Integrating a SAT Solver with an LCF-style Theorem Prover

A Fast Decision Procedure for Propositional Logic for the Isabelle System

Tjark Weber
webertj@in.tum.de

PDPAR'05, July 12, 2005
Motivation

- Verification problems can often be reduced to Boolean satisfiability.
- Recent SAT solver advances have made this approach feasible in practice.

Can an **LCF-style** theorem prover benefit from these advances?
zChaff

- A leading SAT solver (winner of the SAT 2002 and SAT 2004 competitions in several categories)
- Developed by Sharad Malik and Zhaohui Fu, Princeton University
- Returns a satisfying assignment, or . . .
- . . . a proof of unsatisfiability (since 2003)
Preprocessing

Input: propositional formula $\phi$

- CNF conversion
- Normalization
- Removal of duplicate literals
- Removal of tautological clauses

Output: a theorem of the form $\phi = \phi^*$

```thm
thm_of decomp t =
let
  (ts, recomb) = decomb t
in recomb (map (thm_of decomp) ts)
```
The SAT Solver’s Trace

CL: 184 <= 173 28 35 142 154
CL: 185 <= 43 4 11 59 55
[...]  
VAR: 16 L: 35 V: 0 A: 55 Lits: 29 33  
VAR: 26 L: 28 V: 1 A: 202 Lits: 52 98 57  
[...]  
CONF: 206 == 80 82 64 70 37
The SAT Solver’s Trace

<table>
<thead>
<tr>
<th>clause id</th>
<th>resolvents</th>
</tr>
</thead>
<tbody>
<tr>
<td>CL: 184</td>
<td>&lt;= 173 28 35 142 154</td>
</tr>
<tr>
<td>CL: 185</td>
<td>&lt;= 43 4 11 59 55</td>
</tr>
<tr>
<td>[... ]</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>variable id</th>
<th>antecedent</th>
</tr>
</thead>
<tbody>
<tr>
<td>VAR: 16</td>
<td>L: 35</td>
</tr>
<tr>
<td></td>
<td>V: 0</td>
</tr>
<tr>
<td></td>
<td>A: 55</td>
</tr>
<tr>
<td></td>
<td>Lits: 29 33</td>
</tr>
<tr>
<td>VAR: 26</td>
<td>L: 28</td>
</tr>
<tr>
<td></td>
<td>V: 1</td>
</tr>
<tr>
<td></td>
<td>A: 202</td>
</tr>
<tr>
<td></td>
<td>Lits: 52 98 57</td>
</tr>
<tr>
<td>[... ]</td>
<td>variable id</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>conflict clause id</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONF: 206 == 80 82 64 70 37</td>
</tr>
</tbody>
</table>
Proof Reconstruction (1)

- `resolution : Thm.thm list -> Thm.thm`

- `prove_clause : int -> Thm.thm`

- `prove_literal : int -> Thm.thm`
Proof Reconstruction (1)

• resolution : Thm.thm list -> Thm.thm
  Input: \[ X \rightarrow P \lor Q \lor R, X \rightarrow S \lor \neg Q \lor T \]
  Result: \[ X \rightarrow P \lor R \lor S \lor T \]

• prove_clause : int -> Thm.thm

• prove_literal : int -> Thm.thm
Proof Reconstruction (1)

- \textbf{resolution} : \textit{Thm.thm list} \rightarrow \textit{Thm.thm}
  
  \textbf{Input:} \ [X \rightarrow Q, X \rightarrow \neg Q]
  
  \textbf{Result:} \ X \rightarrow \text{False}

- \textbf{prove\_clause} : \textit{int} \rightarrow \textit{Thm.thm}

- \textbf{prove\_literal} : \textit{int} \rightarrow \textit{Thm.thm}
Proof Reconstruction (1)

- **resolution** : Thm.thm list -> Thm.thm
  
  Input: $[X \to Q, X \to \neg Q]$
  
  Result: $X \to \text{False}$

- **prove_clause** : int -> Thm.thm
  
  $\text{prove_clause clause_id} =$
  
  resolution (map prove_clause
  (resolvents_of clause_id))

- **prove_literal** : int -> Thm.thm
Proof Reconstruction (1)

resolution : Thm.thm list -> Thm.thm
Input: [X \rightarrow Q, X \rightarrow \neg Q]
Result: X \rightarrow False

prove_clause : int -> Thm.thm
prove_clause clause_id =
    resolution (map prove_clause (resolvents_of clause_id))

prove_literal : int -> Thm.thm
prove_literal var_id =
    let th_ante = prove_clause (antecedent_of var_id)
    var_ids = filter (\i. i \neq var_id)
        (var_ids_in_clause th_ante)
    in resolution
        (th_ante :: map prove_literal var_ids)
Proof Reconstruction (2)

- Many clauses may be redundant.
- Clauses and literals may be needed many times.
Proof Reconstruction (2)

Many clauses may be redundant.

Clauses and literals may be needed many times.

Two *arrays* store . . .

- each clause’s resolvents *or* its proof,
- each variable’s antecedent *or* its proof

. . . and are *updated* during proof reconstruction.
Proof Reconstruction (2)

Many clauses may be redundant.

Clauses and literals may be needed many times.

Two *arrays* store . . .

- each clause’s resolvents *or* its proof,
- each variable’s antecedent *or* its proof

. . . and are *updated* during proof reconstruction.

1. Initialize arrays with information from the trace.
2. Prove conflict clause $C$.
3. Perform resolution with `prove_literal` for each literal in $C$. 
Evaluation

- Isabelle is several orders of magnitude slower than zverify_df.

- However, zChaff vs. auto/blast/fast . . .
  - 42 propositional problems in TPTP, v2.6.0
    - 19 “easy” problems, solved in less than a second each by auto, blast, fast, and zChaff
    - 23 harder problems
# Performance

<table>
<thead>
<tr>
<th>Problem</th>
<th>Status</th>
<th>auto</th>
<th>blast</th>
<th>fast</th>
<th>zChaff</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSC007-1.008</td>
<td>unsat.</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>726.5</td>
</tr>
<tr>
<td>NUM285-1</td>
<td>sat.</td>
<td>✗</td>
<td>✗</td>
<td></td>
<td>0.2</td>
</tr>
<tr>
<td>PUZ013-1</td>
<td>unsat.</td>
<td></td>
<td>0.5</td>
<td>✗</td>
<td>6.1</td>
</tr>
<tr>
<td>PUZ014-1</td>
<td>unsat.</td>
<td></td>
<td>1.4</td>
<td>✗</td>
<td>0.1</td>
</tr>
<tr>
<td>PUZ015-2.006</td>
<td>unsat.</td>
<td>✗</td>
<td>✗</td>
<td></td>
<td>10.5</td>
</tr>
<tr>
<td>PUZ016-2.004</td>
<td>sat.</td>
<td>✗</td>
<td>✗</td>
<td></td>
<td>0.3</td>
</tr>
<tr>
<td>PUZ016-2.005</td>
<td>unsat.</td>
<td>✗</td>
<td>✗</td>
<td></td>
<td>1.6</td>
</tr>
<tr>
<td>PUZ030-2</td>
<td>unsat.</td>
<td>✗</td>
<td>✗</td>
<td></td>
<td>0.7</td>
</tr>
<tr>
<td>PUZ033-1</td>
<td>unsat.</td>
<td></td>
<td>0.2</td>
<td>6.4</td>
<td>0.1</td>
</tr>
<tr>
<td>SYN001-1.005</td>
<td>unsat.</td>
<td>✗</td>
<td>✗</td>
<td></td>
<td>0.4</td>
</tr>
<tr>
<td>SYN003-1.006</td>
<td>unsat.</td>
<td></td>
<td>0.9</td>
<td>✗</td>
<td>1.6</td>
</tr>
<tr>
<td>SYN004-1.007</td>
<td>unsat.</td>
<td></td>
<td>0.3</td>
<td>822.2</td>
<td>2.8</td>
</tr>
<tr>
<td>SYN010-1.005.005</td>
<td>unsat.</td>
<td>✗</td>
<td>✗</td>
<td></td>
<td>0.4</td>
</tr>
<tr>
<td>SYN086-1.003</td>
<td>sat.</td>
<td>✗</td>
<td>✗</td>
<td></td>
<td>0.1</td>
</tr>
<tr>
<td>SYN087-1.003</td>
<td>sat.</td>
<td>✗</td>
<td>✗</td>
<td></td>
<td>0.1</td>
</tr>
<tr>
<td>SYN090-1.008</td>
<td>unsat.</td>
<td></td>
<td>13.8</td>
<td>✗</td>
<td>0.5</td>
</tr>
<tr>
<td>SYN091-1.003</td>
<td>sat.</td>
<td>✗</td>
<td>✗</td>
<td></td>
<td>0.1</td>
</tr>
<tr>
<td>SYN092-1.003</td>
<td>sat.</td>
<td>✗</td>
<td>✗</td>
<td></td>
<td>0.1</td>
</tr>
<tr>
<td>SYN093-1.002</td>
<td>unsat.</td>
<td></td>
<td>1290.8</td>
<td>16.2</td>
<td>1126.6</td>
</tr>
<tr>
<td>SYN094-1.005</td>
<td>unsat.</td>
<td>✗</td>
<td>✗</td>
<td></td>
<td>0.8</td>
</tr>
<tr>
<td>SYN097-1.002</td>
<td>unsat.</td>
<td>✗</td>
<td>19.2</td>
<td>✗</td>
<td>0.2</td>
</tr>
<tr>
<td>SYN098-1.002</td>
<td>unsat.</td>
<td>✗</td>
<td>✗</td>
<td></td>
<td>0.4</td>
</tr>
<tr>
<td>SYN302-1.003</td>
<td>sat.</td>
<td>✗</td>
<td>✗</td>
<td></td>
<td>0.4</td>
</tr>
</tbody>
</table>
Conclusions and Future Work

- A fast decision procedure for propositional logic
- Counterexamples for unprovable formulae
- Huge SAT problems are still out of scope
- Extension to (fragments of) richer logics
- Integration of first-order provers