Bounded Model Generation for Isabelle/HOL

Using a SAT Solver

Tjark Weber

`webertj@in.tum.de`

TECHNISCHE UNIVERSITÄT MÜNCHEN

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 λ-calculus:

- Types: $\sigma ::= \mathbb{B}$ | α | $\sigma \to \sigma$
- Terms: $t_\sigma ::= x_\sigma$ | $(t_{\sigma' \to \sigma}\, t_{\sigma'})_\sigma$ | $(\lambda x_{\sigma_1} .\, t_{\sigma_2})_{\sigma_1 \to \sigma_2}$

Two logical constants:

- $\Longrightarrow_{\mathbb{B} \to \mathbb{B} \to \mathbb{B}}$, $=_{\sigma \to \sigma \to \mathbb{B}}$

Isabelle/HOL

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

Simply-typed λ-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:

- $\Longrightarrow_{\mathbb{B} \to \mathbb{B} \to \mathbb{B}}$, $=_{\sigma \to \sigma \to \mathbb{B}}$

Other constants, e.g.

$$\texttt{True} \mid \texttt{False} \mid \neg \mid \wedge \mid \vee \mid \forall \mid \exists \mid \exists!$$

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 α a non-empty set D_α.

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_σ to an element $A(x_\sigma)$ in $D(\sigma)$.

Semantics of terms:

- $[\![x_\sigma]\!]_D^A = A(x_\sigma)$
- $[\![(t_{\sigma' \to \sigma}\, t_{\sigma'})_\sigma]\!]_D^A = [\![t_{\sigma' \to \sigma}]\!]_D^A([\![t_{\sigma'}]\!]_D^A)$
- $[\![(\lambda x_{\sigma_1} .\, t_{\sigma_2})_{\sigma_1 \to \sigma_2}]\!]_D^A$ is the function that sends each d in $D(\sigma_1)$ to $[\![t_{\sigma_2}]\!]_D^{A[x_{\sigma_1} \mapsto d]}$
- $\Longrightarrow_{\mathbb{B} \to \mathbb{B} \to \mathbb{B}}, =_{\sigma \to \sigma \to \mathbb{B}}$: implication, equality

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

Overview

Input: HOL formula ϕ

Output: either a model for ϕ, or “no model found”

Overview

Input: HOL formula ϕ

1. Fix a finite environment D.
2. Translate ϕ into a Boolean formula that is satisfiable iff $[\![\phi]\!]_D^A = \top$ 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 ϕ, or "no model found"

Fixing a Finite Environment

Fix a positive integer for every type variable that occurs in the typing of ϕ.

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.

Translation into a Boolean Formula

Boolean formulae:

$$\varphi ::= \texttt{True} \mid \texttt{False} \mid p \mid \neg\varphi \mid \varphi \vee \varphi \mid \varphi \wedge \varphi$$

Idea: Translate a HOL term t_σ into a *tree of Boolean formulae*. The interpretation of the Boolean variables in the tree determines the interpretation of t_σ.

Translation into a Boolean Formula

Boolean formulae:

$$\varphi ::= \texttt{True} \mid \texttt{False} \mid p \mid \neg\varphi \mid \varphi \vee \varphi \mid \varphi \wedge \varphi$$

Idea: Translate a HOL term t_σ into a *tree of Boolean formulae*. The interpretation of the Boolean variables in the tree determines the interpretation of t_σ.

- $\mathrm{create}(\mathbb{B}) = [p_1, p_2]$
- $\mathrm{create}(\alpha) = [p_1, \ldots, p_{|\alpha|}]$
- $\mathrm{create}(\sigma_1 \to \sigma_2) = [\underbrace{\mathrm{create}(\sigma_2), \ldots, \mathrm{create}(\sigma_2)}_{|\sigma_1|}]$

Translation into a Boolean Formula (2)

- $\mathrm{treemap}(f, t)$ applies f to every node of the tree t
- $\mathrm{merge}(f, t_1, t_2)$ applies f to corresponding nodes of t_1 and t_2

- $\mathrm{apply}([t], [\varphi]) = \mathrm{treemap}((\lambda\varphi'.\varphi' \wedge \varphi), t)$
- $\mathrm{apply}([t_1, t_2, \ldots, t_n], [\varphi_1, \varphi_2, \ldots, \varphi_n]) =$ $\mathrm{merge}(\vee, \mathrm{apply}([t_1], [\varphi_1]), \mathrm{apply}([t_2, \ldots, t_n], [\varphi_2, \ldots, \varphi_n]))$

- $\mathrm{enum}([\varphi_1, \ldots, \varphi_n]) = [\varphi_1, \ldots, \varphi_n]$
- $\mathrm{enum}([t_1, \ldots, t_n]) =$ $\mathrm{map}(\mathrm{all}, \mathrm{pick}([\mathrm{enum}(t_1), \ldots, \mathrm{enum}(t_n)]))$

Translation into a Boolean Formula (3)

- $\mathcal{T}_D^B(x_\sigma) = \begin{cases} B(x_\sigma) & \text{if } x_\sigma \in \mathrm{dom}\, B \\ \mathrm{create}(\sigma) & \text{otherwise} \end{cases}$
- $\mathcal{T}_D^B((t_{\sigma'\to\sigma}\, t'_{\sigma'})_\sigma) = \mathrm{apply}(\mathcal{T}_D^B(t_{\sigma'\to\sigma}), \mathrm{enum}(\mathcal{T}_D^B(t'_{\sigma'})))$
- $\mathcal{T}_D^B((\lambda x_{\sigma_1}.\, t_{\sigma_2})_{\sigma_1\to\sigma_2}) =$
  $[\mathcal{T}_D^{B[x_{\sigma_1}\mapsto d_1]}(t_{\sigma_2}), \ldots, \mathcal{T}_D^{B[x_{\sigma_1}\mapsto d_{|\sigma_1|}]}(t_{\sigma_2})]$,
  where $[d_1, \ldots, d_{|\sigma_1|}] = \mathrm{consts}(\sigma_1)$

Example Translation

$S : \alpha \to \beta$, $T : \alpha$, $|\alpha| = 2$, $|\beta| = 3$

Example Translation

$S : \alpha \to \beta$, $T : \alpha$, $|\alpha| = 2$, $|\beta| = 3$

$S = [[s_1^1, s_2^1, s_3^1], [s_1^2, s_2^2, s_3^2]]$
$T = [t_1, t_2]$

Example Translation

$S : \alpha \to \beta$, $T : \alpha$, $|\alpha| = 2$, $|\beta| = 3$

$$S = [[s_1^1, s_2^1, s_3^1], [s_1^2, s_2^2, s_3^2]]$$
$$T = [t_1, t_2]$$

$$(S\,T) = [s_1^1 \wedge t_1 \qquad ,s_2^1 \wedge t_1 \qquad ,s_3^1 \wedge t1 \qquad]$$

Example Translation

$S : \alpha \to \beta$, $T : \alpha$, $|\alpha| = 2$, $|\beta| = 3$

$S = [[s_1^1, s_2^1, s_3^1], [s_1^2, s_2^2, s_3^2]]$
$T = [t_1, t_2]$

$(S\,T) = [s_1^1 \wedge t_1 \vee s_1^2 \wedge t_2, s_2^1 \wedge t_1 \vee s_2^2 \wedge t_2, s_3^1 \wedge t_1 \vee s_3^2 \wedge t_2]$

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.

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

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 σ 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