SAT-based Finite Model Generation for Higher-Order Logic

Tjark Weber

October 9, 2008
Complex systems almost inevitably contain bugs.
Motivation

Complex systems almost inevitably contain bugs.

Complex **formalizations** almost inevitably contain bugs.

- Initial conjectures are frequently false.
- A counterexample often exhibits a fault in the implementation.
Questions

1. Can we use efficient SAT solvers to find counterexamples in higher-order logic automatically?
Questions

1. Can we use efficient SAT solvers to find counterexamples in higher-order logic automatically?

2. Can we use efficient SAT solvers to prove theorems in an LCF-style theorem prover?
Overview

Interactive Theorem Proving

- Ch. 4
- Ch. 5

Countermodels
- Ch. 2, 3

SAT Solvers
Isabelle/HOL: higher-order logic, based on Church’s simple theory of types (1940)

- **Types:** \( \sigma ::= \alpha | (\sigma_1, \ldots, \sigma_n) c \)
- **Terms:** \( t_\sigma ::= x_\sigma | c_\sigma | (t_{\sigma'} \rightarrow_{\sigma} t'_{\sigma'})_{\sigma} | (\lambda x_{\sigma_1} \cdot t_{\sigma_2})_{\sigma_1 \rightarrow \sigma_2} \)

Two special type constructors: \( \text{bool} \) and \( \rightarrow \)

Two logical constants: \( \rightarrow_{\text{bool} \rightarrow \text{bool} \rightarrow \text{bool}} \) and \( =_{\sigma \rightarrow \sigma \rightarrow \text{bool}} \)
The Semantics of HOL

Standard set-theoretic semantics:

- Types denote certain non-empty sets.
  - \([\text{bool}] = \{ \top, \bot \}\)
  - \([\sigma_1 \rightarrow \sigma_2] = [\sigma_2][\sigma_1]\)
- Terms denote elements of these sets.
The Semantics of HOL

Standard set-theoretic semantics:

- Types denote certain non-empty finite sets.
  - $\llbracket \text{bool} \rrbracket = \{ \top, \bot \}$
  - $\llbracket \sigma_1 \rightarrow \sigma_2 \rrbracket = \llbracket \sigma_2 \rrbracket^{\llbracket \sigma_1 \rrbracket}$
- Terms denote elements of these sets.
Translation to Propositional Logic

- Terms of base type: e.g., $x_\alpha$, with $[\alpha] = \{a_0, a_1, a_2, a_3, a_4\}$

$x = a_0$  $x = a_1$  $x = a_2$  $x = a_3$  $x = a_4$
Translation to Propositional Logic

- Terms of base type: e.g., $x_\alpha$, with $[[\alpha]] = \{a_0, a_1, a_2, a_3, a_4\}$

- Functions: e.g., $f_{\beta \rightarrow \alpha}$, with $[[\beta]] = \{b_0, b_1, b_2\}$
Translation to Propositional Logic

- Terms of base type: e.g., $x_\alpha$, with $\llbracket \alpha \rrbracket = \{a_0, a_1, a_2, a_3, a_4\}$

  - $x = a_0$
  - $x = a_1$
  - $x = a_2$
  - $x = a_3$
  - $x = a_4$

- Functions: e.g., $f_{\beta \rightarrow \alpha}$, with $\llbracket \beta \rrbracket = \{b_0, b_1, b_2\}$

- Application, lambda abstraction
Corollary 2.103 (paraphrased)
The resulting propositional formula is satisfiable if and only if the HOL input formula has a standard model of the given size.
Optimizations

- Propositional simplification
- Term abbreviations
- Specialization for certain functions
- Undefined values, 3-valued logic
Optimizations

- Propositional simplification
- Term abbreviations
- **Specialization for certain functions**
- Undefined values, 3-valued logic
Extensions

- Type definitions, constant definitions, overloading
- Axiomatic type classes
- Data types, recursive functions
- Sets, records
- HOLCF
Extensions

- Type definitions, constant definitions, overloading
- Axiomatic type classes
- **Data types, recursive functions**
- Sets, records
- HOLCF
Case Studies

- The RSA-PSS security protocol
- Probabilistic programs
- A SAT-based Sudoku solver
Case Studies

- The RSA-PSS security protocol
  - *security* of an abstract formalization of the protocol

- Probabilistic programs

- A SAT-based Sudoku solver
Case Studies

- The RSA-PSS security protocol
  - security of an abstract formalization of the protocol

- Probabilistic programs
  - an abstract model of probabilistic programs

- A SAT-based Sudoku solver
Case Studies

- The RSA-PSS security protocol
  - security of an abstract formalization of the protocol

- Probabilistic programs
  - an abstract model of probabilistic programs

- A SAT-based Sudoku solver
  - a highly efficient solver with very little implementation effort
System Overview

Input formula → Preprocessing

Counterexample → Model

Theorem → Proof reconstruction

DIMACS CNF

satisfiable?

yes

no

Proof Trace

SAT Solver

Tjark Weber

SAT-based Finite Model Generation for Higher-Order Logic
Representation of SAT Problems

Naive: using HOL connectives $\land$, $\lor$
Representation of SAT Problems

Naive: using HOL connectives $\wedge$, $\vee$

Much better:

1. The whole CNF problem is assumed: $\{\bigwedge_{i=1}^{k} C_i\} \vdash \bigwedge_{i=1}^{k} C_i$.
2. Each clause is derived: $\{\bigwedge_{i=1}^{k} C_i\} \vdash C_1$, ..., $\{\bigwedge_{i=1}^{k} C_i\} \vdash C_k$.
3. Then a sequent representation is used:
   $$\{\bigwedge_{i=1}^{k} C_i, \overline{p_1}, \ldots, \overline{p_n}\} \vdash \text{False}.$$
Representation of SAT Problems

Naive: using HOL connectives $\land$, $\lor$

Much better:

1. The whole CNF problem is assumed: $\{\bigwedge_{i=1}^k C_i\} \vdash \bigwedge_{i=1}^k C_i$.
2. Each clause is derived: $\{\bigwedge_{i=1}^k C_i\} \vdash C_1$, $\ldots$, $\{\bigwedge_{i=1}^k C_i\} \vdash C_k$.
3. Then a sequent representation is used:

   $\{\bigwedge_{i=1}^k C_i, \neg p_1, \ldots, \neg p_n\} \vdash \text{False}.

The problem is a set of clauses.

- Clauses are sets of literals.
- Resolution is fast.
### Performance

#### Evaluation on SATLIB problems:

<table>
<thead>
<tr>
<th>Problem</th>
<th>Variables</th>
<th>Clauses</th>
<th>Resolutions</th>
<th>zChaff (s)</th>
<th>Isabelle (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>c7552mul.miter</td>
<td>11282</td>
<td>69529</td>
<td>242509</td>
<td>45</td>
<td>69</td>
</tr>
<tr>
<td>6pipe</td>
<td>15800</td>
<td>394739</td>
<td>310813</td>
<td>134</td>
<td>192</td>
</tr>
<tr>
<td>6pipe_6_ooo</td>
<td>17064</td>
<td>545612</td>
<td>782903</td>
<td>263</td>
<td>421</td>
</tr>
<tr>
<td>7pipe</td>
<td>23910</td>
<td>751118</td>
<td>497019</td>
<td>440</td>
<td>609</td>
</tr>
</tbody>
</table>

#### Evaluation on pigeonhole instances:

<table>
<thead>
<tr>
<th>Problem</th>
<th>Variables</th>
<th>Clauses</th>
<th>Resolutions</th>
<th>zChaff (s)</th>
<th>Isabelle (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>pigeon-9</td>
<td>90</td>
<td>415</td>
<td>73472</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>pigeon-10</td>
<td>110</td>
<td>561</td>
<td>215718</td>
<td>6</td>
<td>10</td>
</tr>
<tr>
<td>pigeon-11</td>
<td>132</td>
<td>738</td>
<td>601745</td>
<td>24</td>
<td>36</td>
</tr>
<tr>
<td>pigeon-12</td>
<td>156</td>
<td>949</td>
<td>3186775</td>
<td>247</td>
<td>315</td>
</tr>
</tbody>
</table>
Contributions

- A SAT-based **finite model generator** for higher-order logic
  - A satisfiability-equivalent translation from higher-order logic to propositional logic
  - Support for data types, recursive functions, etc.
  - Case studies
Contributions

- A SAT-based **finite model generator** for higher-order logic
  - A satisfiability-equivalent translation from higher-order logic to propositional logic
  - Support for data types, recursive functions, etc.
  - Case studies

- A highly optimized LCF-style **integration of proof-producing SAT solvers**
  - Dramatic performance improvements for propositional logic
  - Optimization techniques also applicable to other provers
Future Work

- Integration with Isabelle
- Optimizations
- External model generators
- Other methods of disproving
Future Work

- Integration with Isabelle
- Optimizations
- External model generators
- Other methods of disproving
- Analysis and optimization of resolution proofs
- SAT-based decision procedures beyond propositional logic
Future Work

- Integration with Isabelle
- Optimizations
- External model generators
- Other methods of disproving
- Analysis and optimization of resolution proofs
- SAT-based decision procedures beyond propositional logic
- Formalization
Thank you for your attention.