SAT-based Finite Model Generation for Isabelle/HOL

Tjark Weber

webertj@in.tum.de

Summer School Marktoberdorf, August 9, 2005

Isabelle

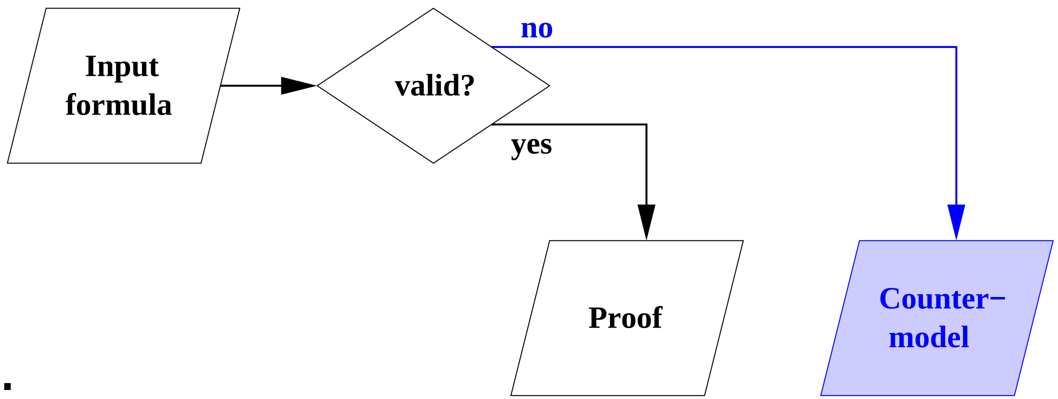
Isabelle is a generic proof assistant:

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

Finite Model Generation

Theorem proving: from formulae to proofs

Finite 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 | $\sigma \rightarrow \sigma$
- Terms: $t_\sigma ::= x_\sigma$ | $(t_{\sigma' \rightarrow \sigma}\, t_{\sigma'})_\sigma$ | $(\lambda x_{\sigma_1} .\, t_{\sigma_2})_{\sigma_1 \rightarrow \sigma_2}$

Two logical constants:

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

Isabelle/HOL

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

Simply-typed λ-calculus:

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

Two logical constants:

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

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:

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

Overview

Input: HOL formula ϕ

Output: either a model for ϕ, or "no model found"

Overview

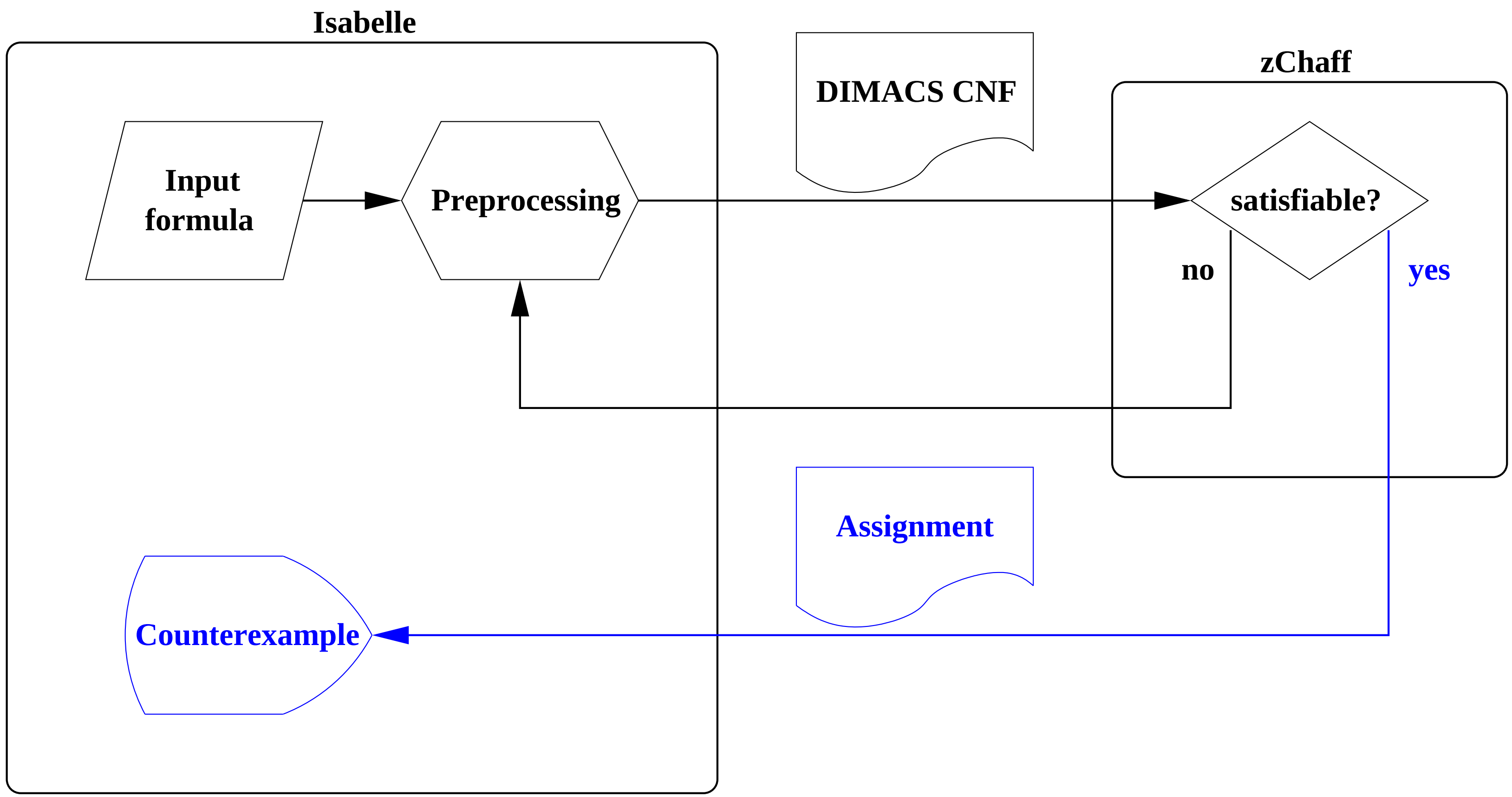
Input: HOL formula ϕ

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"

Overview

Input: HOL formula ϕ



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.

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.

- σ `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 Extensions

Recursive datatypes are restricted to initial fragments.

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

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

- Examples: $\mathrm{Suc}_{\mathrm{nat} \rightarrow \mathrm{nat}}$, $+_{\mathrm{nat} \rightarrow \mathrm{nat} \rightarrow \mathrm{nat}}$, $@_{\sigma \mathrm{list} \rightarrow \sigma \mathrm{list} \rightarrow \sigma \mathrm{list}}$
- 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.

Conclusions and Future Work

- Finite countermodels for HOL formulae

- Further optimizations, benchmarks
- SAT-based decision procedures for fragments of HOL
- Integration of external model generators