Isabelle - a Tutorial

Gábor Alagi

June 28, 2011

Automated Reasoning Systems 2011S

Introduction

- Isabelle is a generic proof assistant tool
- Used for mathematical proofs and formal verification
  - Applied by Hewlett-Packard in the design of the HP 9000 line of servers
  - First formal proof of functional correctness of a general-purpose operating system kernel (seL4 microkernel, 2009)
- A successor of the HOL (Higher Order Logic)
- Developed by the following universities:
  - University of Cambridge
  - Technische Universität München
  - Université Paris-Sud
- http://isabelle.in.tum.de/

Overview

- LCF approach (Logic for Computable Functions) (1972)
- Written in SML (Standard ML)
- Provides a generic framework to encode object logic (a weak type-theory)
- Different object logics exist
  - FOL
  - HOL
  - ZFC
- Main proof method: a higher-order resolution
- Other tools are available
  - Term rewriting engine
  - Tableaux prover
  - Several decision procedures

Installation

- Obtainable on the webpage under *Download*

  http://isabelle.in.tum.de/download.html
- Sufficient instructions are available
- Supported OSs:
  - Linux
  - Mac OS X
  - Windows (only Cygwin)
- This tutorial was made with the Cygwin-version
  - `startxwin`
  - `Isabelle2011/bin/isabelle emacs -p xemacs`

Interface - Proof General

- The default interface of Isabelle
- XEmacs-based

((Show us around))

- Theory files: `.thy`
- Proof-General $>$ Options $>$ X-Symbol
- Isabelle $>$
- Next, Undo, Use, Goto, Retract
- Find
- Windows: goals, response, isabelle, trace

Basics

- meta language
- object language `" . . . "`
  - types, functions, terms, equations, formulas
- structured proof language (Isar)
- `theory`, `import`
- `lemma`
- The proofs are produced step-by-step
- Strong automated tools
  - Simplifier
  - Classic reasoner

Types

- Base types
  - `nat`
  - `bool`
- Function types: `a => b`
- Datatype

```
datatype ('a, 'b) mydlist =
  DNIL | DCONS 'a 'b "('a, 'b) mydlist"
```

- Several more...
- `size` function
- Explicit notation e.g. `x::nat`

Functions

- Isabelle handles only total functions
  - Termination is important
  - Advanced features support a limited partiality as well
- `primrec`, `fun`, `function`
- The definition is given by equations
- Standard functional expressions are available
  - if-then-else
  - let-expressions
  - case-expressions
  - λ-abstractions

Example - Toytree

- Example: `toytree.thy`
- Proving a lemma: `apply`, `done`
- Some basic proof steps
  - `induct_tac` *`var`*
  - `case_tac`
  - `simp`
  - `auto`
- `defer`, `prefer` *`i`*
- `thm`

Simplifier

- What turns into rewriting rules?
  - datatype definition
  - function definition
  - definition (not automatically)
  - theorems and lemmas
    - `[simp]` option
    - `declare` *`theorem`* `[simp]`
    - `declare` *`theorem`* `[simp del]`
    - `apply(simp add/del/only:` *`theorems`*`)`
- Trace

Logic

- Natural deduction
- Backward/Goal-oriented proof search
- Subgoals

$$[\![A_1, \ldots, A_m]\!] \Longrightarrow G$$

- Rules

$$[\![P_1, \ldots, P_n]\!] \Longrightarrow Q$$

  P_1 is the so-called *major premise*
- Different rule-application tactics
  - `rule` (e.g. `conjI`)
  - `erule` (e.g. `exE`)
  - `drule` (e.g. `mp`)
  - `frule`
- `assumption`
- Automated provers: `auto`, `arithm`, `blast`

Example - Confectionery

- There is a certain *cake* called *rigójancsi*
- For every *cake* there is a *customer* who likes it
- Those *customers* who have diabetes don't like *rigójancsi*

Q: Is there a *customer*, who hasn't got diabetes?

Example - Euclid's algorithm

- `gcd(x,y) = if (y=0) then x`
  `else gcd(y,x mod y)`
- Lemma 1: gcd(x,y)|x $\wedge$ gcd(x,y)|y
- Lemma 2: k|x $\wedge$ k|y $\Longrightarrow$ k|gcd(x,y)
- Theorem: k|gcd(x,y) $\Longleftrightarrow$ k|x $\wedge$ k|y

Further...

Tools

- Generating LaTeXdocument
- Quickcheck
- Sledgehammer
- Code generation

Links

- Archive of formal proofs: http://afp.sourceforge.net/
- Projects using Isabelle: http://www.cl.cam.ac.uk/research/hvg/Isabelle/projects.html
- Tutorials: http://isabelle.in.tum.de/documentation.html