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

Tjark Weber

UNIVERSITY OF CAMBRIDGE
Computer Laboratory

ARG Lunch

3 February 2009

Motivation

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. 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

Example

Conjecture:

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

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

Example

Conjecture:

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

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

Counterexample:

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

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

Higher-Order Logic

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: $t_\sigma ::= x_\sigma \mid c_\sigma \mid (t_{\sigma' \to \sigma}\ t'_{\sigma'})_\sigma \mid (\lambda x_{\sigma_1}.\, t_{\sigma_2})_{\sigma_1 \to \sigma_2}$

Two special type constructors: bool and $\to$

Two logical constants: $\Longrightarrow_{\mathrm{bool} \to \mathrm{bool} \to \mathrm{bool}}$ and $=_{\sigma \to \sigma \to \mathrm{bool}}$

The Semantics of HOL

Standard set-theoretic semantics:

- Types denote certain non-empty sets.
  - $[\![\text{bool}]\!] = \{\top, \bot\}$
  - $[\![\sigma_1 \to \sigma_2]\!] = [\![\sigma_2]\!]^{[\![\sigma_1]\!]}$
- Terms denote elements of these sets.

The Semantics of HOL

Standard set-theoretic semantics:

- Types denote certain non-empty finite sets.
  - $[\![\text{bool}]\!] = \{\top, \bot\}$
  - $[\![\sigma_1 \to \sigma_2]\!] = [\![\sigma_2]\!]^{[\![\sigma_1]\!]}$
- Terms denote elements of these sets.

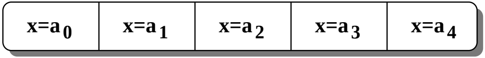
Translation to Propositional Logic

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

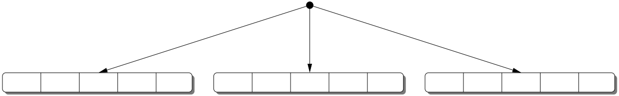
x=a$_0$	**x=a$_1$**	**x=a$_2$**	**x=a$_3$**	**x=a$_4$**

Translation to Propositional Logic

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

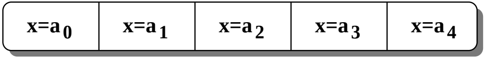


- Functions: e.g., $f_{\beta\to\alpha}$, with $[\![\beta]\!] = \{b_0, b_1, b_2\}$

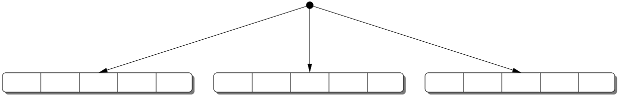


Translation to Propositional Logic

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



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



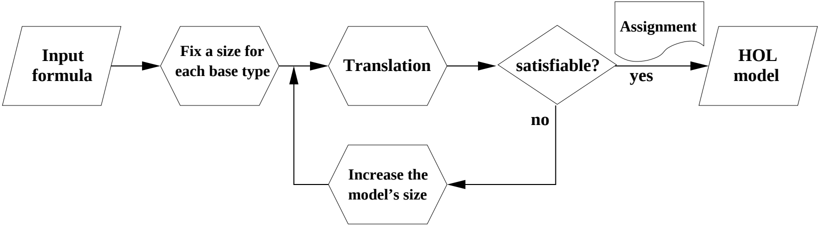
- Application, lambda abstraction

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:



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, ...
- Formulae: $\varphi ::= p \mid \neg\varphi \mid \varphi \vee \varphi \mid \varphi \wedge \varphi$

Propositional Logic

Propositional logic:

- Boolean variables: p, q, ...
- Formulae: $\varphi ::= p \mid \neg\varphi \mid \varphi \vee \varphi \mid \varphi \wedge \varphi$

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, ...
- Formulae: $\varphi ::= p \mid \neg\varphi \mid \varphi \vee \varphi \mid \varphi \wedge \varphi$

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.,

$$(\forall x.\, P\, x) \vee \neg(\forall x.\, P\, x) \quad \mapsto \quad p \vee \neg p$$

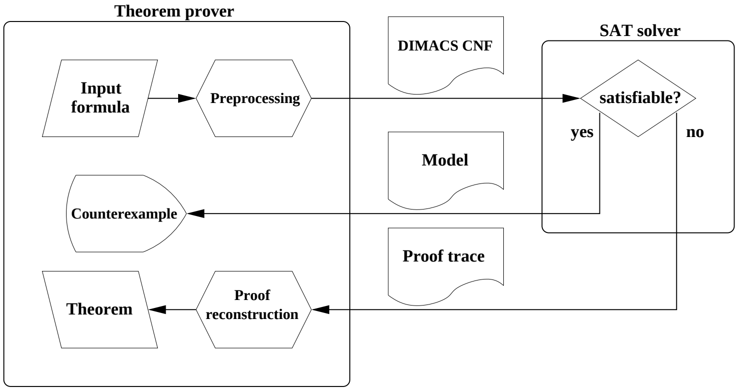
Propositional Resolution

$$\frac{P \cup \{x\} \qquad Q \cup \{\neg x\}}{P \cup Q}$$

Theorem

Propositional resolution is sound and refutation complete.

System Overview



Representation of SAT Problems

Bad: use HOL connectives $\wedge$, $\vee$

Good: use sets of clauses and literals

Representation of SAT Problems

Bad: use HOL connectives $\wedge$, $\vee$

Good: use sets of clauses and literals

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:

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

Representation of SAT Problems

Bad: use HOL connectives $\wedge$, $\vee$

Good: use sets of clauses and literals

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

- The problem is an array of clauses. Clauses are sets of literals.
- Resolution is fast.

Performance

Evaluation on SATLIB problems:

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

Evaluation on pigeonhole instances:

Problem	Variables	Clauses	Resolutions	zChaff (s)	Isabelle (s)
pigeon-9	90	415	73472	1	3
pigeon-10	110	561	215718	6	10
pigeon-11	132	738	601745	24	36
pigeon-12	156	949	3186775	247	315

Satisfiability Modulo Theories

Satisfiability Modulo Theories

Goal: To decide the satisfiability of (quantifier-free) first-order formulae with respect to combinations of (decidable) background theories.

$$\varphi ::= \mathcal{A} \mid \neg\varphi \mid \varphi \vee \varphi \mid \varphi \wedge \varphi$$

Applications:

- Formal verification
- Scheduling
- Compiler optimization
- ...

Example

Theories:

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

Problem: Is

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

satisfiable in $\mathcal{R} \cup \mathcal{L} \cup \mathcal{E}$?

Algorithms

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!

SMT-COMP

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

Questions?

Thank you for your attention.