Isabelle/HOL: Selected Features and Recent Improvements

Tjark Weber
webertj@in.tum.de

Security of Systems Group, Radboud University Nijmegen
February 20, 2007

1 Introduction

2 Core Features
- Logic
- User Interface
- Proof Language
- Automation
- Correctness
- Existing Libraries

3 Selected Extensions

4 Conclusion

Motivation

Errare humanum est.

Cicero

Motivation

Complex systems almost inevitably contain bugs.

Motivation

Complex systems almost inevitably contain bugs.

> *Program testing can be used to show the presence of bugs, but never to show their absence!*
>
> Edsger W. Dijkstra

Motivation

Theorem Provers

A theorem prover is a computer program to prove theorems.

Theorem Provers

A theorem prover is a computer program to prove theorems.

Given a precise description of a system, and a formal specification of its intended behavior,

Theorem Provers

A theorem prover is a computer program to prove theorems.

Given a precise description of a system, and a formal specification of its intended behavior, we obtain a computer-checked proof that the system meets its specification.

Theorem Provers: Characteristic Features

- Logic
- User interface
- Proof language
- Automation
- Correctness
- Existing libraries
- ...

Isabelle

Isabelle is a generic interactive theorem prover, developed by Larry Paulson at Cambridge University and Tobias Nipkow at Technische Universität München. First release in 1986.

Isabelle's Logics

Isabelle implements a meta-logic, Isabelle/Pure, based on the simply-typed λ-calculus.

This meta-logic serves as the basis for several different object logics:

- ZF set theory (Isabelle/ZF)
- First-order logic (Isabelle/FOL)
- Higher-order logic (Isabelle/HOL)
- Modal logics
- ...

Isabelle/HOL

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

Isabelle/HOL

Isabelle/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: $t_\sigma ::= x_\sigma \mid (t_{\sigma' \to \sigma}\ t_{\sigma'})_\sigma \mid (\lambda x_{\sigma_1}.\ t_{\sigma_2})_{\sigma_1 \to \sigma_2}$

Isabelle/HOL

Isabelle/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:

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

Isabelle/HOL

Isabelle/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

Standard set-theoretic semantics:

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

The Semantics of HOL

Standard set-theoretic semantics:

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

Semantics of types:

- $[\![\mathbb{B}]\!] = \{\top, \bot\}$
- $[\![\alpha]\!]$ is given by the model
- $[\![\sigma_1 \rightarrow \sigma_2]\!] = [\![\sigma_2]\!]^{[\![\sigma_1]\!]}$

Features of Isabelle/HOL

Isabelle/HOL also provides ...

- axiomatic type classes
- type and constant definitions
- recursive datatypes
- recursive functions
- inductively defined sets

Isabelle/HOL is both a specification logic and a programming language.

User Interface

Isabelle uses Proof General as its default interface.

- (X-)Emacs-based
- Proof script management
- Mathematical symbols
- Theory dependency graph
- ...

A batch mode is available as well:

```
$ isatool usedir ...
```

Proof Language

Many interactive theorem provers (including Isabelle) offer a tactic-style proof language.

The prover maintains a proof state which keeps track of the formulae that remain to be shown.

Tactics (which can implement simple natural-deduction rules or complex decision procedures) are applied to the proof state to simplify remaining proof goals.

A proof is a sequence of tactic applications.

Tactic-Style Proofs: Example

lemma "$(\exists x. \forall y.\ P\ x\ y) \Longrightarrow (\forall y.\ \exists x.\ P\ x\ y)$"
  apply (erule *exE*)
  apply (rule *allI*)
  apply (erule_tac *x*="*y*" in *allE*)
  apply (rule_tac *x*="*x*" in *exI*)
  apply assumption
done

Tactic-Style Proofs: Shortcomings

Tactic-style proofs have little in common with traditional mathematical proofs.

Tactic-style proofs are impossible to understand independently of the theorem prover.

Tactic-style proofs are hard to maintain.

Isabelle/Isar: Readable Proofs

Isabelle/Isar is a structured proof language, where proofs resemble those found in mathematical textbooks.

lemma "$(\exists x.\, \forall y.\, P\, x\, y) \Longrightarrow (\forall y.\, \exists x.\, P\, x\, y)$"
proof
  assume "$\exists x.\, \forall y.\, P\, x\, y$"
  from *this* obtain x where X: "$\forall y.\, P\, x\, y$" ..
  fix y
  from X have "$P\, x\, y$" ..
  then show "$\exists x.\, P\, x\, y$" ..
qed

Isar proofs can be understood independently of the theorem prover.

Automation

lemma "$(\exists x.\, \forall y.\, P\, x\, y) \Longrightarrow (\forall y.\, \exists x.\, P\, x\, y)$"
proof
  assume "$\exists x.\, \forall y.\, P\, x\, y$"
  from *this* obtain x where X: "$\forall y.\, P\, x\, y$" ..
  fix y
  from X have "$P\, x\, y$" ..
  then show "$\exists x.\, P\, x\, y$" ..
qed

Automation

lemma "$(\exists x.\, \forall y.\, P\, x\, y) \Longrightarrow (\forall y.\, \exists x.\, P\, x\, y)$"
by auto

Isabelle's Automatic Tactics

Isabelle provides several automatic tactics and decision procedures.

- term rewriting (auto, simp)
- tableau-based (blast)
- Fourier-Motzkin (arith)
- Cooper's quantifier elimination (presburger)
- ...

These can easily be instantiated for different object logics.

Correctness

Quis custodiet ipsos custodes?

Juvenal

LCF-Style Systems

Theorems are implemented as an abstract datatype. They can be constructed only in a very controlled manner, through calling functions from the system's kernel.

There is one such kernel function for each inference rule of the system's logic.

Advanced tactics and decision procedures are built on top of the kernel. They must use the kernel functions (or other, more basic tactics) to create theorems.

Existing Libraries

Substantial amounts of mathematics and computer science have been formalized.

- Algebra
- Calculus
- Graph theory
- ...

- Automata theory
- Programming language semantics, program logics
- ...

Isabelle/HOL alone contains over 7,300 definitions and theorems.

Some Recent Projects

- Verified Java Bytecode Verification (Gerwin Klein, PhD thesis, 2003)
- Verisoft: the pervasive formal verification of computer systems (since 2003)
- Flyspeck: a formal proof of the Kepler conjecture (Thomas C. Hales, since 2003)

Document Preparation

Isabelle generates LaTeX sources from theory files. Other output modes (e.g. HTML) are available as well.

Theory files can contain comments and annotations without logical significance. Antiquotations in these comments refer to Isabelle theorems, terms etc.

You can use one tool to prove your theorems and write your paper!

Integration of External Provers

Motivation:

- Increased Automation
- Improved Performance

In Tools We Trust?

The Oracle Approach

A formula is *accepted* as a theorem if the external tool claims it to be provable.

In Tools We Trust?

The Oracle Approach

A formula is *accepted* as a theorem if the external tool claims it to be provable.

The LCF-style Approach

The external tool provides a *certificate* of its answer that is translated into an Isabelle proof.

System Overview

Isabelle

Input formula

Preprocessing

Counterexample

Theorem

Proof reconstruction

DIMACS
SMT–LIB
TPTP

Model

Certificate

External Prover

provable?

no

yes

Recent Prover Integrations

- SAT: zChaff, MiniSAT (Tjark Weber, Alwen Tiu et al.)
- SMT: haRVey (Pascal Fontaine, Stephan Merz et al.)
- FOL: Spass, Vampire (Jia Meng, Larry Paulson)
- HOL: HOL 4, HOL-Light (Sebastian Skalberg, Steven Obua)

Proof Objects

Proof objects constitute certificates that are easily verifiable by an external proof checker.

Proof objects in Isabelle (implemented by Stefan Berghofer) use λ-terms, based on the Curry-Howard isomorphism. Information that can be inferred is omitted to reduce the size of the proof object.

Applications: proof-carrying code, proof export, program extraction from proofs

For programs extracted from proofs, a realizability statement is automatically proven to establish the program's correctness.

Code Generation

Motivation: executing formal specifications, rapid prototyping, program extraction, reflection

ML and Haskell code can be generated from executable HOL terms (Stefan Berghofer, Florian Haftmann). Such terms are built from executable constants, datatypes, inductive relations and recursive functions.

Translating inductive relations requires a mode analysis.

Normalization by evaluation: simplifies executable terms (possibly containing free variables) by evaluating them symbolically. Orders of magnitude faster than rewriting.

Reflection

Instead of implementing a decision procedure in ML, we write it in the logic (Isabelle/HOL)—as a recursive function on an appropriately defined datatype representing terms—, prove its correctness, and generate code from the definition.

Motivation: Assuming that the code generator is part of the trusted kernel, the generated code can simply be executed, with no need for proof checking. This can lead to a significant speed-up.

Amine Chaieb has implemented a reflected version of Cooper's quantifier elimination procedure for Presburger arithmetic and a generic reflection interface for user-defined decision procedures.

Disproving Non-Theorems

Traditionally, the focus in theorem proving has been on, well, proving theorems.

However, disproving a faulty conjecture can be just as important. In formal verification, initial conjectures are more often false than not, and a counterexample often exhibits a fault in the implementation.

Isabelle provides two essentially different tools for disproving conjectures: quickcheck (by Stefan Berghofer) and refute (by Tjark Weber).

quickcheck

- Inspired by the Haskell `quickcheck` library (for testing Haskell programs).
- Free variables are instantiated with random values.
- The code generator is used to simplify the resulting term.
- Parameters: size of instantiations, number of iterations.
- Fast, but not suited for existential or non-executable statements.
- Implications with strong premises are problematic.

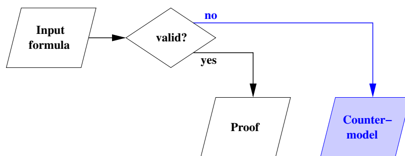
refute

- A SAT-based finite model generator.
- A HOL formula is translated into a propositional formula that is satisfiable iff the HOL formula has a model of a given size.
- Parameters: (minimal, maximal) size of the model.
- Any statement can be handled, but non-elementary complexity.
- Still useful in practice ("small model property").

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

Conclusion

- Isabelle is a powerful interactive theorem prover.
- A reasonably high degree of automation is available through both internal and external tools.
- Definitional packages and existing libraries facilitate the development of new specifications.
- A human-readable proof language, existing lemmas and a decent user interface facilitate the development of proofs.
- Several side applications: proof import/export, code extraction, counterexamples for unprovable formulae

Questions?

Beware of bugs in the above code; I have only proved it correct, not tried it.

Donald Knuth