LOCAL THEORY EXTENSIONS VIA E-MATCHING

Kshitij Bansal, New York University
Andrew Reynolds, EPFL
Tim King, Verimag
Clark Barrett, New York University
Thomas Wies, New York University

CAV, San Francisco, 23 Jul 2015
extern void __VERIFIER_error() __attribute__((noreturn));

int main() {
    unsigned int plus_one = 1;
    int minus_one = -1;

    if (plus_one < minus_one) {
        goto ERROR;
    }

    return (0);
    ERROR: __VERIFIER_error();
    return (-1);
}

(set-logic QF_BV)
(declare-const addr_of_plus_one (_ BitVec 32))
(declare-const plus_one (_ BitVec 32))
(declare-const addr_of_minus_one (_ BitVec 32))
(declare-const minus_one (_ BitVec 32))
(push)
(assert (and (bvult (_ bv1 32) (bvneg (_ bv1 32)))
    true))
(check-sat)
extern void __VERIFIER_error() __attribute__((__noreturn__));

int main() {
    unsigned int plus_one = 1;
    int minus_one = -1;

    if(plus_one < minus_one) {
        goto ERROR;
    }

    return (0);
ERROR: __VERIFIER_error();
return (-1);

(set-logic QF_BV)
(declare-const addr_of_plus_one (_ BitVec 32))
(declare-const plus_one (_ BitVec 32))
(declare-const addr_of_minus_one (_ BitVec 32))
(declare-const minus_one (_ BitVec 32))
(push)
(assert (and (bvult (_ bv1 32) (bvneg (_ bv1 32)))
true))
(check-sat)
BUT OFTEN ...

```c
/* Concatenate two lists 'a' and 'b'.
 * The result is a single list 'res'. */

procedure concat(a: Node, b: Node)
    returns (res: Node)
    requires lseg(a, null) && lseg(b, null)
    ensures lseg(res, null)
{
    if (a == null) {
        return b;
    } else {
        var curr: Node;
        curr := a;
        while (curr.next != null)
            invariant acc(curr) && lseg(a, null)
            {
                curr := curr.next;
            }
        curr.next := b;
        return a;
    }
}
```

```prolog
(set-logic UF)
...

(declare-fun Btwn ((Map (Loc Node) (Loc Node)) (Loc Node) (Loc Node) (Loc Node)) Bool)
...

(assert (forall ((?f (Map (Loc Node) (Loc Node) (Loc Node) (Loc Node))) (?x (Loc Node)) (?y (Loc Node)))
    (or (not (= (read ?f ?x) ?x)) (not (Btwn ?f ?x ?y ?y)) (= ?x ?y))))
...

(assert (or (and (= sk_?XNode_5 (lseg_footprint next b null))
    (Btwn next b null null)) (not (lseg next b null sk_?XNode_5)))
...

(check-sat)```
BUT OFTEN...

Quantified ...

∀x,y...

/* Concatenate two lists 'a' and 'b'. */
* The result is a single list 'res'. */

procedure concat(a: Node, b: Node)
  returns (res: Node)
  requires lseg(a, null) &
  ensures lseg(res, null)
{
  if (a == null) {
    return b;
  } else {
    var curr: Node;
    curr := a;
    while (curr.next != null)
      invariant acc(curr) **- lseg(a, null)
      { curr := curr.next;
    }
    curr.next := b;
    return a;
  }
}

(set-logic UF)
...
(declare-fun Btwn ((Map (Loc Node) (Loc Node)) (Loc Node)) (Loc Node) (Loc Node) Bool)

(assert (forall ((?f (Map (Loc Node) (Loc Node))) (?x (Loc Node)) (?y (Loc Node)))
  (or (not (read ?f ?x) ?y)) (not (Btwn ?f ?x ?y)) (= ?x ?y)))
...

(assert (or (and (= sk_?XNode_5 (lseg_footprint next b null))
  (Btwn next b null null))
  (lseg next b null sk_?XNode_5)))
...
(check-sat)
THIS WORK

- Local theory extensions [Sofronie-Stokkermans, 2005]
- How to use existing SMT solvers for a complete decision procedure
- Improvements in the solvers for better performance
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.

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

\[ G = \{a + b = 1, f(a) + f(b) = 0\} \]
\[
G = \{a + b = 1, f(a) + f(b) = 0\}
\]

Theory of linear arithmetic. \(f : \mathbb{Z} \rightarrow \mathbb{Z}\) monotonically increasing.
\[ G = \{a + b = 1, f(a) + f(b) = 0\} \]

Theory of linear arithmetic. \( f : \mathbb{Z} \rightarrow \mathbb{Z} \) monotonically increasing.

\[ \text{SAT: } \quad a = 0, b = 1, f(x) = \{-1 \text{ if } x \leq 0, 1 \text{ if } x > 0\} \]
\[
G = \{a + b = 1, \ f(a) + f(b) = 0\}
\]

Theory of linear arithmetic. \(f : \mathbb{Z} \rightarrow \mathbb{Z}\) monotonically increasing.

**SAT:** \(a = 0, b = 1, f(x) = \begin{cases} 
-1 & \text{if } x \leq 0 \\
1 & \text{if } x > 0
\end{cases}\)

\[
K = \forall x, y. \ x \leq y \implies f(x) \leq f(y)
\]
\[ G = \{ a + b = 1, \ f(a) + f(b) = 0 \} \]

Theory of linear arithmetic. \( f : \mathbb{Z} \rightarrow \mathbb{Z} \) monotonically increasing.

SAT: \( a = 0, \ b = 1, \ f(x) = \{-1 \text{ if } x \leq 0, \ 1 \text{ if } x > 0\} \)

\[ K = \forall x, y. \ x \leq y \implies f(x) \leq f(y) \]

**Local** if sufficient to instantiate such that all terms already exist in \( G \) or \( K \).
\[ G = \{a + b = 1, f(a) + f(b) = 0\} \]

Theory of linear arithmetic. \(f: \mathbb{Z} \rightarrow \mathbb{Z}\) monotonically increasing.

**SAT:** \(a = 0, b = 1, f(x) = \{-1 \text{ if } x \leq 0, 1 \text{ if } x > 0\}\)

\[ K = \forall x, y. x \leq y \implies f(x) \leq f(y) \]

\[ K\sigma_1 = a \leq b \implies f(a) \leq f(b) \text{ where } \sigma_1 = \{x \mapsto a, y \mapsto b\} \]
\[ K\sigma_2 = b \leq a \implies f(b) \leq f(a) \text{ where } \sigma_2 = \{x \mapsto b, y \mapsto a\} \]
\[ K\sigma_3 = a \leq a \implies f(a) \leq f(a) \text{ where } \sigma_3 = \{x \mapsto a, y \mapsto a\} \]
\[ K\sigma_4 = b \leq b \implies f(b) \leq f(b) \text{ where } \sigma_4 = \{x \mapsto b, y \mapsto b\}. \]
\[ G = \{a + b = 1, f(a) + f(b) = 0\} \]

Theory of linear arithmetic.

\[
\begin{align*}
K\sigma_1 &= a \leq b \implies f(a) \leq f(b) \quad \text{where } \sigma_1 = \{x \mapsto a, y \mapsto b\} \\
K\sigma_2 &= b \leq a \implies f(b) \leq f(a) \quad \text{where } \sigma_2 = \{x \mapsto b, y \mapsto a\} \\
K\sigma_3 &= a \leq a \implies f(a) \leq f(a) \quad \text{where } \sigma_3 = \{x \mapsto a, y \mapsto a\} \\
K\sigma_4 &= b \leq b \implies f(b) \leq f(b) \quad \text{where } \sigma_4 = \{x \mapsto b, y \mapsto b\}.
\end{align*}
\]
Theory of linear arithmetic.

\[
G = \{a + b = 1, f(a) + f(b) = 0\}
\]

\[G \cup K[G]\] is satisfiable in LIA if and only if \(G\) is satisfiable in LIA+\(K\)

\[
K[G] = \begin{cases} 
K\sigma_1 = a \leq b \implies f(a) \leq f(b) \text{ where } \sigma_1 = \{x \mapsto a, y \mapsto b\} \\
K\sigma_2 = b \leq a \implies f(b) \leq f(a) \text{ where } \sigma_2 = \{x \mapsto b, y \mapsto a\} \\
K\sigma_3 = a \leq a \implies f(a) \leq f(a) \text{ where } \sigma_3 = \{x \mapsto a, y \mapsto a\} \\
K\sigma_4 = b \leq b \implies f(b) \leq f(b) \text{ where } \sigma_4 = \{x \mapsto b, y \mapsto b\}.
\end{cases}
\]
\[ G = \{a + b = 1, f(a) + f(b) = 0\} \]

\[ K[G] = \{K\sigma_1, K\sigma_2, K\sigma_3, K\sigma_4\} \]
We start our discussion with a simple example that illustrates the basic idea behind local work. E-matching was first described in [28], and since has been implemented in various SMT solvers. In [24], Jacob's algorithm is orthogonal to ours as the focus of this paper is on 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 is satisfiable.

\[ G = \{a + b = 1, f(a) + f(b) = 0\} \]

\[ K[G] = \{K\sigma_1, K\sigma_2, K\sigma_3, K\sigma_4\} \]

\[ a = 0, b = 1, f(x) = \{-1 \text{ if } x = 0, 1 \text{ if } x = 1, -1 \text{ otherwise}\} \]

There are two useful characterizations of local theory extensions. The first is completeness for the so-called almost uninterpreted fragment of first-order logic. While this is complete for the base theory solver, the SMT solver may eagerly generate all instances of extension axioms upfront, which can be given as hints for the base theory solver. 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 [22], eagerly generates all instances of extension axioms upfront, which can help them further improve the completeness and performance of today's quantifier instantiation modules. However, the benchmarks were artificially generated. Jacob's algorithm is orthogonal to ours as the focus of this paper is on how E-matching can be used to further reduce the 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 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 is satisfiable.

\[ a = 0, b = 1, f(x) = \{-1 \text{ if } x = 0, 1 \text{ if } x = 1, -1 \text{ otherwise}\} \]
\[ G = \{ a + b = 1, f(a) + f(b) = 0 \} \]
\[ K[G] = \{ K\sigma_1, K\sigma_2, K\sigma_3, K\sigma_4 \} \]
\[ a = 0, b = 1, f(x) = \{-1 \text{ if } x = 0, 1 \text{ if } x = 1, -1 \text{ otherwise} \} \]

\[ a = 0, b = 1, f(x) = \{-1 \text{ if } x = 0, 1 \text{ if } x = 1, \text{ undefined otherwise} \} \]
Similarly, if we replace all inequalities in equation (3), is not local:

\[ G = \{a + b = 1, f(a) + f(b) = 0\} \]

\[ K[G] = \{K\sigma_1, K\sigma_2, K\sigma_3, K\sigma_4\} \]

\[ a = 0, b = 1, f(x) = \{-1 \text{ if } x = 0, 1 \text{ if } x = 1, -1 \text{ otherwise}\} \]

\[ a = 0, b = 1, f(x) = \{-1 \text{ if } x = 0, 1 \text{ if } x = 1, \text{undefined otherwise}\} \]

Can be embedded in full model of LIA+K

Embed
EXAMPLES

- Local theory extensions — more general than EPR
- Array property fragment [Bradley, Manna, Sipma, 2006]
- Theory of reachability in linked lists [Lahiri, Qadeer, 2006; Rakamafić, Bingham, Hu, 2007]
- Theory of finite sets and multisets [Zarba, 2004; Zarba 2002]
E-MATCHING

Nelson, 1980; Detlefs, Nelson, Saxe, 2005; deMoura, Bjørner, 2007

- **input:**
  - a set of terms $G$
  - a set of ground equalities $E$ ($t_1 \approx t_2$).
  - patterns $P$ (e.g. $f(x)$)

- **output:**
  - The set of substitutions $\sigma$ 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-MATCH**

*Nelson, 1980; Detlefs, Nelson, Saxe, 2005; deMoura, Bjørner, 2007*

---

**input:**
- a set of terms $G$
- a set of ground equalities $E$ ($t_1 \approx t_2$).
- patterns $P$ (e.g. $f(x)$)

**output:**
- The set of substitutions $\sigma$ over the variables in $p$, modulo $E$, such that:
  
  for all $p \in P$ there exists a $t \in G$ with $E \models t \approx p\sigma$. 

---

$G = \{a, b, c, f(a), f(b), f(c)\}$  
$E = \{a \approx b\}$  
$P = \{f(x), f(y)\}$
E-MATCHING

input:
- a set of terms \( G \)
- a set of ground equalities \( E \)
- patterns \( P \) (e.g. \( f(x) \))

output:
- The set of substitutions \( \sigma \) over the variables in \( P \), modulo \( E \), such that:
  
  for all \( p \in P \) there exists a \( t \in G \) with \( E \models t \approx p\sigma \).
EXAMPLE

\[ \varphi: \]

\[ a + b \approx 1 \]

\[ \land (f(a) + f(b) \approx 0 \lor f(b) + f(c) \approx 0) \]

\[ \land a + c \approx b + d \]

\[ \land c \approx d. \]
EXAMPLE

\[ K = \forall x, y. \ x \leq y \implies f(x) \leq f(y) \]

\[ \varphi: \]

\[ a + b \approx 1 \]

\[ \land (f(a) + f(b) \approx 0 \lor f(b) + f(c) \approx 0) \]

\[ \land a + c \approx b + d \]

\[ \land c \approx d. \]
EXAMPLE

\[
K = \forall x, y. x \leq y \implies f(x) \leq f(y)
\]

Terms: \(a, b, c, d, f(a), f(b), f(c), 0, 1\)

\[\varphi:\]

\[a + b \approx 1\]

\[\land (f(a) + f(b) \approx 0 \lor f(b) + f(c) \approx 0)\]

\[\land a + c \approx b + d\]

\[\land c \approx d.\]
### EXAMPLE

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

**Terms:** $$a, b, c, d, f(a), f(b), f(c), 0, 1$$

**Externally solve:** Instantiate such that all terms already exist in $$G$$ or $$K$$.

$$\varphi:$$

$$a + b \approx 1$$

$$\land (f(a) + f(b) \approx 0 \lor f(b) + f(c) \approx 0)$$

$$\land a + c \approx b + d$$

$$\land c \approx d.$$
EXAMPLE

\[ K = \forall x, y. x \leq y \implies f(x) \leq f(y) \]

Terms: \(a, b, c, d, f(a), f(b), f(c), 0, 1\)

\[ \varphi: \]

\[ a + b \approx 1 \]

\[ \land (f(a) + f(b) \approx 0 \lor f(b) + f(c) \approx 0) \]

\[ \land a + c \approx b + d \]

\[ \land c \approx d. \]

Externally solve: Instantiate such that all terms already exist in \(G\) or \(K\).

\(\{x \rightarrow a, b, c\} \times \{y \rightarrow a, b, c\}\)

Not \(d, 0, 1\) as \(f(.)\) not in \(G\) or \(K\).
EXAMPLE

\[ K = \forall x, y. x \leq y \implies f(x) \leq f(y) \]

\[ \varphi: \]

\[ a + b \approx 1 \]
\[ \land (f(a) + f(b) \approx 0 \lor f(b) + f(c) \approx 0) \]
\[ \land a + c \approx b + d \]
\[ \land c \approx d. \]
EXAMPLE

\[ K = \forall x, y. \ x \leq y \implies f(x) \leq f(y) \]

\( \varphi: \)

\[ a + b \approx 1 \]
\[ \land (f(a) + f(b) \approx 0 \lor f(b) + f(c) \approx 0) \]
\[ \land a + c \approx b + d \]
\[ \land c \approx d. \]
EXAMPLE

\[ K = \forall x, y. x \leq y \implies f(x) \leq f(y) \]

\[ a + b \approx 1 \]
\[ \land f(a) + f(b) \approx 0 \]
\[ \land a + c \approx b + d \]
\[ \land c \approx d. \]
EXAMPLE

\[ K = \forall x, y. x \leq y \implies f(x) \leq f(y) \]

\[
\begin{align*}
  a + b &\approx 1 \\
  f(a) + f(b) &\approx 0 \\
  a + c &\approx b + d \\
  c &\approx d. \\
\end{align*}
\]
EXAMPLE

\[ K = \forall x, y. \, x \leq y \implies f(x) \leq f(y) \]

\[ a + b \approx 1, \]
\[ f(a) + f(b) \approx 0, \]
\[ a + c \approx b + d, \]
\[ c \approx d, \]
\[ a \approx b. \]
EXAMPLE

\[ K = \forall x, y. \, x \leq y \implies f(x) \leq f(y) \]

\[ a + b \approx 1, \quad f(a) + f(b) \approx 0, \]
\[ a + c \approx b + d, \quad c \approx d, \quad a \approx b. \]

\[ K\sigma_1 = a \leq b \implies f(a) \leq f(b) \quad \text{where } \sigma_1 = \{x \mapsto a, y \mapsto b\} \]
\[ K\sigma_2 = b \leq a \implies f(b) \leq f(a) \quad \text{where } \sigma_2 = \{x \mapsto b, y \mapsto a\} \]
\[ K\sigma_3 = a \leq a \implies f(a) \leq f(a) \quad \text{where } \sigma_3 = \{x \mapsto a, y \mapsto a\} \]
\[ K\sigma_4 = b \leq b \implies f(b) \leq f(b) \quad \text{where } \sigma_4 = \{x \mapsto b, y \mapsto b\}. \]
EXAMPLE

\[ K = \forall x, y. x \leq y \implies f(x) \leq f(y) \]

\[ a + b \approx 1, \]
\[ f(a) + f(b) \approx 0, \]
\[ a + c \approx b + d, \]
\[ c \approx d, \]
\[ a \approx b. \]

\[ K_1 = a \leq b \implies f(a) \leq f(b) \text{ where } \sigma_1 = \{x \mapsto a, y \mapsto b\} \]
\[ K_2 = b \leq a \implies f(b) \leq f(a) \text{ where } \sigma_2 = \{x \mapsto b, y \mapsto a\} \]
\[ K_3 = a \leq a \implies f(a) \leq f(a) \text{ where } \sigma_3 = \{x \mapsto a, y \mapsto a\} \]
\[ K_4 = b \leq b \implies f(b) \leq f(b) \text{ where } \sigma_4 = \{x \mapsto b, y \mapsto b\}. \]
EXAMPLE

\[ K = \forall x, y. x \leq y \implies f(x) \leq f(y) \]

\[
\begin{align*}
a + b &\approx 1, \\
f(a) + f(b) &\approx 0, \\
a + c &\approx b + d, \\
c &\approx d, \\
a &\approx b.
\end{align*}
\]

\[
\begin{align*}
K\sigma_1 &= a \leq b \implies f(a) \leq f(b) \quad \text{where } \sigma_1 = \{x \mapsto a, y \mapsto b\} \\
K\sigma_2 &= b \leq a \implies f(b) \leq f(a) \quad \text{where } \sigma_2 = \{x \mapsto b, y \mapsto a\} \\
\text{\textbf{\underline{K}\sigma_3 = a \leq a \implies f(a) \leq f(a) \quad \text{where } \sigma_3 = \{x \mapsto a, y \mapsto a\}}} \\
K\sigma_4 &= b \leq b \implies f(b) \leq f(b) \quad \text{where } \sigma_4 = \{x \mapsto b, y \mapsto b\}.
\end{align*}
\]
EXAMPLE

\[ K = \forall x, y. x \leq y \implies f(x) \leq f(y) \]

\[ a + b \approx 1, \]
\[ f(a) + f(b) \approx 0, \]
\[ a + c \approx b + d, \]
\[ c \approx d, \]
\[ a \approx b. \]
EXAMPLE

\[ K = \forall x, y. \ x \leq y \implies f(x) \leq f(y) \]

\[ a + b \approx 1, \]
\[ f(a) + f(b) \approx 0, \]
\[ a + c \approx b + d, \]
\[ c \approx d, \]
\[ a \approx b. \]

Extension Theory Solver

E-matching
EXAMPLE

\[ K = \forall x, y. x \leq y \implies f(x) \leq f(y) \]

\[ a + b \approx 1, \]
\[ f(a) + f(b) \approx 0, \]
\[ a + c \approx b + d, \]
\[ c \approx d, \]
\[ a \approx b. \]

\[ G=\{a, b, c, d, a+c, b+d, 0, 1, f(a), f(b)\} \]
EXAMPLE

\[ K = \forall x, y. x \leq y \implies f(x) \leq f(y) \]

\( a + b \approx 1, \)
\( f(a) + f(b) \approx 0, \)
\( a + c \approx b + d, \)
\( c \approx d, \)
\( a \approx b. \)

\( G=\{a, b, c, d, a+c, b+d, 0, 1, f(a), f(b)\} \)
\( E=\{a+c\approx b+d, c\approx d, a\approx b\} \)
EXAMPLE

\[ K = \forall x, y. \, x \leq y \implies f(x) \leq f(y) \]

\[ a + b \approx 1, \]
\[ f(a) + f(b) \approx 0, \]
\[ a + c \approx b + d, \]
\[ c \approx d, \]
\[ a \approx b. \]

\[ G = \{ a, b, c, d, a+c, b+d, 0, 1, f(a), f(b) \} \]
\[ E = \{ a+c \approx b+d, c \approx d, a \approx b \} \]
\[ P = \{ f(x), f(y) \} \]
EXAMPLE

\[ K = \forall x, y. \ x \leq y \rightarrow f(x) \leq f(y) \]

\[ a + b \equiv 1, \]
\[ f(a) + f(b) \equiv 0, \]
\[ a + c \equiv b + d, \]
\[ c \equiv d, \]
\[ a \equiv b. \]

\[ G = \{a, b, c, d, a+c, b+d, 0, 1, f(a), f(b)\} \]
\[ E = \{a+c \equiv b+d, c \equiv d, a \equiv b\} \]
\[ P = \{f(x), f(y)\} \]

**Extension Theory Solver**

1. E-matching

\[ \{x \rightarrow a, y \rightarrow a\} \]
**ALGORITHM**

Input: $\phi$, $\mathcal{K}$, $Z$, $G$, $E$
Local variable: $Z' = \{\}$

1. For each $K$ in $\mathcal{K}$:
   1. Define patterns $P$ to be the function symbols in $K$ containing variables.
   2. Run E-matching algorithm with input $(E, G, P)$. Obtain substitutions $S$.
   3. For each $\sigma \in S$, if there exists no $K\sigma'$ in $Z$ such that $\sigma \sim_E \sigma'$, then add $K\sigma$ to $Z'$.

2. If $Z'$ is empty, return sat, else return $Z'$. 
**ALGORITHM**

Input: $\phi$, $\mathcal{K}$, $Z$, $G$, $E$

Local variable: $Z' = \{\}$

1. For each $K$ in $\mathcal{K}$:
   1. Define patterns $P$ to be the function symbols in $K$ containing variables.
   2. Run **E-matching** algorithm with input $(E, G, P)$. Obtain substitutions $S$.
   3. For each $\sigma \in S$, if there exists no $K\sigma'$ in $Z$ such that $\sigma \sim_E \sigma'$, then add $K\sigma$ to $Z'$.

2. If $Z'$ is empty, return sat, else return $Z'$.  

*Handled by incremental E-matching procedures, which are well-studied, already implemented in SMT Solvers*
ALGORITHM

- Minimal work while using existing solvers to get complete decision procedure.

- Solver improvements if told axioms encode local theory extension
  - Complete, stop search early when SAT
  - Further optimizations (see Section 6 in paper)

- Can be extended to Psi-local extensions (see Section 5 in paper)
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.
EXPERIMENT 1
EXPERIMENT 1

# instantiations by CVC4, baseline (C UL)

# eager instantiation

# instantiations by CVC4, baseline (C ULO)

# eager instantiation
## EXPERIMENT 2

<table>
<thead>
<tr>
<th>family</th>
<th>#</th>
<th>C</th>
<th>UD</th>
</tr>
</thead>
<tbody>
<tr>
<td>sl lists</td>
<td>139</td>
<td>127</td>
<td>70</td>
</tr>
<tr>
<td>dl lists</td>
<td>70</td>
<td>66</td>
<td>1717</td>
</tr>
<tr>
<td>sl nested</td>
<td>63</td>
<td>63</td>
<td>1060</td>
</tr>
<tr>
<td>sls lists</td>
<td>208</td>
<td>181</td>
<td>6046</td>
</tr>
<tr>
<td>trees</td>
<td>243</td>
<td>229</td>
<td>2121</td>
</tr>
<tr>
<td>soundness</td>
<td>79</td>
<td>76</td>
<td>17</td>
</tr>
<tr>
<td>sat</td>
<td>14</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>total</td>
<td>816</td>
<td>742</td>
<td>11032</td>
</tr>
</tbody>
</table>

Table 1. Comparison of solvers on uninstantiated benchmarks (time in sec.)
## EXPERIMENT 2

We compared our algorithms against two state-of-the-art solvers: GRASShopper and CVC4. We instrumented GRASShopper to eagerly instantiate all axioms. Subfigure (a) compares upfront instantiations with a baseline implementation of our E-matching algorithm. Points along the $x$-axis required no instantiations in CVC4 to conclude unsat.

We have plotted the above charts up to $10^{10}$ instantiations. There were four outlying benchmarks where upfront instantiations had between $10^{10}$ and up to $10^{14}$ 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 (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).
- **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 significantly.

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

<table>
<thead>
<tr>
<th>family</th>
<th>#</th>
<th>C</th>
<th>UD</th>
<th></th>
<th>C</th>
<th>UL</th>
</tr>
</thead>
<tbody>
<tr>
<td>sl lists</td>
<td>139</td>
<td>127</td>
<td>70</td>
<td></td>
<td>139</td>
<td>383</td>
</tr>
<tr>
<td>dl lists</td>
<td>70</td>
<td>66</td>
<td>1717</td>
<td></td>
<td>70</td>
<td>843</td>
</tr>
<tr>
<td>sl nested</td>
<td>63</td>
<td>63</td>
<td>1060</td>
<td></td>
<td>63</td>
<td>307</td>
</tr>
<tr>
<td>sls lists</td>
<td>208</td>
<td>181</td>
<td>6046</td>
<td></td>
<td>204</td>
<td>11230</td>
</tr>
<tr>
<td>trees</td>
<td>243</td>
<td>229</td>
<td>2121</td>
<td></td>
<td>228</td>
<td>22042</td>
</tr>
<tr>
<td>soundness</td>
<td>79</td>
<td>76</td>
<td>17</td>
<td></td>
<td>79</td>
<td>1533</td>
</tr>
<tr>
<td>sat</td>
<td>14</td>
<td>-</td>
<td>-</td>
<td></td>
<td>14</td>
<td>670</td>
</tr>
<tr>
<td>total</td>
<td>816</td>
<td>742</td>
<td>11032</td>
<td></td>
<td>797</td>
<td>37009</td>
</tr>
</tbody>
</table>

### Table 1.

Comparison of solvers on uninstantiated benchmarks (time in sec.)
## Experiment 2

We ran 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).
- **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 significantly, often by several orders of magnitude.

### Table 1

<table>
<thead>
<tr>
<th>family</th>
<th>#</th>
<th>C</th>
<th>UD</th>
<th>#</th>
<th>time</th>
<th>C</th>
<th>UL</th>
<th>#</th>
<th>time</th>
<th>C</th>
<th>ULO</th>
<th>#</th>
<th>time</th>
</tr>
</thead>
<tbody>
<tr>
<td>sl lists</td>
<td>139</td>
<td>127</td>
<td>70</td>
<td>139</td>
<td>383</td>
<td>139</td>
<td>17</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>dl lists</td>
<td>70</td>
<td>66</td>
<td>1717</td>
<td>70</td>
<td>843</td>
<td>70</td>
<td>33</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>sl nested</td>
<td>63</td>
<td>63</td>
<td>1060</td>
<td>63</td>
<td>307</td>
<td>63</td>
<td>13</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>sls lists</td>
<td>208</td>
<td>181</td>
<td>6046</td>
<td>204</td>
<td>11230</td>
<td>208</td>
<td>3401</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>trees</td>
<td>243</td>
<td>229</td>
<td>2121</td>
<td>228</td>
<td>22042</td>
<td>239</td>
<td>7187</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>soundness</td>
<td>79</td>
<td>76</td>
<td>17</td>
<td>79</td>
<td>1533</td>
<td>79</td>
<td>70</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>sat</td>
<td>14</td>
<td></td>
<td></td>
<td>14</td>
<td>670</td>
<td>14</td>
<td>12</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>total</td>
<td>816</td>
<td>742</td>
<td>11032</td>
<td>797</td>
<td>37009</td>
<td>812</td>
<td>10732</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
EXPERIMENT 2

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.

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 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.

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

<table>
<thead>
<tr>
<th>family</th>
<th>#</th>
<th>C</th>
<th>UD</th>
<th>#</th>
<th>time</th>
<th>C</th>
<th>UL</th>
<th>#</th>
<th>time</th>
<th>C</th>
<th>ULO</th>
<th>#</th>
<th>time</th>
<th>Z3</th>
<th>UD</th>
<th>#</th>
<th>time</th>
<th>Z3</th>
<th>UL</th>
<th>#</th>
<th>time</th>
<th>Z3</th>
<th>ULO</th>
<th>#</th>
<th>time</th>
</tr>
</thead>
<tbody>
<tr>
<td>sl lists</td>
<td>139</td>
<td>127</td>
<td>70</td>
<td>139</td>
<td>383</td>
<td>139</td>
<td>17</td>
<td>138</td>
<td>1955</td>
<td>138</td>
<td>1950</td>
<td>139</td>
<td>68</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>dl lists</td>
<td>70</td>
<td>66</td>
<td>1717</td>
<td>70</td>
<td>843</td>
<td>70</td>
<td>33</td>
<td>56</td>
<td>11375</td>
<td>56</td>
<td>11358</td>
<td>70</td>
<td>2555</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>sl nested</td>
<td>63</td>
<td>63</td>
<td>1060</td>
<td>63</td>
<td>307</td>
<td>63</td>
<td>13</td>
<td>52</td>
<td>6999</td>
<td>52</td>
<td>6982</td>
<td>59</td>
<td>1992</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>sls lists</td>
<td>208</td>
<td>181</td>
<td>6046</td>
<td>204</td>
<td>11230</td>
<td>208</td>
<td>3401</td>
<td>182</td>
<td>20596</td>
<td>182</td>
<td>20354</td>
<td>207</td>
<td>4486</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>trees</td>
<td>243</td>
<td>229</td>
<td>2121</td>
<td>228</td>
<td>22042</td>
<td>239</td>
<td>7187</td>
<td>183</td>
<td>41208</td>
<td>183</td>
<td>40619</td>
<td>236</td>
<td>27095</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>soundness</td>
<td>79</td>
<td>76</td>
<td>17</td>
<td>79</td>
<td>1533</td>
<td>79</td>
<td>70</td>
<td>76</td>
<td>7996</td>
<td>76</td>
<td>8000</td>
<td>79</td>
<td>336</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>sat</td>
<td>14</td>
<td>-</td>
<td>-</td>
<td>14</td>
<td>670</td>
<td>14</td>
<td>12</td>
<td>-</td>
<td>-</td>
<td>10</td>
<td>3964</td>
<td>14</td>
<td>898</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>total</td>
<td>816</td>
<td>742</td>
<td>11032</td>
<td>797</td>
<td>37009</td>
<td>812</td>
<td>10732</td>
<td>687</td>
<td>90130</td>
<td>697</td>
<td>93228</td>
<td>804</td>
<td>37430</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## EXPERIMENT 3

### Comparison of solvers on partially instantiated benchmarks

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

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 self 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.

In its simplest form, the algorithm does not require any modifications to the solver it is applied to and the only work it must do is to apply FLUSH to the solver it is applied to. This approach can only be expected to help where there are EPR-like axioms in the benchmarks, and we did have some which were heavier on these. We found that on singly linked list and tree benchmarks this helped compared to the purely E-matching algorithm. On the other hand, on nested list benchmarks, which make more heavy use of purely equational axioms, this technique does not help compared to the purely E-matching algorithm.

We have experimented with different configurations of two SMT solvers, implementing a number of optimizations of our base algorithm. In our evaluation we have experimented with different configurations of two SMT solvers, implementing a number of optimizations of our base algorithm. Our results suggest interesting directions to further improve the performance of SMT solvers, and at the same time make them complete on more expressive decidable fragments.

### Acknowledgments

We would like to thank the anonymous reviewers for their insightful comments and suggestions.
BIBLIOGRAPHY

- Ge, Y., de Moura, L.: Complete instantiation for quantified formulas in satisfiability modulo theories. CAV 2009.
CONCLUSION

- Algorithm for deciding local theory extensions using E-matching

- Uses existing SMT solvers: simple syntactic modifications to input
  For users: http://cs.nyu.edu/~kshitij/localtheories/

- Explored additional optimizations for SMT solvers

- Future directions: combining with model-based instantiation techniques.