Isabelle/HOL and SMT

Sascha Böhme

Technische Universität München

September 10, 2009
1. Introduction
   - Isabelle/HOL
   - SMT
   - Isabelle/HOL and SMT

2. From Isabelle/HOL to SMT ...
   - Supported SMT Solvers
   - Preprocessing

3. ... and back again
   - Z3 Proofs
   - Proof Reconstruction for Z3
   - Evaluation

4. Conclusion
Isabelle – A Generic Theorem Prover

- theorems: abstract type
- inference rules: intuitionistic higher-order logic
- terms, types, …
Isabelle – A Generic Theorem Prover

- theories
- proof tools
- object logic

Kernel

- theorems: abstract type
- inference rules: intuitionistic
- higher-order logic

Infra-structure

- terms, types, …
Isabelle’s Meta-Logic

Terms:
- constants ($\land$, $\rightarrow$, $\equiv$)
- variables
- $\lambda$-abstraction
- application

Theorems: $H \vdash P$

Rules:
- assumption
- introduction and elimination of $\land$ and $\rightarrow$ and $\equiv$
- reflexivity, symmetry, transitivity, congruence
- generalization, instantiation
- higher-order resolution
Isabelle/HOL – Higher-Order Logic in Isabelle

- theories
- proof tools
- object logic

Kernel

- theorems: abstract type
- inference rules: intuitionistic higher-order logic
- terms, types, ...
Isabelle/HOL – Higher-Order Logic in Isabelle

- **theories**
- **proof tools**

**HOL**
- shallow embedding in meta logic
- usual connectives and functions

**Kernel**
- theorems: abstract type
- inference rules: intuitionistic higher-order logic

**Infra-structure**
- terms, types, ...
Isabelle/HOL – Higher-Order Logic in Isabelle

- term rewriting
- tableaux prover
- arithmetic
- shallow embedding in meta logic
- usual connectives and functions
- theorems: abstract type
- inference rules: intuitionistic higher-order logic
- terms, types, . . .
Satisfiability Modulo Theories (SMT)

Many-sorted first-order logic

Theories:

- equality and uninterpreted functions
- linear (integer/real) arithmetic
- arrays
- bitvectors
- algebraic datatypes

Combination: in general undecidable with high complexity
- necessary fragment still successful: program verification, model checking, …

SMT solvers: CVC3, Yices, Z3, …
Isabelle/HOL and SMT

Observation: many essentially first-order propositions:
- Sledgehammer: connection to first-order provers

With SMT:
- built-in support for additional theories (e.g., linear arithmetic)
- weaker on quantifiers

SMT cannot (directly) deal with:
- polymorphism
- λ-abstractions
- induction
Isabelle/HOL and SMT

Observation: many essentially first-order propositions:

- Sledgehammer: connection to first-order provers

With SMT:

- built-in support for additional theories (e.g. linear arithmetic)
- weaker on quantifiers

SMT cannot (directly) deal with:

- polymorphism: monomorphization, encoding of types in terms
- λ-abstractions
- induction
Isabelle/HOL and SMT

Observation: many essentially first-order propositions:
- Sledgehammer: connection to first-order provers

With SMT:
- built-in support for additional theories (e.g. linear arithmetic)
- weaker on quantifiers

SMT cannot (directly) deal with:
- polymorphism: monomorphization, encoding of types in terms
- \(\lambda\)-abstractions: combinatory logic (SKI), lifting
- induction
Isabelle/HOL and SMT

Observation: many essentially first-order propositions:
- Sledgehammer: connection to first-order provers

With SMT:
- built-in support for additional theories (e.g. linear arithmetic)
- weaker on quantifiers

SMT cannot (directly) deal with:
- polymorphism: monomorphization, encoding of types in terms
- $\lambda$-abstractions: combinatory logic (SKI), lifting
- induction (but partial unfolding of recursive functions)
Introduction

From Isabelle/HOL to SMT ...

... and back again

Conclusion

generic interface

goal

preprocessing

proof

counterexample

specific interface

interface-specific information

serialization

proof reconstruction

SMT solver

oracle

unsat

sat

unknown
Supported SMT Solvers and Formats

Generic approach:
- low effort to integrate new solvers

SMT-LIB format:
- supported by practically all available solvers
- separates terms and formulas
- fixed logics (combination of theories)
- no polymorphism

Z3 low-level format:
- no separation between terms and formulas
- supports all theories and any combination
- restricted polymorphism
generic interface

- goal
- preprocessing

specific interface

- interface-specific information
- serialization
- proof reconstruction
- oracle

- SMT solver

proof

counterexample

unsat

sat

unknown
Preprocessing

SMT:
- requires transformations of essentially first-order HOL terms
SMT:

- requires transformations of essentially first-order HOL terms
SMT:
- requires transformations of essentially first-order HOL terms

Rewriting of theorems (normalization):
- establish properties necessary for serialization and proof reconstruction
Preprocessing

SMT:
- requires transformations of essentially first-order HOL terms

Rewriting of theorems (normalization):
- establish properties necessary for serialization and proof reconstruction

Term transformations (decoration):
- prepare only serialization
- can use "dirty" tricks
- faster/simpler than theorem rewriting
Rewriting of Theorems (Normalization)

- Negative numerals: rewrite into negated positive numerals
Rewriting of Theorems (Normalization)

- Negative numerals: rewrite into negated positive numerals
- Natural numbers: embed into integers
  - add axiomatization of \textit{nat} and \textit{int}

Example

\[ P (2 + x) \rightsquigarrow P (\text{nat} (2 + \text{int} x)) \]
Rewriting of Theorems (Normalization)

- Negative numerals: rewrite into negated positive numerals
- Natural numbers: embed into integers
  - add axiomatization of \textit{nat} and \textit{int}

Example

\[
P (2 + x) \leadsto P (\text{nat} (2 + \text{int} \ x))
\]

- Lambda terms: lift

Example

\[
\text{map} (\lambda x. x + 1) [1, 2] = [2, 3] \leadsto \begin{cases} \\
\forall x. f \ x = x + 1 \\
\text{map} \ f \ [1, 2] = [2, 3]
\end{cases}
\]
Rewriting of Theorems (Normalization)

- Negative numerals: rewrite into negated positive numerals
- Natural numbers: embed into integers
  - add axiomatization of \textit{nat} and \textit{int}

\textbf{Example}

\[
P (2 + x) \rightsquigarrow P (\text{nat} (2 + \text{int} \ x))
\]

- Lambda terms: lift

\textbf{Example}

\[
\text{map} (\lambda x. x + 1) [1, 2] = [2, 3] \rightsquigarrow \begin{cases} 
\forall x. f \ x = x + 1 \\
\text{map} \ f \ [1, 2] = [2, 3]
\end{cases}
\]

- Axiomatization for \textit{abs}, \textit{min}, \textit{max}, and pairs
Term Transformations (Decoration)

Monomorphization:

- compute necessary instances of polymorphic constants
- copy and instantiate polymorphic assumptions
- enforce termination: upper limit on generated copies
- simple, but can cause blow-up of formulas
Term Transformations (Decoration)

Monomorphization:
- compute necessary instances of polymorphic constants
- copy and instantiate polymorphic assumptions
- enforce termination: upper limit on generated copies
- simple, but can cause blow-up of formulas

Identification of built-in symbols
Term Transformations (Decoration)

Monomorphization:
- compute necessary instances of polymorphic constants
- copy and instantiate polymorphic assumptions
- enforce termination: upper limit on generated copies
- simple, but can cause blow-up of formulas

Identification of built-in symbols

Separation between formulas and terms:
- insert marker symbol
- add axiomatization for term-level occurrences of $\land$, $\lor$, $\leq$, $\ldots$
Term Transformations (Decoration)

Monomorphization:
- compute necessary instances of polymorphic constants
- copy and instantiate polymorphic assumptions
- enforce termination: upper limit on generated copies
- simple, but can cause blow-up of formulas

Identification of built-in symbols

Separation between formulas and terms:
- insert marker symbol
- add axiomatization for term-level occurrences of $\land$, $\lor$, $\leq$, ...  

Transformation of partially-applied functions:
- additional symbol: make application explicit
generic interface

- goal
  - preprocessing

specific interface

- interface-specific information
  - serialization
    - proof reconstruction
      - proof
        - counterexample
  - SMT solver
    - oracle
      - unsat
      - sat
      - unknown
Introduction

From Isabelle/HOL to SMT ...

... and back again

Conclusion

generic interface

- goal
- preprocessing
- proof
- counterexample

specific interface

- interface-specific information
- serialization
- proof reconstruction
- oracle

SMT solver

unsat

sat

unknown
Outline
- Introduction
- From Isabelle/HOL to SMT ...
- ... and back again
- Conclusion

**generic interface**
- goal
- preprocessing

**specific interface**
- interface-specific information
- serialization
- SMT solver
- proof reconstruction
- oracle
- unsat
- sat
- unknown

**counterexample**
- proof
Z3 Terms

Signature:
- types: basic types ($\text{int}$, $\text{real}$) and user-defined types (nullary type constructors)
- function symbols: fixed arity, no polymorphism

Terms:
- variables: $x$, $y$
- applications: $f \; t_1 \ldots t_n$
- quantifiers (triggers are ignored)

Formulas (terms of sort $\text{bool}$): $P$, $Q$

Natural mapping into HOL term structure
Equisatisfiability

Example

\((\neg x \lor \text{false}) \sim (\neg y)\)

Semantics: existential closure

Example

\((\exists x. \neg x \lor \text{false}) \leftrightarrow (\exists y. \neg y)\)

Representation in HOL:

- equivalence without existential closure
- exception: Skolemization
Natural deduction style:

Example

\[
\begin{array}{c}
\neg \text{true} \vdash \neg \text{true} \\
\hline
\neg \text{true} \vdash \neg \text{true} \leftrightarrow \text{false} \\
\hline
\neg \text{true} \vdash \text{false}
\end{array}
\]
Natural deduction style:

Example

\[
\neg \text{true} \vdash \neg \text{true} \quad \text{asserted} \quad \vdash \neg \text{true} \leftrightarrow \text{false} \quad \text{rewrite} \\
\neg \text{true} \vdash \text{false} \quad \text{mp} \leftrightarrow
\]

28 proof rules:

- 14 core rules
- 7 quantifier rules
- 5 equality rules
- 2 theory rules
Proof Reconstruction

Follows the proof structure:

\[
\begin{align*}
\neg \text{true} & \vdash \neg \text{true} \\
\vdash \neg \text{true} & \leftrightarrow \text{false} \\
\neg \text{true} & \vdash \text{false}
\end{align*}
\]
Proof Reconstruction

Follows the proof structure:

- bottom-up
- one method for every rule

\[ \neg \text{true} \vdash \neg \text{true} \]

\[ \vdash \neg \text{true} \leftrightarrow \text{false} \]

\[ \neg \text{true} \vdash \text{false} \]
Proof Reconstruction

Follows the proof structure:

- bottom-up
- one method for every rule

\[
\neg \text{true} \vdash \neg \text{true} \\
\neg \text{true} \vdash \neg \text{true} \leftrightarrow \text{false} \\
\neg \text{true} \vdash \text{false}
\]
Proof Reconstruction

Follows the proof structure:

- bottom-up
- one method for every rule

\[
\neg \text{true} \vdash \neg \text{true} \quad \vdash \neg \text{true} \leftrightarrow \text{false} \quad \text{mp}\leftrightarrow
\]

\[
\neg \text{true} \vdash \text{false}
\]
Proof Reconstruction

Follows the proof structure:

- bottom-up
- one method for every rule
- all inferences certified by Isabelle kernel

\[
\neg \text{true} \vdash \neg \text{true} \quad \text{asserted}
\]

\[
\vdash \neg \text{true} \leftrightarrow \text{false} \quad \text{rewrite}
\]

\[
\neg \text{true} \vdash \text{false} \quad \text{mp}
\]
Proof Reconstruction

Follows the proof structure:

- bottom-up
- one method for every rule
- all inferences certified by Isabelle kernel
- global check at the end

\[
\neg \text{true} \vdash \neg \text{true} \quad \text{asserted} \\
\vdash \neg \text{true} \leftrightarrow \text{false} \quad \text{rewrite} \\
\neg \text{true} \vdash \text{false} \quad \text{mp} \leftrightarrow
\]
Proof Reconstruction

Follows the proof structure:

- bottom-up
- one method for every rule
- all inferences certified by Isabelle kernel
- global check at the end
- local checks for debugging

\[
\neg \text{true} \vdash \neg \text{true} \quad \text{asserted}\quad \neg \text{true} \leftrightarrow \text{false} \quad \text{rewrite}\quad \text{mp}\leftrightarrow
\]

\[
\neg \text{true} \vdash \neg \text{true} \leftrightarrow \text{false} \quad \text{mp}\leftrightarrow
\]

\[
\neg \text{true} \vdash \text{false} \quad \text{mp}\leftrightarrow
\]

\[
\neg \text{true} \vdash \text{false}
\]
Reconstruction Methods
Reconstruction Methods

- Direct representation or basic inference rule: (3 rules)

<table>
<thead>
<tr>
<th>Examples</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>⊢ true-prop</td>
<td>asserted</td>
</tr>
<tr>
<td>⊢ true</td>
<td>P ⊢ P</td>
</tr>
</tbody>
</table>
Reconstruction Methods

• Direct representation or basic inference rule:

Examples

\[ \vdash \text{true-prop} \quad \vdash P \rightarrow P \]

(3 rules)

• Theorem or inference rule, and resolution:

Example

\[ \Gamma_1 \vdash P_1 \quad \Gamma_2 \vdash P_1 \leftrightarrow P_2 \quad \Gamma_1 \cup \Gamma_2 \vdash \text{mp} \]

in Isabelle: \[ P_1 \rightarrow P_1 \leftrightarrow P_2 \rightarrow P_2 \]
Reconstruction Methods

- Direct representation or basic inference rule: (3 rules)

  Examples
  \[
  \begin{align*}
  \vdash & \text{true-prop} \quad \vdash & P \vdash P
  \end{align*}
  \]

- Theorem or inference rule, and resolution: (9 rules)

  Example
  \[
  \begin{align*}
  \Gamma_1 & \vdash P_1 \quad \Gamma_2 & \vdash P_1 \leftrightarrow P_2 \\
  \Gamma_1 \cup \Gamma_2 & \vdash P_2 \quad \text{mp}\leftrightarrow
  \end{align*}
  \]

  in Isabelle: 
  \[
  P_1 \Rightarrow P_1 \leftrightarrow P_2 \Rightarrow P_2
  \]

- Isabelle proof tools (7 rules)
Reconstruction Methods: The Remaining 9 Rules

Special treatment due to:
- no available proof tools
- optimizations for central proof rules

Optimizations:
- meta-equality instead of HOL equality
- cheap inference rules of Isabelle kernel
- memoize intermediate steps
- reduce number of resolution steps, prepare suitable theorems
Unit Resolution

Example

\[
P_1 \lor \neg P_2 \lor \neg P_3 \quad P_2
\]

\[
P_1 \lor P_2 \lor \neg P_3 \equiv P_1 \lor \neg P_3
\]
Unit Resolution

Example

\[
P_1 \lor (\neg P_2 \lor \neg P_3) \quad P_2
\]

\[
P_1 \lor \neg P_3
\]
Unit Resolution

Example

\[
P_1 \lor (\neg P_2 \lor \neg P_3) \quad P_2
\]

\[
P_1 \lor \neg P_3
\]

Idea: combine resolution with rewriting

Example with rewriting

\[
P_1 \lor (\neg P_2 \lor \neg P_3) \quad P_2
\]

\[
P_1 \lor (\neg P_2 \lor \neg P_3) \equiv P_1 \lor \neg P_3
\]

\[
P_1 \lor \neg P_3
\]
Unit Resolution

\[ P_1 \lor \neg P_2 \lor \neg P_3 \equiv P_1 \lor \neg P_3 \]
Unit Resolution

\[ P_1 \equiv P_1 \]

\[
P_1 \lor \neg P_2 \lor \neg P_3 \equiv P_1 \lor \neg P_3
\]
Unit Resolution

\[ P_1 \equiv P_1 \quad \quad \quad \quad \neg P_2 \lor \neg P_3 \equiv \neg P_3 \]

\[ P_1 \lor \neg P_2 \lor \neg P_3 \equiv P_1 \lor \neg P_3 \]
Unit Resolution

\[\overline{P_2} \quad \neg P_2 \lor \neg P_3 \equiv \neg P_3\]
\[P_1 \equiv P_1 \quad \neg P_2 \lor \neg P_3 \equiv \neg P_3\]
\[P_1 \lor \neg P_2 \lor \neg P_3 \equiv P_1 \lor \neg P_3\]
Unit Resolution

\[ P_2 \]

\[ \neg P_2 \lor \neg P_3 \equiv \neg P_3 \quad E_1 \]

\[ P_1 \equiv P_1 \]

\[ \neg P_2 \lor \neg P_3 \equiv \neg P_3 \]

\[ P_1 \lor \neg P_2 \lor \neg P_3 \equiv P_1 \lor \neg P_3 \]

\[ E_1 : \]

\[ P_2 \quad Q_1 \iff \neg Q_1 \lor Q_2 \equiv Q_2 \]

\[ \neg P_2 \lor Q_2 \equiv Q_2 \]
Unit Resolution

\[
P_1 \equiv P_1
\]
\[
\frac{P_2}{\neg P_2 \lor \neg P_3 \equiv \neg P_3}
\]
\[
E_1
\]
\[
\frac{\neg P_3 \equiv \neg P_3}{\neg P_2 \lor \neg P_3 \equiv \neg P_3}
\]
\[
P_1 \lor \neg P_2 \lor \neg P_3 \equiv P_1 \lor \neg P_3
\]

\[
E_1 : \quad P_2 \quad Q_1 \iff \neg Q_1 \lor Q_2 \equiv Q_2
\]
\[
\frac{\neg P_2 \lor Q_2 \equiv Q_2}{\neg P_2 \lor Q_2 \equiv Q_2}
\]
Natural choice: use Isabelle’s simplifier

But: custom-made procedure provides much better performance

Idea: combine reflexivity and congruence of basic inference rules

Example

\[
\begin{align*}
  f & \equiv f  \\
  a & \equiv b  \\
  f \ a & \equiv f \ b  \\
  c & \equiv c  \\
  f \ a \ c & \equiv f \ b \ c  \\
  d & \equiv e  \\
  f \ a \ c \ d & \equiv f \ b \ c \ e
\end{align*}
\]
Memoization for Conjunction Elimination

**Example**

\[
P_1 \land P_2 \land P_3 \quad \Rightarrow \quad P_2
\]

Similar: conclude \( P_1 \) or \( P_3 \)

Idea:
1. explode \( P_1 \land P_2 \land P_3 \) once into literals
2. memoize literals
3. pick required literal on demand

Dually for negated disjunction elimination
Skolemization

Example

\[ \vdash (\exists x. P \, x \, y) \sim P (f \, y) \, y \]
Skolemization

Example

\[ \vdash (\exists x. P x y) \sim P (f y) y \]

With Hilbert choice operator $\varepsilon$

\[ f \equiv (\lambda y. \varepsilon x. P x y) \vdash (\exists x. P x y) \iff P (f y) y \]
Skolemization

Example

\[ \vdash (\exists x. P \ x \ y) \sim P (f \ y) \ y \]

With Hilbert choice operator \( \varepsilon \)

\[ f \equiv (\lambda y. \varepsilon x. P \ x \ y) \vdash (\exists x. P \ x \ y) \leftrightarrow P (f \ y) \ y \]

At the end of reconstruction:

\[ \Gamma, f \equiv (\lambda y. \varepsilon x. P \ x \ y) \vdash false \]
Skolemization

Example

\[ \vdash (\exists x. P \times y) \sim P (f \times y) \times y \]

With Hilbert choice operator \( \varepsilon \)

\[ f \equiv (\lambda y. \varepsilon x. P \times y) \vdash (\exists x. P \times y) \leftrightarrow P (f \times y) \times y \]

At the end of reconstruction:

\[ \Gamma, f \equiv (\lambda y. \varepsilon x. P \times y) \vdash \text{false} \]

\[ \Gamma \vdash f \equiv (\lambda y. \varepsilon x. P \times y) \Rightarrow \text{false} \]
Skolemization

Example

\[ \vdash (\exists x. P \times y) \sim P (f \times y) y \]

With Hilbert choice operator \( \varepsilon \)

\[ f \equiv (\lambda y. \varepsilon x. P \times y) \vdash (\exists x. P \times y) \iff P (f \times y) y \]

At the end of reconstruction:

\[ \Gamma, f \equiv (\lambda y. \varepsilon x. P \times y) \vdash false \]

\[ \Gamma \vdash f \equiv (\lambda y. \varepsilon x. P \times y) \Rightarrow false \]

\[ \Gamma \vdash (\lambda y. \varepsilon x. P \times y) \equiv (\lambda y. \varepsilon x. P \times y) \Rightarrow false \]
Skolemization

Example

\[ \vdash (\exists x. P \times y) \sim P (f \ y) \ y \]

With Hilbert choice operator \( \varepsilon \)

\[ f \equiv (\lambda y. \varepsilon x. P \times y) \vdash (\exists x. P \times y) \leftrightarrow P (f \ y) \ y \]

At the end of reconstruction:

\[ \Gamma, f \equiv (\lambda y. \varepsilon x. P \times y) \vdash \text{false} \]

\[ \Gamma \vdash f \equiv (\lambda y. \varepsilon x. P \times y) \implies \text{false} \]

\[ \Gamma \vdash (\lambda y. \varepsilon x. P \times y) \equiv (\lambda y. \varepsilon x. P \times y) \implies \text{false} \]

\[ \Gamma \vdash \text{false} \]
"The head function symbol of the left-hand side is interpreted."

Examples

\[ P_1 \land P_2 \land true = P_2 \land P_1 \]
\[ (x < y) = (y + (-1 \times x) > 0) \]

Several possible simplification steps:

- ACI rewriting of \( \land \) and \( \lor \)
- AC rewriting of non-idempotent functions (e.g. \( + \))
- arithmetic: polynomial normal-form
- array: application of access/update-rules
- quantifier elimination: \( (\exists x. 1 \leq x \land x < y) = (1 < y) \)
Approach 1: try

1. identified simplication rules
2. custom-made ACI rewriting for $\land$ and $\lor$
3. simplifier (arrays) and arithmetic decision procedures

Approach 2:

- choose the appropriate method
- based on the head symbol of the left-hand side

Overall difference negligible:

- Isabelle’s arithmetic DPs take much longer
Recurrence relation $x_{i+2} = |x_{i+1}| - x_i$ has period 9:
- with Isabelle’s arithmetic: 4 minutes
- with Z3: 15 seconds

SMT-LIB benchmarks:
- industrial problems: huge formulas
- Z3 proofs: around 100KB, up to several MB
- reconstruction: around 20 times slower than proof finding
Some Quirks in Z3’s Proof Generation

\[ \vdash P \land (\forall x : \text{int.} \ x > 0) \leftrightarrow false \land P \]
Some Quirks in Z3’s Proof Generation

\[ \vdash P \land \left( \forall x : \text{int} . \ x > 0 \right) \leftrightarrow \text{false} \land P \quad \text{rewrite} \]

\[ \Gamma_1 \vdash P_1 \lor P_2 \lor P_1 \quad \Gamma_2 \vdash \neg P_2 \]
\[ \Gamma_1 \cup \Gamma_2 \vdash P_1 \quad \text{unit} \]
Some Quirks in Z3’s Proof Generation

\[
\vdash P \land (\forall x : \text{int.} \ x > 0) \iff \text{false} \land P
\]

\[
\frac{}{\Gamma_1 \vdash P_1 \lor P_2 \lor P_1 \quad \Gamma_2 \vdash \neg P_2} \quad \text{unit}
\]

\[
\frac{}{\Gamma_1 \cup \Gamma_2 \vdash P_1}
\]

\[
\frac{}{\Gamma_1 \vdash s = t \quad \Gamma_2 \vdash u = t} \quad \text{trans}
\]

\[
\frac{}{\Gamma_1 \cup \Gamma_2 \vdash s = u}
\]
Some Quirks in Z3’s Proof Generation

\[ \vdash P \land (\forall x : \text{int.} . x > 0) \iff false \land P \text{ rewrite} \]

\[ \Gamma_1 \vdash P_1 \lor P_2 \lor P_1 \quad \Gamma_2 \vdash \neg P_2 \]
\[ \Gamma_1 \cup \Gamma_2 \vdash P_1 \text{ unit} \]

\[ \Gamma_1 \vdash s = t \quad \Gamma_2 \vdash u = t \]
\[ \Gamma_1 \cup \Gamma_2 \vdash s = u \text{ trans} \]

\[ f \ x = 1 + x + g \ x \text{ rewrite}* \]
Generic connection of SMT solvers with Isabelle/HOL:
- can solve many essentially first-order formulas
- can cope (to some extent) with polymorphism, λ-expressions, and recursive functions

Proof reconstruction for Z3:
- certifying connection of Z3 with Isabelle/HOL
- several optimizations
- helped to improve Z3 proof generation