Finite Model Generation, Proof-Producing SAT Solvers, and SMT

Tjark Weber

UNIVERSITY OF CAMBRIDGE
Computer Laboratory

ARG Lunch
3 February 2009
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.
1. How can we find **counterexamples** in higher-order logic automatically?
Questions

1. How can we find counterexamples in higher-order logic automatically?

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

1. How can we find *counterexamples* in higher-order logic automatically?

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

3. Can we use efficient provers for *richer logics*, beyond SAT?
SAT-Based

Finite Model Generation

for Higher-Order Logic
Conjecture:

The transitive closure of \( A \cap B \) is equal to the intersection of the transitive closures of \( A_{(\alpha \times \alpha) \text{set}} \) and \( B_{(\alpha \times \alpha) \text{set}} \), i.e.,

\[
(A \cap B)^+ = A^+ \cap B^+
\]
Conjecture:

The transitive closure of $A \cap B$ is equal to the intersection of the transitive closures of $A_{(\alpha \times \alpha)}$ set and $B_{(\alpha \times \alpha)}$ set, i.e.,

$$(A \cap B)^+ = A^+ \cap B^+$$

Counterexample:

$\alpha = \{x, y\}$

$A = \{(x, y), (y, x), (y, y)\}$

$B = \{(x, x), (y, x), (y, y)\}$
HOL 4, Isabelle/HOL, etc.: higher-order logic, based on Church’s “simple theory of types” (1940)

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

Two special type constructors: bool and $\rightarrow$
Two logical constants: $\implies_{\text{bool} \rightarrow \text{bool} \rightarrow \text{bool}}$ 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.
  - \([\text{bool}] = \{\top, \bot\}\)
  - \([\sigma_1 \rightarrow \sigma_2] = [\sigma_2][\sigma_1]\)

- 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 $[\alpha] = \{a_0, a_1, a_2, a_3, a_4\}$

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

- Application, lambda abstraction
Soundness, Completeness

**Theorem**

The resulting propositional formula is **satisfiable** if and only if the HOL input formula has a standard **model** of the given size.

**Algorithm:**

1. **Input formula**
2. Fix a size for each base type
3. **Translation**
4. **satisfiable?**
   - yes: **Assignment**
   - no: Increase the model’s size
5. **HOL model**
Extensions and Optimizations

- Integrated with Isabelle/HOL (*refute*)
- Various optimizations
  - Propositional simplification
  - Term abbreviations
  - Specialization for certain functions
  - Undefined values, 3-valued logic
- Various 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
An LCF-Style Integration of Proof-Producing SAT Solvers
Propositional Logic

Propositional logic:

- Boolean variables: \( p, q, \ldots \)
- Formulae: \( \varphi ::= p \mid \neg \varphi \mid \varphi \lor \varphi \mid \varphi \land \varphi \)
Propositional Logic

Propositional logic:
- Boolean variables: $p, q, \ldots$
- Formulae: $\varphi ::= p \mid \neg \varphi \mid \varphi \lor \varphi \mid \varphi \land \varphi$

**Conjunctive normal form (CNF):** a conjunction of **clauses**, where each clause is a disjunction of **literals** (i.e., possibly negated variables)
Propositional Logic

Propositional logic:

- Boolean variables: $p, q, \ldots$
- Formulae: $\varphi ::= p \mid \neg \varphi \mid \varphi \lor \varphi \mid \varphi \land \varphi$

Conjunctive normal form (CNF): a conjunction of clauses, where each clause is a disjunction of literals (i.e., possibly negated variables)

Abstraction from higher-order to propositional logic: replace subterms by Boolean variables, e.g.,

$$(\forall x. P x) \lor \neg \left( \forall x. P x \right) \quad \mapsto \quad p \lor \neg p$$
Propositional Resolution

Theorem
Propositional resolution is **sound** and **refutation complete**.
System Overview

Theorem prover

Input formula → Preprocessing

Counterexample

Theorem → Proof reconstruction

DIMACS CNF

SAT solver

Model

Proof trace

satisfiable?

yes → no
Bad: use HOL connectives $\land$, $\lor$

Good: use *sets* of clauses and literals
Representation of SAT Problems

Bad: use HOL connectives $\land$, $\lor$

Good: use sets of clauses and literals

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

Bad: use HOL connectives $\land$, $\lor$

Good: use sets of clauses and literals

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

- The problem is an array 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>
Satisfiability Modulo Theories
**Goal:** To decide the satisfiability of (quantifier-free) first-order formulae with respect to combinations of (decidable) background theories.

\[ \varphi ::= A \mid \neg \varphi \mid \varphi \lor \varphi \mid \varphi \land \varphi \]

**Applications:**
- Formal verification
- Scheduling
- Compiler optimization
- ...
Example

Theories:

- $\mathcal{R}$: theory of rationals
  $\Sigma_\mathcal{R} = \{\leq, +, -, 0, 1\}$
- $\mathcal{L}$: theory of lists
  $\Sigma_\mathcal{L} = \{=, \text{hd}, \text{tl}, \text{nil}, \text{cons}\}$
- $\mathcal{E}$: theory of equality
  $\Sigma$: free function and predicate symbols

Problem: Is

$$x \leq y \land y \leq x + \text{hd} (\text{cons} \ 0 \ \text{nil}) \land P (f \ x - f \ y) \land \neg P \ 0$$

satisfiable in $\mathcal{R} \cup \mathcal{L} \cup \mathcal{E}$?
SMT solvers typically use a combination of SAT solving and theory-specific decision procedures.

- **DPLL**: standard decision procedure for SAT (based on splitting and unit propagation)
- **Nelson-Oppen**: a decision procedure for the union of decidable theories (using variable abstraction and equality propagation)
- **DPLL(T)**: tight integration of a theory-specific decision procedure with the DPLL algorithm
SMT-LIB

Collection of SMT benchmark problems

- Standard syntax
- Various theories (arrays, bit vectors, integers, reals)
- Many logics (difference logic, linear arithmetic, ...)
- http://goedel.cs.uiowa.edu/smtlib/

Greatly helped to unify the field!
Satisfiability Modulo Theories Competition

- Annual satellite event of CAV (since 2005)
- Many different categories
- Many participating solvers: Barcelogic, clsat, CVC3, MathSAT, Yices, Z3, ...
- http://www.smtcomp.org/

Stimulates further solver improvement!
Future Work

3-year EPSRC research project “Expressive Multi-theory Reasoning for Interactive Verification” (until Dec. 2011)

- LCF-style integration of SMT solvers
- Improved quantifier support
- Performance enhancements
- Validation case studies
Thank you for your attention.