Bounded Model Generation for Isabelle/HOL

Using a SAT Solver

Tjark Weber
webertj@in.tum.de

IJCAR – Workshop on Disproving
Cork, July 5th, 2004
Isabelle

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: $\sigma ::= \mathbb{B} | \alpha | \sigma \rightarrow \sigma$
- 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:

- $\implies \mathbb{B} \rightarrow \mathbb{B} \rightarrow \mathbb{B}, = \sigma \rightarrow \sigma \rightarrow \mathbb{B}$
Isabelle/HOL

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

Simply-typed $\lambda$-calculus:

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

Two logical constants:

- $\rightarrow B \rightarrow B \rightarrow B$, $\equiv \sigma \rightarrow \sigma \rightarrow B$

Other constants, e.g.

- $\text{True} | \text{False} | \neg | \land | \lor | \forall | \exists | \exists!$

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:

- $D(\mathbb{B}) = \{\top, \bot\}$
- $D(\alpha) = D_\alpha$
- $D(\sigma_1 \rightarrow \sigma_2) = D(\sigma_2)^{D(\sigma_1)}$
The Semantics of HOL (2)

A **variable assignment** $A$ maps each variable $x_\sigma$ to an element $A(x_\sigma)$ in $D(\sigma)$.

**Semantics of terms:**

1. $[x_\sigma]^A_D = A(x_\sigma)$
2. $[\sigma' \rightarrow \sigma \ t_\sigma']^A_D = [\sigma' \rightarrow \sigma]^A_D([t_\sigma']^A_D)$
3. $[\lambda x_{\sigma_1} \cdot t_{\sigma_2}]_{\sigma_1 \rightarrow \sigma_2}^A_D$ is the function that sends each $d$ in $D(\sigma_1)$ to $[t_{\sigma_2}]^A_D[x_{\sigma_1} \mapsto d]$

$\rightarrow\beta \rightarrow B, =_{\sigma} \rightarrow \sigma \rightarrow B$: implication, 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 $\left[\phi\right]_D^A = T$ 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:

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

Finite model generation is a generalization of satisfiability checking, where the search tree is not necessarily binary.
Translation into a Boolean Formula

Boolean formulae:

\[ \varphi ::= \text{True} | \text{False} | p | \neg \varphi | \varphi \lor \varphi | \varphi \land \varphi \]

Idea: Translate a HOL term \( t_\sigma \) into a tree of Boolean formulae. The interpretation of the Boolean variables in the tree determines the interpretation of \( t_\sigma \).
Translation into a Boolean Formula

Boolean formulae:

\[ \varphi ::= \text{True} | \text{False} | p | \neg \varphi | \varphi \lor \varphi | \varphi \land \varphi \]

Idea: Translate a HOL term \( t_\sigma \) into a tree of Boolean formulae. The interpretation of the Boolean variables in the tree determines the interpretation of \( t_\sigma \).

- \( \text{create}(\mathbb{B}) = [p_1, p_2] \)
- \( \text{create}(\alpha) = [p_1, \ldots, p|\alpha|] \)
- \( \text{create}(\sigma_1 \rightarrow \sigma_2) = [\text{create}(\sigma_2), \ldots, \text{create}(\sigma_2)] \)
Translation into a Boolean Formula (2)

- treemap\( (f, t) \) applies \( f \) to every node of the tree \( t \)
- merge\( (f, t_1, t_2) \) applies \( f \) to corresponding nodes of \( t_1 \) and \( t_2 \)
- \( \text{apply}([t], [\varphi]) = \text{treemap}((\lambda \varphi'. \varphi' \land \varphi), t) \)
- \( \text{apply}([t_1, t_2, \ldots, t_n], [\varphi_1, \varphi_2, \ldots, \varphi_n]) = \text{merge}(\lor, \text{apply}([t_1], [\varphi_1]), \text{apply}([t_2, \ldots, t_n], [\varphi_2, \ldots, \varphi_n])) \)
- \( \text{enum}([\varphi_1, \ldots, \varphi_n]) = [\varphi_1, \ldots, \varphi_n] \)
- \( \text{enum}([t_1, \ldots, t_n]) = \text{map(all, pick([enum(t_1), \ldots, enum(t_n)])))} \)
Translation into a Boolean Formula (3)

\[ T^B_D(x_\sigma) = \begin{cases} B(x_\sigma) & \text{if } x_\sigma \in \text{dom } B \\ \text{create}(\sigma) & \text{otherwise} \end{cases} \]

\[ T^B_D((t_{\sigma' \rightarrow \sigma} t'_{\sigma'})_\sigma) = \text{apply}(T^B_D(t_{\sigma' \rightarrow \sigma}), \text{enum}(T^B_D(t'_{\sigma'}))) \]

\[ T^B_D((\lambda x_{\sigma_1} t_{\sigma_2})_{\sigma_1 \rightarrow \sigma_2}) = [T^B_D[x_{\sigma_1} \mapsto d_1](t_{\sigma_2}), \ldots, T^B_D[x_{\sigma_1} \mapsto d_{|\sigma_1|}](t_{\sigma_2})], \]

where \([d_1, \ldots, d_{|\sigma_1|}] = \text{consts}(\sigma_1)\)
Example Translation

\[ S : \alpha \rightarrow \beta, T : \alpha, |\alpha| = 2, |\beta| = 3 \]
Example Translation

\[ S : \alpha \rightarrow \beta, \quad T : \alpha, \quad |\alpha| = 2, \quad |\beta| = 3 \]

\[ S = [[s_1^1, s_2^1, s_3^1], [s_1^2, s_2^2, s_3^2]] \]

\[ T = [t_1, t_2] \]
Example Translation

\[ S : \alpha \rightarrow \beta, \ T : \alpha, \ |\alpha| = 2, \ |\beta| = 3 \]

\[ S = [[s_1^1, s_2^1, s_3^1], [s_1^2, s_2^2, s_3^2]] \]

\[ T = [t_1, t_2] \]

\[ (ST) = [s_1^1 \land t_1, s_2^1 \land t_1, s_3^1 \land t_1] \]
Example Translation

\[ S : \alpha \rightarrow \beta, \ T : \alpha, \ |\alpha| = 2, \ |\beta| = 3 \]

\[ S = [[s^1_1, s^1_2, s^1_3], [s^2_1, s^2_2, s^2_3]] \]

\[ T = [t_1, t_2] \]

\[ (S T) = [s^1_1 \wedge t_1 \vee s^2_1 \wedge t_2, s^1_2 \wedge t_1 \vee s^2_2 \wedge t_2, s^1_3 \wedge t_1 \vee s^2_3 \wedge t_2] \]
The SAT Solver

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.

- $(\alpha_1, \ldots, \alpha_n)\sigma ::= C_1 \sigma_1^1 \ldots \sigma_{m_1}^1 \ldots | C_k \sigma_1^k \ldots \sigma_{m_k}^k$
- $|(\alpha_1, \ldots, \alpha_n)\sigma| = \sum_{i=1}^{k} \prod_{j=1}^{m_i} |\sigma_j^i|$
- Examples: option, sum, product types
Some Optimizations

- At most one Boolean variable is used for types $\sigma$ with $|\sigma| \leq 2$
- On-the-fly simplification of the Boolean formula (e.g. closed HOL formulae simply become True/False)
- Hard-coded translation for logical constants
- Specialization of the rule for function application
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.
Implementation

- Seamless integration with Isabelle/HOL
- Roughly 2,800 lines of ML code
- Several user-definable parameters (e.g. a runtime limit)
Future Work

- Serious applications:
  - Benchmarks
  - Cryptographic protocols
- Support for other HOL constructs:
  - Inductive datatypes
  - Recursive functions
  - ...
- A better translation:
  - Fewer Boolean variables
  - Shorter Boolean formulae