SAT-based Finite Model Generation for Higher-Order Logic

Tjark Weber

TECHNISCHE UNIVERSITÄT MÜNCHEN

October 9, 2008

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

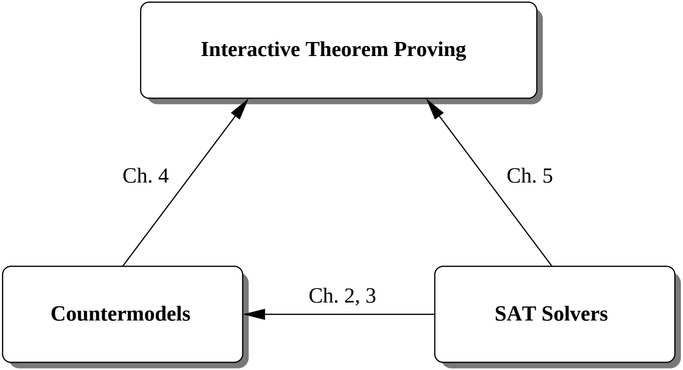
1. Can we use efficient SAT solvers to find counterexamples in higher-order logic automatically?

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

Higher-Order Logic

Isabelle/HOL: 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.
  - $[\![\mathrm{bool}]\!] = \{\top, \bot\}$
  - $[\![\sigma_1 \rightarrow \sigma_2]\!] = [\![\sigma_2]\!]^{[\![\sigma_1]\!]}$
- Terms denote elements of these sets.

The Semantics of HOL

Standard set-theoretic semantics:

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

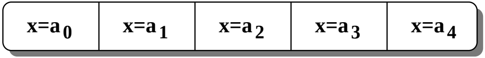
Translation to Propositional Logic

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

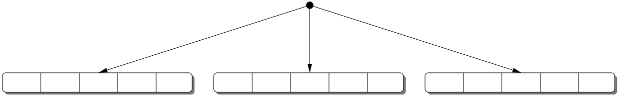
$x=a_0$	$x=a_1$	$x=a_2$	$x=a_3$	$x=a_4$

Translation to Propositional Logic

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

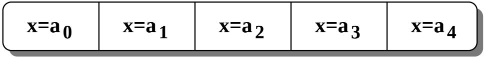


- Functions: e.g., $f_{\beta\to\alpha}$, with $[\![\beta]\!] = \{b_0, b_1, b_2\}$

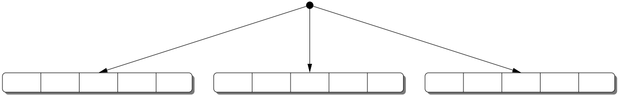


Translation to Propositional Logic

- 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

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

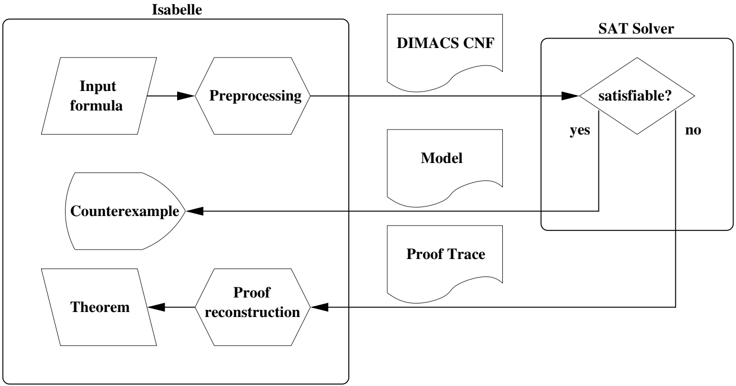
Case Studies

- 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



Representation of SAT Problems

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

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:

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

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:

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

- The problem is a set 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

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

Questions?

Thank you for your attention.