Validity checking with if-then-else expressions modulo rich first-order theories

Silvio Ranise and David Déharbe∗•

LORIA/CASSIS & INRIA-Lorraine
Nancy, France
{deharbe, ranise}@loria.fr
http://www.loria.fr/~ranise
http://www.loria.fr/~deharbe

∗: UFRN/DIMAp – Natal, Brazil

• Partially supported by CAPES grant BEX0006/02-5.

March 05, 2003
### Background

- **Decision procedures** are embedded in many theorem provers to accelerate proofs (SVC, CVC, Simplify, Acl2, PVS, HOL, Coq, Acl2, RDL, ...)

- **Problem**: decision procedures are designed and/or incorporated by *ad-hoc* means (most of the times)

  - Design $\triangleright$ difficulties in proving correctness
  - Incorporation $\triangleright$ difficulties in combining with other decision procedures\(^1\) and reasoning activities\(^2\)

---

\(^1\)Shostak combination schema *docet!*

\(^2\)such as handling the boolean structure and instantiating quantifiers

---

Validity checking with if-then-else expressions modulo rich first-order theories – JSIVQP’03
What

- Eliminating trivial subgoals represented as sequents modulo a background theory $T$

- In many cases, this can be reduced to satisfiability checking, i.e. proving the unsatisfiability of conjunctions of ground literals modulo $T$, by standard techniques

Example $\forall$ $E$ is the quantifier-free theory of equality.

$E \models f^5(c) = c, f^3(c) = c \Rightarrow f(c) = c$

can be reduced to checking the $E$-unsatisfiability of

$f^5(c) = c \land f^3(c) = c \land f(c) \neq c$

--- Validity checking with if-then-else expressions modulo rich first-order theories – JSIVQP’03
### Who & How

<table>
<thead>
<tr>
<th>Approach</th>
<th>Propositional</th>
<th>First-Order</th>
<th>Efficiency</th>
<th>Flexibility</th>
</tr>
</thead>
<tbody>
<tr>
<td>naive</td>
<td>DNF</td>
<td>anything</td>
<td>hopeless</td>
<td>++</td>
</tr>
<tr>
<td>Simplify</td>
<td>DNF + splitting</td>
<td>NO. coop. DPs</td>
<td>+/-</td>
<td>++</td>
</tr>
<tr>
<td>SVC</td>
<td>ITE + splitting</td>
<td>Sh. coop. DPs</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>CVC, ICS</td>
<td>SAT + lemmas</td>
<td>Sh. coop. DPs</td>
<td>++</td>
<td>+</td>
</tr>
<tr>
<td>UCLID, EVC, sep</td>
<td>SAT</td>
<td>reduction</td>
<td>++</td>
<td>– –</td>
</tr>
</tbody>
</table>

Our goal: maintain efficiency while augmenting flexibility

<table>
<thead>
<tr>
<th>Approach</th>
<th>Propositional</th>
<th>First-Order</th>
<th>Efficiency</th>
<th>Flexibility</th>
</tr>
</thead>
<tbody>
<tr>
<td>haRVey</td>
<td>BDD + pruning</td>
<td>Superposition</td>
<td>++</td>
<td>+++</td>
</tr>
</tbody>
</table>

Validity checking with if-then-else expressions modulo rich first-order theories – JSIVQP’03

<< 3 >>
Difficulties to overcome

• Sometimes, checking the satisfiability of ground literals is not enough
  ▶ the capability of finding suitable instantiations of existentially quantified variables is required

• Handling user-defined symbols commonly used to structure specifications
  ▶ no satisfactory solution but mixture of manual unfolding and validity checking

• \textit{Simplify} features a heuristic matching mechanism to instantiate variables but... too heuristic $\Rightarrow$ lack of “precision” (false negatives)
Plan of the talk

- Satisfiability checking of conjunctions of ground literals (superposition)
- Extension to arbitrary boolean combinations of ground literals (BDDs)
- Pruning (efficiency)
- Extension to non-ground formulae (flexibility)
- Applications
- On-going and future work

-- Validity checking with if-then-else expressions modulo rich first-order theories -- JSIVQP’03
Plan of the Talk

- Satisfiability checking of conjunctions of ground literals
- Extension to arbitrary boolean combination of ground literals
- Pruning
- Extension to non-ground formulae
- Applications
- On-going and future work
A Rewriting Approach to Satisfiability Procedures
[Armando, Ranise & Rusinowitch, IC’03]

• **Idea**: pick a superposition calculus off-the-shelf and show that it can be used as a satisfiability procedure for the theory\(^3\) of interest

• **Generality**: the approach works for a wide class of theories, e.g.
  
  ▶ quantifier-free theory of equality
  
  ▶ (variants of) theory of lists
  
  ▶ theory of arrays and sets with/without extensionality
  
  ▶ combination of arrays and lists

\(^3\) whose satisfiability problem is decidable and which is presented by a finite set of clauses
A Rewriting Approach to Satisfiability Procedures—I

• Given

  ▶ a set \( S \) of ground literals

  ▶ a clausal presentation \( Ax(T) \) of a theory \( T \) extending \( E^4 \)

• We want to derive

  ▶ a procedure capable of establishing whether \( S \) is \( T \)-satisfiable\(^5\)

---

\(^4\)\( Ax(T) \) is not restricted to equations!
\(^5\)i.e. \( S \cup T \) is satisfiable
A Rewriting Approach to Satisfiability Procedures—II

• The recipe
  ▶ Flatten the input set \( S \) of ground literals to obtain \( S' \)
  ▶ Use a refutation complete superposition inference system and prove termination for \( S' \cup Ax(T) \), for all set of ground flat literals \( S' \)

• Features
  ▶ uniform\(^6\) and simple\(^7\)
  ▶ easy to implement... take a state-of-the-art saturation theorem prover

\(^6\)it works for theories which are important in verification
\(^7\)based on the rewriting framework, which is well-understood

– Validity checking with if-then-else expressions modulo rich first-order theories – JSIVQP’03
A Rewriting Approach to Satisfiability Procedures—III

- **flattening** = add “fresh” constants to name subterms

Example: \( \{ f(c, c') = h(h(a)), h(h(h(a))) \neq a \} \) is flattened to

\[ \{ f(c, c') = c_2, c_3 \neq a \} \cup \{ c_1 = h(a), c_3 = h(c_2), c_2 = h(c_1) \} \]

**Fact 1.** Let \( S \) be a finite set of \( \Sigma \)-literals. Then there exists a finite set of flat \( \Sigma' \)-literals \( S' \) s.t. \( S' \) is \( T \)-satisfiable iff \( S \) is.\(^8\)

- **Termination proofs** in the second step are substantially easier\(^9\)

\(^8\)\( \Sigma' \) is obtained from \( \Sigma \) by adding a finite number of constants

\(^9\)easier to reason about superpositions when depth of terms \( \leq 1 \)
# A Rewriting Approach to Satisfiability Procedures—IV

## The Superposition Calculus: Inference Rules

<table>
<thead>
<tr>
<th>Name</th>
<th>Rule</th>
<th>Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Sup.</strong></td>
<td>$\Gamma \rightarrow \Delta, l[u'] = r \quad \Pi \rightarrow \Sigma, u = v$</td>
<td>$u \not\triangleleft v, l[u'] \not\triangleleft r, \ast$</td>
</tr>
<tr>
<td></td>
<td>$\Gamma, \Pi \rightarrow \Delta, \Sigma, l[v] = r$</td>
<td></td>
</tr>
<tr>
<td><strong>Par.</strong></td>
<td>$\Gamma, l[u'] = r \rightarrow \Delta \quad \Pi \rightarrow \Sigma, u = v$</td>
<td>$u \not\triangleleft v, l[u'] \not\triangleleft r, \ast$</td>
</tr>
<tr>
<td></td>
<td>$l[v] = r, \Gamma, \Pi \rightarrow \Delta, \Sigma$</td>
<td></td>
</tr>
<tr>
<td><strong>Ref.</strong></td>
<td>$\Gamma, u' = u \rightarrow \Delta$</td>
<td>$(u' = u) \not\triangleleft (\Gamma \cup \Delta)$</td>
</tr>
<tr>
<td></td>
<td>$\Gamma \rightarrow \Delta$</td>
<td></td>
</tr>
<tr>
<td><strong>Fac.</strong></td>
<td>$\Gamma \rightarrow \Delta, u = t, u' = t'$</td>
<td>$u \not\triangleleft t, u \not\triangleleft \Gamma, (u = t) \not\triangleleft$</td>
</tr>
<tr>
<td></td>
<td>$\Gamma, t = t' \rightarrow \Delta, u' = t'$</td>
<td>${u' = t'} \cup \Delta$</td>
</tr>
</tbody>
</table>

$\ast (u = v) \not\triangleleft (\Pi \cup \Sigma), (l[u'] = r) \not\triangleleft (\Gamma \cup \Delta)$

$** \sigma = mgu(u, u')$ implicitly applied to consequents and conditions
### A Rewriting Approach to Satisfiability Procedures—V
#### The Superposition Calculus: Simplification Rules

<table>
<thead>
<tr>
<th>Name</th>
<th>Rule</th>
<th>Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Subsumption</strong></td>
<td>$S \cup {C, C'}$ \rightarrow $S \cup {C}$</td>
<td>for some $\theta$, $\theta(C') \subseteq C'$, and for no $\rho$, $\rho(C') = C$</td>
</tr>
<tr>
<td><strong>Simplification</strong></td>
<td>$S \cup {C[\theta(l)], l = r}$ \rightarrow $S \cup {C[\theta(r)], l = r}$</td>
<td>$\theta(l) \succ \theta(r)$, $C[\theta(l)] \succ ($(\theta(l) = \theta(r))$)</td>
</tr>
<tr>
<td><strong>Deletion</strong></td>
<td>$S \cup {\Gamma \rightarrow \Delta, t = t}$ \rightarrow $S$</td>
<td></td>
</tr>
</tbody>
</table>
A Rewriting Approach to Satisfiability Procedures—VI
The Superposition Calculus: Details

• Reduction Ordering $\succ$ total on ground terms and s.t. $f(c_1, \ldots, c_n) \succ c_0$, for each $n$-ary function symbol $f$ and constants $c_1, \ldots, c_n$ ($n \geq 1$)

$$\triangledown (a = b) \succ_e (c = d) \iff \{a, b\} \succ \{c, d\}^{10},$$

clauses (i.e. multisets of literals) are compared by the multiset extension of $\succ_e$

• Refutation complete = any fair application of the rules to an unsatisfiable set of clauses will derive the empty clause

• Saturation of a set of clauses is the final set of clauses generated by a fair derivation using the rules with higher priority given to the simplification rules

---

10 $\succ$ is the multiset extension of $\succ$ on terms

---

Validity checking with if-then-else expressions modulo rich first-order theories – JSIVQP’03
A Satisfiability Procedure for the theory $\mathcal{A}$ of Arrays

- Finite presentation

$Ax(\mathcal{A}) := \left\{ \begin{array}{l}
\text{read}(\text{write}(A, I, E), I) = E \\
I \neq J \Rightarrow \text{read}(\text{write}(A, I, E), J) = \text{read}(A, J)
\end{array} \right\}$

- Intuition

\[
\begin{array}{c}
\text{read}(i_1 : e_1, i_2) = e_2 \\
\text{write}(i_2 : e_2, i_3 : e_3, i_2, e'_2) =
\end{array}
\begin{array}{c}
\text{write}(i_1 : e_1) \\
\text{write}(i_2 : e'_2, i_3 : e_3, i_4 : e_4)
\end{array}
\]
Lemma 1. Let $S$ be a finite set of flat literals. The clauses occurring in the saturations of $S \cup Ax(A)$ by $SP$ can only be:

i) the empty clause; ii) axioms iii) ground flat literals

iv) clauses of type $t \bigtriangledown t' \lor c_1 = c'_1 \lor \cdots \lor c_n = c'_n$
with $t \bigtriangledown t' \in \{ c \neq c', \text{read}(c, i) = c', \text{read}(c, i) = \text{read}(c', i') \}$

v) clauses of type $\text{read}(c, x) = \text{read}(c', x) \lor c_1 = k_1 \lor \cdots \lor c_n = k_n$, where $k_i$ is either $x$ or $c_i$.  

Proof $\Rightarrow$ simple induction on the length of derivations $\Rightarrow$ saturations of $S' \cup Ax(A)$ are finite

Theorem 1. $SP$ is a satisfiability procedure for $A$.

Remark $\Rightarrow$ coarse grained complexity result $O(2^{n^2})$

---

$^{11}i, c, c', c_1, c'_1, \ldots, c_n, c'_n$ are constants, and $x$ is a variable

$^{12}$the satisfiability problem for $A$ is “only” $O(2^n)$ [Downey & Sethi, JACM 25(4)]
Plan of the Talk

- Satisfiability checking of conjunctions of ground literals
- Extension to arbitrary boolean combination of ground literals
- Pruning
- Extension to non-ground formulae
- Applications
- On-going and future work

– Validity checking with if-then-else expressions modulo rich first-order theories – JSIVQP’03
Arbitrary combination of ground literals—Ⅰ

Starting with a naive approach:

- to show that a ground formula $\varphi$ is $\mathcal{T}$-valid:
  - put $\neg\varphi$ into DNF: $\{S_i \mid i = 1, ..., n\}$
  - if there exists $j \in \{1, ..., n\}$ s.t. $S_j$ is $\mathcal{T}$-satisfiable, then $\varphi$ is not $\mathcal{T}$-valid and $S_j$ is a “counter-example”
  - if for all $i = 1, ..., n$ $S_i$ is $\mathcal{T}$-unsatisfiable, then $\varphi$ is $\mathcal{T}$-valid

- Ideas: (1) abstract atoms to propositional vars,
  (2) use BDDs to represent DNFs,
  (3) check $\mathcal{T}$-satisfiability by saturation
Reduced Ordered Binary Decision Diagrams (BDDs) provide an efficient data structure for propositional reasoning.

The BDD of a formula $f(x_1, x_2 \ldots x_n)$ can be viewed as Shannon’s expansion:

$$[x_1 \land f(\top, x_2 \ldots x_n)] \lor [\neg x_1 \land f(\bot, x_2 \ldots x_n)].$$

$\begin{array}{c}
\text{\textbullet} \\
\text{\textbullet} \\
\text{\textbullet}
\end{array}$

Validity checking with if-then-else expressions modulo rich first-order theories – JSIVQP’03
Arbitrary combination of ground literals—III

- **Reduced**: no distinct isomorphic subtrees
- **Ordered**: variables appear on a given order from root to leaves
- **BDDs are canonical** (for propositional logic):
  - a formula is *valid* iff its BDD is the same as the BDD for $\top$
  - a formula is *satisfiable* iff it is different from the BDD for $\bot$: each branch from a root to the $\top$ leaf is a satisfiable valuation
  - no search!

– Validity checking with if-then-else expressions modulo rich first-order theories – JSIVQP’03
Arbitrary combination of ground literals—IV

- BDDs are no more canonical for first-order logic

Example

\[
\text{ite}(x = y, \\
\text{ite}(p(x), \\
\text{ite}(p(y), \bot, \top), \\
\top) \\
p(x))
\]

Notice: the \(\top\)-branch \(x = y \land p(x) \land \neg p(y)\) is not \(E\)-satisfiable

- Search for \(T\)-satisfiable branches (\(\equiv\) conjunctions of ground literals)
  - check branches for \(T\)-(un)satisfiability by superposition
Arbitrary combination of ground literals—V

function search_satisfiable (T: theory; φ: formula)
while φ ≠ ⊥ do
  β ← branch(φ)
  ρ ← saturate(T, β)
  if ∅ ∉ ρ then return (yes, β)
  φ ← φ ∧ ¬β
done
return (no, −)
end

• Unfortunately, inefficient because of exhaustive search (exponentially many branches)
Plan of the talk

- Satisfiability checking of conjunctions of ground literals
- Extension to arbitrary boolean combination of ground literals
- Pruning
- Extension to non-ground formulae
- Applications
- On-going and future work

- Validity checking with if-then-else expressions modulo rich first-order theories – JSIVQP’03
Pruning—I

• Avoid repeated $T$-satisfiability checking of “similar” branches

  **Intuition** ▷ many branches in the BDD with the same prefix containing the literals which are “responsible” for the $T$-unsatisfiability

• $S_1$ and $S_2$ are “similar” if there exists $S$ s.t. $S \subset S_1$, $S \subset S_2$, and the $T$-unsatisfiability of $S$ entails the $T$-unsatisfiability of $S_1$ and $S_2$

  ▷ the problem of determining the smallest set $S$ is NP-complete

  ▷ in practice: $S$ is (heuristically) “small”

• modify *saturate* to return also the proof of the empty clause

  ▷ the proof will contain $S$, (usually) a subset of the branch $\beta$
Pruning—II

Pruning subsumed branches

• Let $\pi$ be the saturation proof showing that the branch $\beta$ is $T$-unsatisfiable;

• Let $\text{hypothesis}(\pi)$ be the set of all the literals in $\pi$ belonging to the branch $\beta$;

• Then $\text{hypothesis}(\pi)$ is $T$-unsatisfiable, and $\neg\text{hypothesis}(\pi)$ is $T$-satisfiable.
Pruning—III

function search_satisfiable (T: theory; φ: formula)
  while φ ≠ ⊥ do
    β ← branch(φ)
    (ρ, π) ← saturate(T, β)
    if [] ∉ ρ then return (yes, β)
    φ ← φ ∧ ¬hypothesis(π)
  done
  return (no, −)
end
These techniques have been implemented in haRVey:

- it uses:
  - CWI’s ATerm library
  - Schulz E-prover
  - David Long’s BDD library

- available on the Web at
  [http://www.loria.fr/equipes/cassis/softwares/haRVey](http://www.loria.fr/equipes/cassis/softwares/haRVey)
Input: \( \langle Ax(T), \varphi \rangle \), where

- \( Ax(T) \) axiomatises the background theory
- \( \varphi \) is the formula to be proved valid.

Output:

- Yes or
- No, and a “counter-example”
Experimental results

- We compare the execution time of haRVey with and without pruning of subsumed branches.

- The benchmark we use are the verification condition (valid purely equational ground formulae) for the proof of Burns’ mutual exclusion protocol\(^\text{13}\)

<table>
<thead>
<tr>
<th>Proof obligation</th>
<th>elimination</th>
<th>Proof obligation</th>
<th>elimination</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>off</td>
<td>on</td>
<td></td>
</tr>
<tr>
<td>Burns 6</td>
<td>0.90</td>
<td>0.17</td>
<td>Burns 8</td>
</tr>
<tr>
<td>Burns 12</td>
<td>0.73</td>
<td>0.26</td>
<td>Burns 14</td>
</tr>
<tr>
<td>Burns 23</td>
<td>10.53</td>
<td>0.09</td>
<td>Burns 24</td>
</tr>
<tr>
<td>Burns 28</td>
<td>14.94</td>
<td>0.24</td>
<td>Burns 31</td>
</tr>
<tr>
<td>Burns 33</td>
<td>8.97</td>
<td>0.25</td>
<td>Burns 34</td>
</tr>
<tr>
<td>Burns 36</td>
<td>13.73</td>
<td>0.84</td>
<td>Burns 37</td>
</tr>
</tbody>
</table>

\(^{13}\)Warning: The natural number \(N\) in Burns\(N\) uniquely identifies a proof obligation. It is by no means related to the size of the proof obligations.

average speed-up: 2 orders of magnitude
Plan of the talk

- Satisfiability checking of conjunctions of ground literals
- Extension to arbitrary boolean combination of ground literals
- Pruning
- Extension to non-ground formulae
- Applications
- On-going and future work
Instantiating Quantifiers–I

- **Problem:** sometimes checking the satisfiability of ground literals is not enough

  - the capability of finding suitable instantiations of existentially quantified variables is required

  for example checking the validity of formulae of the following form is required:

  $$\left(\forall x_1, \ldots, x_n \phi\right) \implies \left(\forall y_1, \ldots, y_m \psi\right)$$

  implicit existential quantif.

where the only free var’s of $\phi$ ($\psi$) are $x_1, \ldots, x_n$ ($y_1, \ldots, y_m$, resp.)
Instantiating Quantifiers–II

• Solution: use basic properties of the consequence relation \( \models \) to pre-process the proof obligation and then invoke haRVey with an extended background theory

\[
\mathcal{T} \models (\forall x_1, \ldots, x_n \phi) \implies (\forall y_1, \ldots, y_m \psi)
\]
can be transformed to

\[
\mathcal{T}, \forall x_1, \ldots, x_n \phi \models \forall y_1, \ldots, y_m \psi
\]
extended background theory

similarly for proof obligations of the form

\[
(\exists x_1, \ldots, x_n \phi) \implies (\exists y_1, \ldots, y_m \psi)
\]
since they can be transformed into

\[
(\forall x_1, \ldots, x_n \neg \psi) \implies (\forall y_1, \ldots, y_m \neg \phi)
\]
Notice that we are hiding the difficulties in the background reasoner.

Fortunately, haRVey adopts a full-fledge first-order equational prover as background reasoner.

So, in haRVey: integrating

$T$-satisfiability checking of ground literals

and

quantifier reasoning for $\forall x_1, \ldots, x_n \phi$

Remark: fully automatic semi-decision procedure, no user-guidance!
Instantiating Quantifiers–IV

• A general approach

  ▶ replace each quantified subformula \( Qx_1 \ldots x_n \phi(x_1 \ldots x_n) \) with a fresh propositional constant \( p \);

  ▶ add the clausification of formula \( p \iff Qx_1 \ldots x_n \phi(x_1 \ldots x_n) \) to the background theory

  refinement ▶ take into account polarity, e.g. \( T \models \forall x.A(x) \Rightarrow G \)

  to check the \( T \)-unsatisfiability of \( \forall x.A(x) \land \neg G \), it is sufficient

  to check the \( T \cup \{ p \Rightarrow \forall x.A(x) \} \)-unsatisfiability of \( p \land \neg G \)
Instantiating Quantifiers–V

Example

\[ T \models (a = a' \land \exists x.a = ins(x, \emptyset)) \implies \exists x.a' = ins(x, \emptyset) \]

quantified subformulae are replaced with constants:

\[ T \cup \{p \leftrightarrow \exists x.a = ins(x, \emptyset), p' \leftrightarrow \exists x.a' = ins(x, \emptyset)\} \models (a = a' \land p) \implies p' \]

axioms are clausified:

\[ T \cup \{\neg p \lor a = ins(sk, \emptyset), p \lor a = ins(X, \emptyset), \ldots\} \models (a = a' \land p) \implies p' \]
Plan of the talk

- Satisfiability checking of conjunctions of ground literals
- Extension to arbitrary boolean combination of ground literals
- Pruning
- Extension to non-ground formulae
- Applications
- On-going and future work

– Validity checking with if-then-else expressions modulo rich first-order theories – JSIVQP’03
Applications—I

- Parameterized protocols (Burns) ✓
- Debugging of pointer-manipulating programs
- Correctness of (imperative) programs
- Push-button proofs of invariants of B machines (work in progress)
Applications—II

Debugging of pointer-manipulating programs (written in C)

• Loops are unrolled a fixed number of times (sufficient to expose bugs)
• Theory of arrays to model pointer (de-)referencing
• Theory of lists to reason about records
• This approach allows for (declarative)
  ▶ cleanness checking of pointer-manipulating prg.\(^\text{14}\)
  ▶ verification of user specified annotations

(a) user-defined property

<table>
<thead>
<tr>
<th></th>
<th>haRVey</th>
<th>E</th>
<th>Simplify</th>
</tr>
</thead>
<tbody>
<tr>
<td>search\textsuperscript{1}</td>
<td>0.08</td>
<td>0.03</td>
<td>0.03</td>
</tr>
<tr>
<td>search\textsuperscript{2}</td>
<td>0.17</td>
<td>0.17</td>
<td>0.10</td>
</tr>
<tr>
<td>search\textsuperscript{3}</td>
<td>0.68</td>
<td>0.16</td>
<td>0.71</td>
</tr>
<tr>
<td>search\textsuperscript{4}</td>
<td>0.16</td>
<td>0.36</td>
<td>1.78</td>
</tr>
<tr>
<td>search\textsuperscript{5}</td>
<td>0.17</td>
<td>1.22</td>
<td>21.36</td>
</tr>
<tr>
<td>search\textsuperscript{6}</td>
<td>0.23</td>
<td>5.48</td>
<td>mem.out</td>
</tr>
<tr>
<td>remove\textsuperscript{1}</td>
<td>0.15</td>
<td>0.02</td>
<td>0.02</td>
</tr>
<tr>
<td>remove\textsuperscript{2}</td>
<td>0.19</td>
<td>0.39</td>
<td>0.14</td>
</tr>
<tr>
<td>remove\textsuperscript{3}</td>
<td>0.21</td>
<td>11.40</td>
<td>2.18</td>
</tr>
<tr>
<td>remove\textsuperscript{4}</td>
<td>0.37</td>
<td>40.55</td>
<td>mem.out</td>
</tr>
</tbody>
</table>

(b) read undefined

<table>
<thead>
<tr>
<th></th>
<th>haRVey</th>
<th>E</th>
<th>Simplify</th>
</tr>
</thead>
<tbody>
<tr>
<td>fumble\textsuperscript{3}</td>
<td>0.06</td>
<td>1.62</td>
<td>0.04</td>
</tr>
<tr>
<td>insert\textsuperscript{3}</td>
<td>0.34</td>
<td>111.89</td>
<td>0.29</td>
</tr>
<tr>
<td>merge\textsuperscript{2}</td>
<td>0.19</td>
<td>time out</td>
<td>3.61</td>
</tr>
<tr>
<td>remove\textsuperscript{3}</td>
<td>0.04</td>
<td>time out</td>
<td>3.46</td>
</tr>
<tr>
<td>remove-all\textsuperscript{3}</td>
<td>0.06</td>
<td>0.03</td>
<td>0.03</td>
</tr>
<tr>
<td>reverse\textsuperscript{3}</td>
<td>0.26</td>
<td>2.64</td>
<td>0.08</td>
</tr>
<tr>
<td>rotate\textsuperscript{3}</td>
<td>0.48</td>
<td>10.80</td>
<td>0.11</td>
</tr>
<tr>
<td>search\textsuperscript{3}</td>
<td>0.33</td>
<td>145.40</td>
<td>0.79</td>
</tr>
<tr>
<td>swap\textsuperscript{3}</td>
<td>0.21</td>
<td>0.10</td>
<td>0.04</td>
</tr>
</tbody>
</table>

▷ The numbers appearing as exponents to the names of the proof obligations indicate how many times the loop in the corresponding C program has been unrolled.
### Applications—IV

#### (c) null pointer dereferencing

<table>
<thead>
<tr>
<th>Operation</th>
<th>haRVey</th>
<th>E</th>
<th>Simplify</th>
</tr>
</thead>
<tbody>
<tr>
<td>insert</td>
<td>0.83</td>
<td>time out</td>
<td>3.22</td>
</tr>
<tr>
<td>merge</td>
<td>5.83</td>
<td>sp. out</td>
<td>41.71</td>
</tr>
<tr>
<td>remove</td>
<td>0.05</td>
<td>sp. out</td>
<td>9.69</td>
</tr>
<tr>
<td>remove-all</td>
<td>0.06</td>
<td>0.05</td>
<td>0.02</td>
</tr>
<tr>
<td>reverse</td>
<td>0.20</td>
<td>time out</td>
<td>0.51</td>
</tr>
<tr>
<td>rotate</td>
<td>0.39</td>
<td>time out</td>
<td>0.85</td>
</tr>
<tr>
<td>swap</td>
<td>0.25</td>
<td>0.12</td>
<td>0.04</td>
</tr>
</tbody>
</table>

#### (d) memory leakage

<table>
<thead>
<tr>
<th>Operation</th>
<th>haRVey</th>
<th>E</th>
<th>Simplify</th>
</tr>
</thead>
<tbody>
<tr>
<td>fumble</td>
<td>0.10</td>
<td>time out</td>
<td>38.08</td>
</tr>
<tr>
<td>insert</td>
<td>0.07</td>
<td>time out</td>
<td>25.93</td>
</tr>
<tr>
<td>merge</td>
<td>0.48</td>
<td>sp. out</td>
<td>mem. out</td>
</tr>
<tr>
<td>remove</td>
<td>0.11</td>
<td>sp. out</td>
<td>mem. out</td>
</tr>
<tr>
<td>remove-all</td>
<td>0.05</td>
<td>sp. out</td>
<td>0.65</td>
</tr>
<tr>
<td>reverse</td>
<td>1.20</td>
<td>sp. out</td>
<td>mem. out</td>
</tr>
<tr>
<td>rotate</td>
<td>1.68</td>
<td>sp. out</td>
<td>mem. out</td>
</tr>
<tr>
<td>search</td>
<td>0.05</td>
<td>time out</td>
<td>3.64</td>
</tr>
<tr>
<td>swap</td>
<td>0.05</td>
<td>time out</td>
<td>1.17</td>
</tr>
</tbody>
</table>

---

Validity checking with if-then-else expressions modulo rich first-order theories – JSIVQP’03
Applications—V

Correctness of (imperative) algorithms

• **challenge**: automatically check all the proof obligations of Union-Find algorithm given in Nelson’s POPL’81 paper

• Simplify (heuristic) handling of quantifier is **not capable of coping** with such a complex situation

  ▶ sophisticated pointer manipulations

  ▶ Nelson’s formalization of linked lists is different from the previous one ⇒ **flexibility** of haRVey w.r.t. changes in the background theory
Applications—VI

<table>
<thead>
<tr>
<th></th>
<th>B1</th>
<th>B2</th>
<th>B3</th>
<th>B4</th>
<th>B5</th>
<th>B6</th>
</tr>
</thead>
<tbody>
<tr>
<td>haRVey</td>
<td>0.03</td>
<td>0.06</td>
<td>0.03</td>
<td>0.03</td>
<td>0.06</td>
<td>0.14</td>
</tr>
<tr>
<td>E prover</td>
<td>0.03</td>
<td>0.03</td>
<td>0.10</td>
<td>23.16</td>
<td>0.03</td>
<td>0.04</td>
</tr>
<tr>
<td>Simplify</td>
<td>—</td>
<td>0.01</td>
<td>—</td>
<td>0.04</td>
<td>0.01</td>
<td>0.01</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>B7</th>
<th>B8</th>
<th>B9</th>
<th>B10</th>
<th>B11</th>
<th>B12</th>
</tr>
</thead>
<tbody>
<tr>
<td>haRVey</td>
<td>0.11</td>
<td>0.33</td>
<td>0.69</td>
<td>2.46</td>
<td>0.31</td>
<td>1.50</td>
</tr>
<tr>
<td>E prover</td>
<td>0.04</td>
<td>0.18</td>
<td>0.75</td>
<td>time out</td>
<td>0.13</td>
<td>0.65</td>
</tr>
<tr>
<td>Simplify</td>
<td>0.20</td>
<td>—</td>
<td>†</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
</tbody>
</table>

— indicates that Simplify gave a (false) negative answer.
† indicates that Simplify aborted.

Push-button proofs of invariants of B machines

- B machines are annotated with invariants;
- B machines are written with set theory operators;
- standard methodology to generate first-order proof obligations which entails that an invariant is an inductive invariant;
- B constructs require quantifiers
Applications—VIII

MACHINE scheduler
SETS PID
VARIABLES active, ready, waiting
INVARIANT active <: PID \ ready <: PID \ waiting <: PID \ card(active) <= 1 \ ready \ active = {} \ waiting \ ready = {} \ active \ waiting = {}

INITIALISATION
active, ready, waiting := {}, {}, {}

OPERATIONS
new(pp) = PRE pp: PID - (active \ ready \ waiting) THEN waiting := waiting \ {pp} END
swap = PRE active /= {} THEN
waiting := waiting \ active ||
IF (ready /= {})
    ANY pr WHERE pr: ready THEN active := {pr} || ready := ready - {pr} END
END
ready = PRE pw: waiting THEN
waiting := waiting \ {pw} ||
IF active /= {} THEN ready := ready \ {pw}
ELSE active := {pw} END
END
del(dd) = PRE dd: waiting THEN waiting := waiting - {dd} END
END

– Validity checking with if-then-else expressions modulo rich first-order theories – JSIVQP’03
Future Work & Advertising

• Push-button theorem prover for invariant verification of B machines

• Integration of arithmetic reasoning in haRVey
  ▶ (generalization of) Nelson and Oppen combination schema

• Benchmarks, benchmarks, benchmarks, benchmarks, ... 
  ▶ SMT-LIB Initiative: Satisfiability Modulo Theory Library
    http://combination.cs.uiowa.edu/smtlib

  ▶ PDPAR Workshop (affiliated to CADE): Practical Aspects of Decision Procedures in Automated Reasoning
    http://www.loria.fr/~ranise/pdpar03

> Validity checking with if-then-else expressions modulo rich first-order theories – JSIVQP’03
Take a look at
http://www.loria.fr/equipes/cassis/softwares/haRVey

● A rewriting approach to satisfiability procedures

▷ A. Armando, S. Ranise, M. Rusinowitch. **Uniform Derivation of Decision Procedures by Superposition.** In Annual Conf. on Computer Science Logic (CSL01), 2001.


● Combining theory and propositional reasoning