Bounded Model Generation for Isabelle/HOL

and Related Applications of SAT Solvers in Interactive Theorem Proving

Tjark Weber
webertj@in.tum.de

Winterhütte, März 2005
Isabelle is a generic proof assistant:

- Highly flexible
- Interactive
- Automatic proof procedures
- Advanced user interface
- Readable proofs
- Large theories of formal mathematics
Bounded Model Generation

Theorem proving: from formulae to proofs
Bounded model generation: *from formulae to models*

Applications:

- Finding counterexamples to false conjectures
- Showing the consistency of a specification
- Solving open mathematical problems
- Guiding resolution-based provers
**Isabelle/HOL**

*HOL*: higher-order logic based on Church’s simple theory of types (1940)

Simply-typed $\lambda$-calculus:

- **Types**: $\sigma ::= \mathbb{B} \mid \alpha \mid \sigma \to \sigma$
- **Terms**: $t_\sigma ::= x_\sigma \mid (t_{\sigma'\to\sigma} t'_{\sigma'})_\sigma \mid (\lambda x_{\sigma_1}. t_{\sigma_2})_{\sigma_1\to\sigma_2}$

Two logical constants:

- $\implies \mathbb{B} \to \mathbb{B} \to \mathbb{B}, \equiv \sigma \to \sigma \to \mathbb{B}$
**Isabelle/HOL**

**HOL**: higher-order logic based on Church’s simple theory of types (1940)

Simply-typed \(\lambda\)-calculus:

- **Types**: \( \sigma ::= \mathbb{B} \mid \alpha \mid \sigma \rightarrow \sigma \)
- **Terms**: \( t_\sigma ::= x_\sigma \mid (t_{\sigma' \rightarrow \sigma} \ t_{\sigma'})_\sigma \mid (\lambda x_{\sigma_1} \cdot t_{\sigma_2})_{\sigma_1 \rightarrow \sigma_2} \)

Two logical constants:

- \( \implies \mathbb{B} \rightarrow \mathbb{B} \rightarrow \mathbb{B} \), \( = \sigma \rightarrow \sigma \rightarrow \mathbb{B} \)

Other constants, e.g.

\[
\text{True} \mid \text{False} \mid \neg \mid \land \mid \lor \mid \forall \mid \exists \mid \exists!
\]

are definable.
The Semantics of HOL

Set-theoretic semantics:
- Types denote certain sets.
- Terms denote elements of these sets.
The Semantics of HOL

Set-theoretic semantics:

- Types denote certain sets.
- Terms denote elements of these sets.

An environment \( D \) assigns to each type variable \( \alpha \) a non-empty set \( D_\alpha \).

Semantics of types:

- \( D(\mathbb{B}) = \{ \top, \bot \} \)
- \( D(\alpha) = D_\alpha \)
- \( D(\sigma_1 \rightarrow \sigma_2) = D(\sigma_2)^{D(\sigma_1)} \)
A variable assignment $A$ maps each variable $x_\sigma$ to an element $A(x_\sigma)$ in $D(\sigma)$.

Semantics of terms:

- $[x_\sigma]_D^A = A(x_\sigma)$
- $[(t_{\sigma'} \rightarrow_\sigma t_{\sigma'})]_D^A = [t_{\sigma'} \rightarrow_\sigma]_D^A([t_{\sigma'}]_D^A)$
- $[(\lambda x_{\sigma_1} \cdot t_{\sigma_2})_{\sigma_1 \rightarrow \sigma_2}]_D^A$ is the function that sends each $d$ in $D(\sigma_1)$ to $[t_{\sigma_2}]_D^A[x_{\sigma_1} \rightarrow d]$

$\rightarrow : \text{implication}, \quad = : \text{equality}$

Hence the semantics of a term is an element of the set denoted by the term’s type.
Overview

Input: HOL formula $\phi$

Output: either a model for $\phi$, or “no model found”
Overview

Input: HOL formula $\phi$

1. Fix a finite environment $D$.

2. Translate $\phi$ into a Boolean formula that is satisfiable iff
   \[
   [\phi]_D^A = \top \text{ for some variable assignment } A.
   \]

3. Use a SAT solver to search for a satisfying assignment.

4. If a satisfying assignment was found, compute from it the variable assignment $A$. Otherwise repeat for a larger environment.

Output: either a model for $\phi$, or “no model found”
Fixing a Finite Environment

Fix a positive integer for every type variable that occurs in the typing of $\phi$.

Every type then has a finite size:

- $|\mathbb{B}| = 2$
- $|\alpha|$ is given by the environment
- $|\sigma_1 \rightarrow \sigma_2| = |\sigma_2|^{\sigma_1}$

Finite model generation is a generalization of satisfiability checking, where the search tree is not necessarily binary.
Several *external* SAT solvers (zChaff, BerkMin, Jerusat, ...) are supported.

- Efficiency
- Advances in SAT solver technology are “for free”
The SAT Solver

Several *external* SAT solvers (zChaff, BerkMin, Jerusat, . . . ) are supported.

- Efficiency
- Advances in SAT solver technology are “for free”

Simple *internal* solvers are available as well.

- Easy installation
- Compatibility
- Fast enough for small examples
Some Extensions

**Sets** are interpreted as characteristic functions.

- $\sigma \text{ set} \cong \sigma \to \mathbb{B}$
- $x \in P \cong P \, x$
- $\{x. \, P \, x\} \cong P$

**Non-recursive datatypes** can be interpreted in a finite model.

- $(\alpha_1, \ldots, \alpha_n)\sigma ::= C_1 \, \sigma_1^1 \cdots \sigma_{m_1}^1 \cdots \cdots C_k \, \sigma_k^1 \cdots \sigma_{m_k}^k$
- $| (\alpha_1, \ldots, \alpha_n)\sigma | = \sum_{i=1}^k \prod_{j=1}^{m_i} |\sigma_i^j|$

Examples: *option, sum, product* types
Some Extensions

**Recursive datatypes** are restricted to initial fragments.

- Examples: $\text{nat}, \sigma \text{ list}, \text{lambdaterm}$
- $\text{nat}^1 = \{0\}$, $\text{nat}^2 = \{0, 1\}$, $\text{nat}^3 = \{0, 1, 2\}, \ldots$
- This works for datatypes that occur only positively.

Datatype **constructors** and **recursive functions** can be interpreted as partial functions.

- Examples: $\text{Suc}_{\text{nat}} \rightarrow \text{nat}$, $+_{\text{nat}} \rightarrow \text{nat} \rightarrow \text{nat}$, $\odot_{\sigma \text{ list}} \rightarrow \sigma \text{ list} \rightarrow \sigma \text{ list}$
- 3-valued logic: true, false, unknown

**Axiomatic type classes** introduce additional axioms that must be satisfied by the model.

**Records** and **inductively defined sets** can be treated as well.
Soundness and Completeness

If the SAT solver is sound/complete, we have ...

- **Soundness**: The algorithm returns “model found” only if the given formula has a finite model.

- **Completeness**: If the given formula has a finite model, the algorithm will find it (given enough time).

- **Minimality**: The model found is a smallest model for the given formula.
“No Model Found”
Unsatisfiability – Helpful at All?

- If the Boolean formula is unsatisfiable, the HOL formula $\phi$ does not have a model of a certain size.
- If $\phi$ has the finite model property, we can test all models up to the required size.
- If no model is found, $\neg \phi$ must be provable.

Difficult to implement . . . let’s only look at Boolean formulae for now.
Deciding Boolean Formulae with zChaff

Isabelle

Input formula → Preprocessing

Counterexample

Theorem → Proof reconstruction

DIMACS CNF

zChaff

satisfiable?

Assignment

Trace

yes

no
The Algorithm

Preprocessing:

- No conversion from HOL is necessary, only from Boolean logic into CNF.
- But the conversion must be \textit{proof-generating}, i.e. return a theorem $\phi = \phi_{\text{CNF}}$. 
The Algorithm

Preprocessing:

- No conversion from HOL is necessary, only from Boolean logic into CNF.
- But the conversion must be \textit{proof-generating}, i.e. return a theorem $\phi = \phi_{\text{CNF}}$.

Proof reconstruction:

- \texttt{zChaff} returns a \textit{resolution-style proof} of unsatisfiability.
- The proof is replayed in \texttt{Isabelle/HOL} to derive $\neg \phi$. 
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_tac
  - 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>X</td>
<td>X</td>
<td>X</td>
<td>726.5</td>
</tr>
<tr>
<td>NUM285-1</td>
<td>sat.</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>0.2</td>
</tr>
<tr>
<td>PUZ013-1</td>
<td>unsat.</td>
<td>0.5</td>
<td>X</td>
<td>5.0</td>
<td>0.1</td>
</tr>
<tr>
<td>PUZ014-1</td>
<td>unsat.</td>
<td>1.4</td>
<td>X</td>
<td>6.1</td>
<td>0.1</td>
</tr>
<tr>
<td>PUZ015-2.006</td>
<td>unsat.</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>10.5</td>
</tr>
<tr>
<td>PUZ016-2.004</td>
<td>sat.</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>0.3</td>
</tr>
<tr>
<td>PUZ016-2.005</td>
<td>unsat.</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>1.6</td>
</tr>
<tr>
<td>PUZ030-2</td>
<td>unsat.</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>0.7</td>
</tr>
<tr>
<td>PUZ033-1</td>
<td>unsat.</td>
<td>0.2</td>
<td>6.4</td>
<td>0.1</td>
<td>0.1</td>
</tr>
<tr>
<td>SYN001-1.005</td>
<td>unsat.</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>0.4</td>
</tr>
<tr>
<td>SYN003-1.006</td>
<td>unsat.</td>
<td>0.9</td>
<td>X</td>
<td>1.6</td>
<td>0.1</td>
</tr>
<tr>
<td>SYN004-1.007</td>
<td>unsat.</td>
<td>0.3</td>
<td>822.2</td>
<td>2.8</td>
<td>0.1</td>
</tr>
<tr>
<td>SYN010-1.005.005</td>
<td>unsat.</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>0.4</td>
</tr>
<tr>
<td>SYN086-1.003</td>
<td>sat.</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>0.1</td>
</tr>
<tr>
<td>SYN087-1.003</td>
<td>sat.</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>0.1</td>
</tr>
<tr>
<td>SYN090-1.008</td>
<td>unsat.</td>
<td>13.8</td>
<td>X</td>
<td>X</td>
<td>0.5</td>
</tr>
<tr>
<td>SYN091-1.003</td>
<td>sat.</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>0.1</td>
</tr>
<tr>
<td>SYN092-1.003</td>
<td>sat.</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>0.1</td>
</tr>
<tr>
<td>SYN093-1.002</td>
<td>unsat.</td>
<td>1290.8</td>
<td>16.2</td>
<td>1126.6</td>
<td>0.1</td>
</tr>
<tr>
<td>SYN094-1.005</td>
<td>unsat.</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>0.8</td>
</tr>
<tr>
<td>SYN097-1.002</td>
<td>unsat.</td>
<td>X</td>
<td>19.2</td>
<td>X</td>
<td>0.2</td>
</tr>
<tr>
<td>SYN098-1.002</td>
<td>unsat.</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>0.4</td>
</tr>
<tr>
<td>SYN302-1.003</td>
<td>sat.</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>0.4</td>
</tr>
</tbody>
</table>
Conclusions and Future Work

- Finite countermodels for HOL formulae
- A fast decision procedure for Boolean formulae

- Further optimizations, benchmarks
- A SAT-based decision procedure for a fragment of HOL
- Integration of external model generators
- …