Z can be also thought of as internal state maintained by the $\mathcal{T}_1$-theory solver module, with new instances Z' sent out as theory lemmas and Z updated to $Z \cup Z'$ after each call to $\mathfrak{D}_{\mathcal{T}_1}$. If $\mathfrak{D}_{\mathcal{T}_1}$ returns sat, so does the SMT solver and stops. On the other hand, if it returns a new set of instances, the SMT solver continues the search to additionally satisfy these.

E-matching. In order to describe our procedure, we introduce the well-studied E-matching problem. Given a universally quantified Σ-sentence K, let $X(K)$ denote the quantified variables. Define a Σ-substitution σ of K to be a mapping from variables $X(K)$ to Σ-terms of corresponding sort. Given a Σ-term p, let $p\sigma$ denote the term obtained by substituting variables in p by the substitutions provided in σ. Two substitutions σ_1, σ_2 with the same domain X are equivalent modulo a set of equalities E if $\forall x \in X. E \models \sigma_1(x) \approx \sigma_2(x)$. We denote this as $\sigma_1 \sim_E \sigma_2$.

Problem 3 (E-matching).

input: A set of ground equalities E, a set of Σ-terms G, and patterns P.
output: The set of substitutions σ over the variables in p, modulo E, such that for all $p \in P$ there exists a $t \in G$ with $E \models t \approx p\sigma$.

E-matching is a well-studied problem, specifically in the context of SMT. An algorithm for E-matching that is efficient and backtrackable is described in [10]. We denote this procedure by $\mathfrak{E}$.

The procedure $\mathfrak{D}_{\mathcal{T}_1}(\phi, \mathcal{K}_e, Z, G, E)$ is given in Fig. 1. Intuitively, it adds all the new instances along the current search path that are required for local theory reasoning as given in Definition 2, but modulo equality. For each axiom K in $\mathcal{K}_e$, the algorithm looks for function symbols containing variables. For example, if we think of the monotonicity axiom in Sect. 2, these would be the terms $f(x)$ and $f(y)$. These terms serve as patterns for the E-matching procedure. Next, with the help of the E-matching algorithm, all *new*

$\mathfrak{D}_{\mathcal{T}_1}(\phi, \mathcal{K}_e, Z, G, E)$
Local variable: Z', initially an empty set.

1. For each $K \in \mathcal{K}_e$:
   (a) Define the set of patterns P to be the function symbols in K containing variables. We observe that because the axioms are linear and flat, these patterns are always of the form $f(x_1, \ldots, x_n)$ where f is an extension symbol and the x_i are quantified variables.
   (b) Run $\mathfrak{E}(E, G, P)$ obtaining substitutions $\mathcal{S}$. Without loss of generality, assume that $\sigma \in \mathcal{S}$ returned by the algorithm are such that $\mathsf{st}(K\sigma) \subseteq \mathsf{st}(G \cup \mathcal{K}_e)$. For the special case of the patterns in (a), for any σ not respecting the condition there exists one in the equivalence class that respects the condition. Formally, $\forall\sigma.\exists\sigma'.\sigma' \sim_E \sigma \wedge \mathsf{st}(K\sigma') \subseteq \mathsf{st}(G \cup \mathcal{K}_e)$. We make this assumption only for simplicity of arguments later in the paper. If one uses an E-matching procedure not respecting this constraint, our procedure will still be terminating and correct (albeit total number of instantiations suboptimal).
   (c) For each $\sigma \in \mathcal{S}$, if there exists no $K\sigma'$ in Z such that $\sigma \sim_E \sigma'$, then add $K\sigma$ to Z' as a new instantiation to be made.
2. If Z' is empty, return sat, else return Z'.

Fig. 1. Procedure $\mathfrak{D}_{\mathcal{T}_1}$

instances are computed (to be more precise, all instances for the axiom K in Z which are equivalent modulo $\sim_E$ are skipped). If there are no new instances for any axiom in $\mathcal{K}_e$, and the set G of literals implies ϕ, we stop with sat. as effectively we have that $G \cup \mathcal{K}_e[G]$ is satisfiable modulo $\mathcal{T}_0$. Otherwise, we return this set.

We note that though the algorithm $\mathfrak{D}_{\mathcal{T}_1}$ may *look* inefficient because of the presence of nested loops, keeping track of which substitutions have already happened, and which substitutions are new. However, in actual implementations all of this is taken care of by the E-matching algorithm. There has been significant research on fast, incremental algorithms for E-matching in the context of SMT, and one advantage of our approach is to be able to leverage this work.

Correctness. The correctness argument relies on two aspects: one, that if the SMT solver says sat (resp. unsat) then ϕ is satisfiable (resp. unsatisfiable) modulo $\mathcal{T}_1$, and second, that it terminates.

For the case where the output is unsat, the correctness follows from the fact that Z only contains instances of $\mathcal{K}_e$. The sat case is more tricky, but the main idea is that the set of instances made by $\mathfrak{D}_{\mathcal{T}_1}(\phi, \mathcal{K}_e, Z, G, E)$ are logically equivalent to $\mathcal{K}_e[G]$. Thus, when the solver stops, $G \cup \mathcal{K}_e[G]$ is satisfiable modulo $\mathcal{T}_0$. As a consequence, G is satisfiable modulo $\mathcal{T}_1$. Since $G \models \phi$, we have that ϕ is satisfiable modulo $\mathcal{T}_1$.

The termination relies on the fact that the instantiations returned by procedure $\mathfrak{D}_{\mathcal{T}_1}(\phi, \mathcal{K}_e, Z, G, E)$ do not add new terms, and they are always a subset of $\mathcal{K}_e[\phi]$. Since, $\mathcal{K}_e[\phi]$ is finite, eventually $\mathfrak{D}$ will stop making new instantiations. Assuming that we have a terminating decision procedure for the ground SMT problem of $\mathcal{T}_0$, we get a terminating decision procedure for $\mathcal{T}_1$.

Theorem 4. *An SMT solver with theory module $\mathfrak{D}_{\mathcal{T}_1}$ is a decision procedure for the satisfiability problem modulo $\mathcal{T}_1$.*

Psi-local theories. We briefly explain how our approach can be extended to the more general notion of Psi-local theory extensions [21]. Sometimes, it is not sufficient to consider only local instances of extension axioms to decide satisfiability modulo a theory extension. For example, consider the following set of ground literals:

$$G = \{f(a) = f(b), a \neq b\}$$

Suppose we interpret G in a theory of an injective function $f : S \to S$ with a partial inverse $g : S \to S$ for some set S. We can axiomatize this theory as a theory extension of the theory of uninterpreted functions using the axiom

$$K = \forall x, y.\, f(x) = y \implies g(y) = x \ .$$

G is unsatisfiable in the theory extension, but the local instances of K with respect to the ground terms $\mathsf{st}(G) = \{a, b, f(a), f(b)\}$ are insufficient to yield a contradiction in the base theory. However, if we consider the local instances with respect to the larger set of ground terms

$$\Psi(\mathsf{st}(G)) = \{a, b, f(a), f(b), g(f(a)), g(f(b))\},$$

then we obtain, among others, the instances

$$f(a) = f(b) \implies g(f(b)) = a \quad \text{and} \quad f(b) = f(a) \implies g(f(a)) = b \ .$$

Together with G, these instances are unsatisfiable in the base theory.

The set $\Psi(\mathsf{st}(G))$ is computed as follows:

$$\Psi(\mathsf{st}(G)) = \mathsf{st}(G) \cup \{\, g(f(t)) \mid t \in \mathsf{st}(G) \,\}$$

It turns out that considering local instances with respect to $\Psi(\mathsf{st}(G))$ is sufficient to check satisfiability modulo the theory extension K for arbitrary sets of ground clauses G. Moreover, $\Psi(\mathsf{st}(G))$ is always finite. Thus, we still obtain a decision procedure for the theory extension via finite instantiation of extension axioms. Psi-local theory extensions formalize this idea. In particular, if Ψ satisfies certain properties including monotonicity and idempotence, one can again provide a model-theoretic characterization of completeness in terms of embeddings of partial models. We refer the reader to [21] for the technical details.

To use our algorithm for deciding satisfiability of a set of ground literals G modulo a Psi-local theory extension $(\mathcal{T}_0, \mathcal{K}_e, \mathcal{T}_1)$, we simply need to add an additional preprocessing step in which we compute $\Psi(\mathsf{st}(G))$ and define $G' = G \cup \{\, \mathtt{instclosure}(t) \mid t \in \Psi(\mathsf{st}(G)) \,\}$ where `instclosure` is a fresh predicate symbol. Then calling our procedure for $\mathcal{T}_1$ with G' decides satisfiability of G modulo $\mathcal{T}_1$.

6 Implementation and Experimental Results

Benchmarks. We evaluated our techniques on a set of benchmarks generated by the deductive verification tool GRASShopper [19]. The benchmarks encode memory safety

and functional correctness properties of programs that manipulate complex heap-allocated data structures. The programs are written in a type-safe imperative language without garbage collection. The tool makes no simplifying assumptions about these programs like acyclicity of heap structures.

GRASShopper supports mixed specifications in (classical) first-order logic and separation logic (SL) [35]. The tool reduces the program and specification to verification conditions that are encoded in a hierarchical combination of (Psi-)local theory extensions. This hierarchy of extensions is organized as follows:

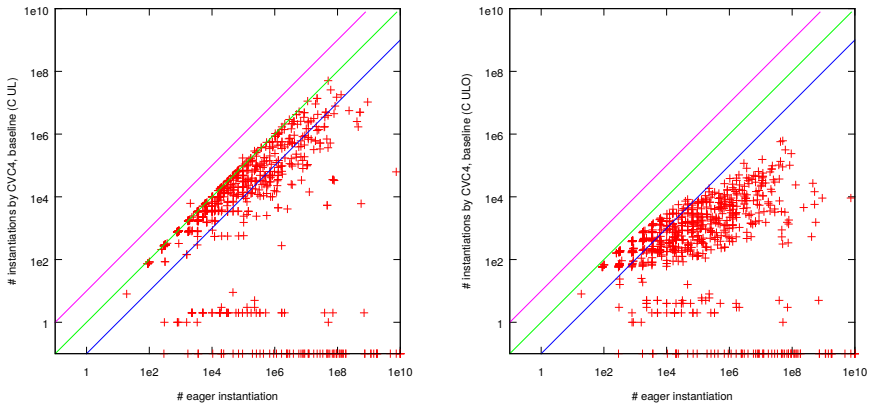
1. *Base theory:* at the lowest level we have `UFLIA`, the theory of uninterpreted functions and linear integer arithmetic, which is directly supported by SMT solvers.
2. *GRASS:* the first extension layer consists of the theory of graph reachability and stratified sets. This theory is a disjoint combination of two local theory extensions: the theory of linked lists with reachability [25] and the theory of sets over interpreted elements [39].
3. *Frame axioms:* the second extension layer consists of axioms that encode the frame rule of separation logic. This theory extension includes arrays as a subtheory.
4. *Program-specific extensions:* The final extension layer consists of a combination of local extensions that encode properties specific to the program and data structures under consideration. These include:
   - axioms defining memory footprints of SL specifications,
   - axioms defining structural constraints on the shape of data structures,
   - sorted constraints, and
   - axioms defining partial inverses of certain functions, e.g., to express injectivity of functions and to specify the content of data structures.

We refer the interested reader to [29–31] for further details about the encoding.

The programs considered include sorting algorithms, common data structure operations, such as inserting and removing elements, as well as complex operations on abstract data types. Our selection of data structures consists of singly and doubly-linked lists, sorted lists, nested linked lists with head pointers, binary search trees, skew heaps, and a union find data structure. The input programs comprise 108 procedures with a total of 2000 lines of code, 260 lines of procedure contracts and loop invariants, and 250 lines of data structure specifications (including some duplicate specifications that could be shared across data structures). The verification of these specifications are reduced by GRASShopper to 816 SMT queries, each serves as one benchmark in our experiments. 802 benchmarks are unsatisfiable. The remaining 14 satisfiable benchmarks stem from programs that have bugs in their implementation or specification. All of these are genuine bugs that users of GRASShopper made while writing the programs.[4] We considered several versions of each benchmark, which we describe in more detail below. Each of these versions is encoded as an SMT-LIB 2 input file.

Experimental setup. All experiments were conducted on the StarExec platform [37] with a CPU time limit of one hour and a memory limit of 100 GB. We focus on the SMT

[4] See `www.cs.nyu.edu/~kshitij/localtheories/` for the programs and benchmarks used.



(a) CVC4 with baseline algorithm (b) CVC4 with optimized algorithm

Fig. 2. # of eager instantiations vs. E-matching instantiations inside the solver

solvers CVC4 [3] and Z3 [11][5] as both support UFLIA and quantifiers via E-matching. This version of CVC4 is a fork of v1.4 with special support for quantifiers.[6]

In order to be able to test our approach with both CVC4 and Z3, wherever possible we transformed the benchmarks to simulate our algorithm. We describe these transformations in this paragraph. First, the quantified formulas in the benchmarks were linearized and flattened, and annotated with patterns to simulate Step 1(a) of our algorithm (this was done by GRASShopper in our experiments, but may also be handled by an SMT solver aware of local theories). Both CVC4 and Z3 support using these annotations for controlling instantiations in their E-matching procedures. In order to handle Psi-local theories, the additional terms required for completeness were provided as dummy assertions, so that these appear as ground terms to the solver. In CVC4, we also made some changes internally so as to treat these assertions specially and apply certain additional optimizations which we describe later in this section.

Experiment 1. Our first experiment aims at comparing the effectiveness of eager instantiation versus incremental instantiation up to congruence (as done by E-matching). Figure 2 charts the number of eager instantiations versus the number of E-matching instantiations for each query in a logarithmic plot.[7] Points lying on the central line have an equal number of instantiations in both series while points lying on the lower line have 10 times as many eager instantiations as E-matching instantiations. (The upper line corresponds to $\frac{1}{10}$.) Most benchmarks require substantially more eager instantiations. We instrumented GRASShopper to eagerly instantiate all axioms. Subfigure (a) compares upfront instantiations with a baseline implementation of our E-matching al-

[5] We used the version of Z3 downloaded from the git master branch at http://z3.codeplex.com on Jan 17, 2015.

[6] This version is available at www.github.com/kbansal/CVC4/tree/cav14-lte-draft.

[7] Figure 2 does not include timeouts for CVC4.

		C	UD	C	UL	C	ULO	Z3	UD	Z3	UL	Z3	ULO
family	#	#	time	#	time	#	time	#	time	#	time	#	time
sl lists	139	127	70	139	383	**139**	**17**	138	1955	138	1950	139	68
dl lists	70	66	1717	70	843	**70**	**33**	56	11375	56	11358	70	2555
sl nested	63	63	1060	63	307	**63**	**13**	52	6999	52	6982	59	1992
sls lists	208	181	6046	204	11230	**208**	**3401**	182	20596	182	20354	207	4486
trees	243	229	2121	228	22042	**239**	**7187**	183	41208	183	40619	236	27095
soundness	79	76	17	79	1533	**79**	**70**	76	7996	76	8000	79	336
sat	14	-	-	14	670	**14**	**12**	-	-	10	3964	14	898
total	816	742	11032	797	37009	**812**	**10732**	687	90130	697	93228	804	37430

Table 1. Comparison of solvers on uninstantiated benchmarks (time in sec.)

gorithm. Points along the x-axis required no instantiations in CVC4 to conclude unsat. We have plotted the above charts up to 10e10 instantiations. There were four outlying benchmarks where upfront instantiations had between 10e10 and up to 10e14 instances. E-matching had zero instantiations for all four. Subfigure (b) compares against an optimized version of our algorithm implemented in CVC4. It shows that incremental solving reduces the number of instantiations significantly, often by several orders of magnitude. The details of these optimizations are given later in the section.

Experiment 2. Next, we did a more thorough comparison on running times and number of benchmarks solved for *uninstantiated benchmarks*. These results are in Table 1. The benchmarks are partitioned according to the types of data structures occurring in the programs from which the benchmarks have been generated. Here, "sl" stands for singly-linked, "dl" for double-linked, and "sls" for sorted singly-linked. The binary search tree, skew heap, and union find benchmarks have all been summarized in the "trees" row. The row "soundness" contains unsatisfiable benchmarks that come from programs with incorrect code or specifications. These programs manipulate various types of data structures. The actual satisfiable queries that reveal the bugs in these programs are summarized in the "sat" row.

We simulated our algorithm and ran these experiments on both CVC4 (C) and Z3 obtaining similar improvements with both. We ran each with three configurations:

UD Default. For comparison purposes, we ran the solvers with default options. CVC4's default solver uses an E-matching based heuristic instantiation procedure, whereas Z3's uses both E-matching and model-based quantifier instantiation (MBQI). For both of the solvers, the default procedures are incomplete for our benchmarks.

UL These columns refer to the E-matching based complete procedure for local theory extensions (algorithm in Fig. 1).[8]

ULO Doing instantiations inside the solver instead of upfront, opens the room for optimizations wherein one tries some instantiations before others, or reduces the number of instantiations using other heuristics that do not affect completeness. The results in these columns show the effect of all such optimizations.

[8] The configuration C UL had one memory out on a benchmark in the tree family.

As noted above, the UL and ULO procedures are both complete, whereas UD is not. This is also reflected in the "sat" row in Table 1. Incomplete Instantiation-based procedures cannot hope to answer "sat". A significant improvement can be seen between the UL and ULO columns. The general thrust of the optimizations was to avoid blowup of instantiations by doing ground theory checks on a subset of instantiations. Our intuition is that the theory lemmas learned from these checks eliminate large parts of the search space before we do further instantiations.

For example, we used a heuristic for Psi-local theories inspired from the observation that the axioms involving Psi-terms are needed mostly for completeness, and that we can prove unsatisfiable without instantiating axioms with these terms most of the time. We tried an approach where the instantiations were staged. First, the instantiations were done according to the algorithm in Fig. 1 for locality with respect to ground terms from the original query. Only when those were saturated, the instantiations for the auxiliary Psi-terms were generated. We found this to be very helpful. Since this required non-trivial changes inside the solver, we only implemented this optimization in CVC4; but we think that staging instantiations for Psi-local theories is a good strategy in general.

A second optimization, again with the idea of cutting instantiations, was adding assertions in the benchmarks of the form $(a = b) \vee (a \neq b)$ where a, b are ground terms. This forces an arbitrary arrangement over the ground terms before the instantiation procedure kicks in. Intuitively, the solver first does checks with many terms equal to each other (and hence fewer instantiations) eliminating as much of the search space as possible. Only when equality or disequality is relevant to the reasoning is the solver forced to instantiate with terms disequal to each other. One may contrast this with ideas being used successfully in the care-graph-based theory combination framework in SMT where one needs to try all possible arrangements of equalities over terms. It has been observed that equality or disequality is sometimes relevant only for a subset of pairs of terms. Whereas in theory combination this idea is used to cut down the number of arrangements that need to be considered, we use it to reduce the number of unnecessary instantiations. We found this really helped CVC4 on many benchmarks.

Another optimization was instantiating special cases of the axioms first by enforcing equalities between variables of the same sort, before doing a full instantiation. We did this for axioms that yield a particularly large number of instances (instantiations growing with the fourth power of the number of ground terms). Again, we believe this could be a good heuristic in general.

Experiment 3. Effective propositional Logic (EPR) is the fragment of first order-logic consisting of formulas of the shape $\exists \boldsymbol{x} \forall \boldsymbol{y}.G$ with G quantifier-free and where none of the universally quantified variables $\boldsymbol{y}$ appears below a function symbol in G. Theory extensions that fall into EPR are always local. Our third exploration is to see if we can exploit dedicated procedures for this fragment when such fragments occur in the benchmarks. For the EPR fragment, Z3 has a complete decision procedure that uses model-based quantifier instantiation. We therefore implemented a hybrid approach wherein we did upfront partial instantiation to the EPR fragment using E-matching with respect to top-level equalities (as described in our algorithm). The resulting EPR benchmark is then decided using Z3's MBQI mode. This approach can only be expect to help where there are EPR-like axioms in the benchmarks, and we did have some which were

		C	PL	C	PLO	Z3	PM	Z3	PL	Z3	PLO
family	#	#	time	#	time	#	time	#	time	#	time
sl lists	139	139	664	139	20	**139**	**9**	139	683	139	29
dl lists	70	70	3352	70	50	**70**	**41**	67	12552	70	423
sl nested	63	63	2819	63	427	**63**	**182**	56	7068	62	804
sls lists	208	206	14222	207	3086	**208**	**37**	203	17245	208	1954
trees	243	232	7185	243	6558	**243**	**663**	222	34519	242	8089
soundness	79	78	156	79	49	**79**	**23**	79	2781	79	39
sat	14	14	85	**14**	**22**	13	21	12	1329	14	109
total	816	802	28484	**815**	10213	**815**	**976**	778	76177	814	11447

Table 2. Comparison of solvers on partially instantiated benchmarks (time in sec.)

heavier on these. We found that on singly linked list and tree benchmarks this hybrid algorithm significantly outperforms all other solver configurations that we have tried in our experiments. On the other hand, on nested list benchmarks, which make more heavy use of purely equational axioms, this technique does not help compared to only using E-matching because the partial instantiation already yields ground formulas.

The results with our hybrid algorithm are summarized in Column Z3 PM of Table 2. Since EPR is a special case of local theories, we also tried our E-matching based algorithm on these benchmarks. We found that the staged instantiation improves performance on these as well. The optimization that help on the uninstantiated benchmarks also work here. These results are summarized in the same table.

Overall, our experiments indicate that there is a lot of potential in the design of quantifier modules to further improve the performance of SMT solvers, and at the same time make them complete on more expressive decidable fragments.

7 Conclusion

We have presented a new algorithm for deciding local theory extensions, a class of theories that plays an important role in verification applications. Our algorithm relies on existing SMT solver technology so that it can be easily implemented in today's solvers. In its simplest form, the algorithm does not require any modifications to the solver itself but only trivial syntactic modifications to its input. These are: (1) flattening and linearizing the extension axioms; and (2) adding trigger annotations to encode locality constraints for E-matching. In our evaluation we have experimented with different configurations of two SMT solvers, implementing a number of optimizations of our base line algorithm. Our results suggest interesting directions to further improve the quantifier modules of current SMT solvers, promising better performance and usability for applications in automated verification.

Acknowledgments. We would like to thank the anonymous reviewers for their insightful comments and suggestions. This work was supported in part by the National Science Foundation under grants CNS-1228768 and CCF-1320583.

References

1. Aws Albarghouthi, Yi Li, Arie Gurfinkel, and Marsha Chechik. Ufo: A framework for abstraction- and interpolation-based software verification. In *CAV*, volume 7358 of *Lecture Notes in Computer Science*, pages 672–678. Springer, 2012.
2. Francesco Alberti, Silvio Ghilardi, and Natasha Sharygina. Decision procedures for flat array properties. In *TACAS*, volume 8413 of *LNCS*, pages 15–30. Springer, 2014.
3. Clark Barrett, Christopher L. Conway, Morgan Deters, Liana Hadarean, Dejan Jovanovic, Tim King, Andrew Reynolds, and Cesare Tinelli. Cvc4. In *CAV*, pages 171–177, 2011.
4. David A. Basin and Harald Ganzinger. Complexity analysis based on ordered resolution. In *Proceedings, 11th Annual IEEE Symposium on Logic in Computer Science, New Brunswick, New Jersey, USA, July 27-30, 1996*, pages 456–465. IEEE, 1996.
5. Tewodros A. Beyene, Swarat Chaudhuri, Corneliu Popeea, and Andrey Rybalchenko. A constraint-based approach to solving games on infinite graphs. In *POPL*, pages 221–234. ACM, 2014.
6. Rastislav Bodík and Emina Torlak. Synthesizing programs with constraint solvers. In *CAV*, volume 7358 of *LNCS*, page 3. Springer, 2012.
7. Ahmed Bouajjani, Cezara Dragoi, Constantin Enea, and Mihaela Sighireanu. Accurate invariant checking for programs manipulating lists and arrays with infinite data. In *ATVA*, volume 7561 of *LNCS*, pages 167–182. Springer, 2012.
8. Aaron R. Bradley, Zohar Manna, and Henny B. Sipma. What's decidable about arrays? In *VMCAI*, volume 3855 of *LNCS*, pages 427–442. Springer, 2006.
9. Angelo Brillout, Daniel Kroening, Philipp Rümmer, and Thomas Wahl. An interpolating sequent calculus for quantifier-free presburger arithmetic. *J. Autom. Reasoning*, 47(4):341–367, 2011.
10. Leonardo de Moura and Nikolaj Bjørner. Efficient E-Matching for SMT solvers. In *Automated Deduction - CADE-21, 21st International Conference on Automated Deduction, Bremen, Germany, July 17-20, 2007, Proceedings*, volume 4603 of *Lecture Notes in Computer Science*, pages 183–198. Springer, 2007.
11. Leonardo de Moura and Nikolaj Bjørner. Z3: An efficient SMT solver. In *TACAS*, pages 337–340, 2008.
12. Leonardo Mendonça de Moura and Grant Olney Passmore. The strategy challenge in SMT solving. In *Automated Reasoning and Mathematics - Essays in Memory of William W. McCune*, pages 15–44, 2013.
13. Claire Dross, Sylvain Conchon, Johannes Kanig, and Andrei Paskevich. Reasoning with triggers. In Pascal Fontaine and Amit Goel, editors, *SMT 2012*, volume 20 of *EPiC Series*, pages 22–31. EasyChair, 2013.
14. Jean-Christophe Filliâtre and Andrei Paskevich. Why3 - where programs meet provers. In *ESOP*, volume 7792 of *LNCS*, pages 125–128. Springer, 2013.
15. H. Ganzinger. Relating semantic and proof-theoretic concepts for polynomial time decidability of uniform word problems. In *Logic in Computer Science, 2001. Proceedings. 16th Annual IEEE Symposium on*, pages 81–90, 2001.
16. Yeting Ge, Clark Barrett, and Cesare Tinelli. Solving quantified verification conditions using satisfiability modulo theories. *Annals of Mathematics and Artificial Intelligence*, 55(1-2):101–122, February 2009.
17. Yeting Ge and Leonardo Moura. Complete instantiation for quantified formulas in satisfiabiliby modulo theories. In *Proceedings of the 21st International Conference on Computer Aided Verification*, CAV '09, pages 306–320, Berlin, Heidelberg, 2009. Springer-Verlag.
18. Robert Givan and David A. McAllester. New results on local inference relations. In *KR*, pages 403–412. Morgan Kaufmann, 1992.

19. GRASShopper tool web page. http://cs.nyu.edu/wies/software/grasshopper. Accessed: Feb 2015.
20. Matthias Horbach and Viorica Sofronie-Stokkermans. Obtaining finite local theory axiomatizations via saturation. In *FroCoS*, volume 8152 of *LNCS*, pages 198–213. Springer, 2013.
21. Carsten Ihlemann, Swen Jacobs, and Viorica Sofronie-Stokkermans. On local reasoning in verification. In *TACAS*, pages 265–281, 2008.
22. Carsten Ihlemann and Viorica Sofronie-Stokkermans. System description: H-pilot. In *CADE*, volume 5663 of *LNCS*, pages 131–139. Springer, 2009.
23. Shachar Itzhaky, Anindya Banerjee, Neil Immerman, Ori Lahav, Aleksandar Nanevski, and Mooly Sagiv. Modular reasoning about heap paths via effectively propositional formulas. In *POPL*, pages 385–396. ACM, 2014.
24. Swen Jacobs. Incremental instance generation in local reasoning. In *Proceedings of the 21st International Conference on Computer Aided Verification*, CAV '09, pages 368–382, Berlin, Heidelberg, 2009. Springer-Verlag.
25. Shuvendu K. Lahiri and Shaz Qadeer. Back to the future: revisiting precise program verification using SMT solvers. In *POPL*, pages 171–182, 2008.
26. K. Rustan M. Leino. Dafny: An automatic program verifier for functional correctness. In *Logic for Programming, Artificial Intelligence, and Reasoning - 16th International Conference, LPAR-16, Dakar, Senegal, April 25-May 1, 2010, Revised Selected Papers*, volume 6355 of *LNCS*, pages 348–370. Springer, 2010.
27. Kenneth L. McMillan. Interpolants from Z3 proofs. In *International Conference on Formal Methods in Computer-Aided Design, FMCAD '11, Austin, TX, USA, October 30 - November 02, 2011*, pages 19–27, 2011.
28. Charles Gregory Nelson. *Techniques for Program Verification*. PhD thesis, Stanford, CA, USA, 1980. AAI8011683.
29. Ruzica Piskac, Thomas Wies, and Damien Zufferey. Automating Separation Logic Using SMT. In *CAV*, volume 8044 of *LNCS*, pages 773–789. Springer, 2013.
30. Ruzica Piskac, Thomas Wies, and Damien Zufferey. Automating separation logic with trees and data. In *CAV*, volume 3855 of *LNCS*, pages 711–728. Springer, 2014.
31. Ruzica Piskac, Thomas Wies, and Damien Zufferey. GRASShopper: Complete Heap Verification with Mixed Specifications. In *TACAS*. Springer, 2014.
32. Zvonimir Rakamaric, Jesse D. Bingham, and Alan J. Hu. An inference-rule-based decision procedure for verification of heap-manipulating programs with mutable data and cyclic data structures. In *VMCAI*, volume 4349 of *LNCS*, pages 106–121. Springer, 2007.
33. Andrew Reynolds, Cesare Tinelli, Amit Goel, Sava Krstić, Morgan Deters, and Clark Barrett. Quantifier instantiation techniques for finite model finding in SMT. In M. P. Bonacina, editor, *Proceedings of the 24th International Conference on Automated Deduction (Lake Placid, NY, USA)*, volume 7898 of *Lecture Notes in Computer Science*, pages 377–391. Springer, 2013.
34. Andrew Reynolds, Cesare Tinelli, and Leonardo De Moura. Finding conflicting instances of quantified formulas in SMT. In *Formal Methods in Computer-Aided Design (FMCAD)*, 2014.
35. John C. Reynolds. Separation logic: a logic for shared mutable data structures. In *LICS*, pages 55–74, 2002.
36. Viorica Sofronie-Stokkermans. Hierarchic reasoning in local theory extensions. In *CADE-20*, volume 3632 of *LNCS*, pages 219–234. Springer, 2005.
37. Aaron Stump, Geoff Sutcliffe, and Cesare Tinelli. StarExec: a Cross-Community Infrastructure for Logic Solving. In *IJCAR*, pages 367–373, 2014.
38. Calogero G. Zarba. Combining multisets with integers. In *CADE-18*, 2002.
39. Calogero G. Zarba. Combining sets with elements. In *Verification: Theory and Practice, Essays Dedicated to Zohar Manna on the Occasion of His 64th Birthday*, volume 2772 of *LNCS*, pages 762–782. Springer, 2003.