Getting Started With Isabelle

Lecture I: Tour

Lawrence C. Paulson
Computer Laboratory
What is Isabelle?

a generic proof assistant …
… based on a logical framework
a tool for mechanizing formalisms
a proof environment for its built-in logics:

- ZF set theory
- HOL (higher-order logic)
- many others including TLA & UNITY
**Generic Features — Available to Many Logics**

**Theories** declaring types, constants, etc. & inheriting from other theories

**Flexible syntax** including rewrite rules on abstract syntax trees

**Order-sorted polymorphism** to generalize results over related structures

**Simplifier** accepting conditional, permutative rewrite rules, ...

**Classical reasoner** to search for proofs using analytic rules
Features Specific to Isabelle/HOL

Proof libraries for integers, reals, lists, sets, cardinality, …

Worked examples in semantics, security, concurrency, non-standard analysis, …

Datatype definitions to model recursive types in functional programs

Inductive and Co-Inductive definitions to formalize semantics

Recursive functions defined over arbitrary well-founded relations

AND a link-up to SVC, the Stanford Validity Checker
Logical Reasoning in Isabelle/HOL

Goal "(ALL x. honest(x) & industrious(x) --> healthy(x)) & \\  ~ (EX x. grocer(x) & healthy(x)) & \\  (ALL x. industrious(x) & grocer(x) --> honest(x)) & \\  (ALL x. cyclist(x) --> industrious(x)) & \\  (ALL x. ~ healthy(x) & cyclist(x) --> ~ honest(x)) \ \\ --> (ALL x. grocer(x) --> ~ cyclist(x))";

The command \texttt{Goal} states the formula to be proved.

\texttt{by (Blast_tac 1)};

The command \texttt{by} applies a \texttt{tactic} to the subgoals.

- Proved in zero seconds!
- \texttt{Blast_tac} is a powerful, generic tableau prover.
Goal "(\(i: I \cdot A(i) \cap B(i)\)) = \(\cap\ A(i) \cap (\cap\ B(i))\); 

Here the goal is

\[
\left( \bigcap_{i \in I} A_i \cap B_i \right) = \left( \bigcap_{i \in I} A_i \right) \cap \left( \bigcap_{i \in I} B_i \right)
\]

by (Blast_tac 1);

- Blast_tac’s default rules cover set theory and much more!
- You can insert new default rules.
NatSum = Main +

consts sum :: "[nat=>nat, nat] => nat"

primrec
  "sum f 0 = 0"
  "sum f (Suc n) = f(n) + sum f n"

end

Theory NatSum extends Main, the standard parent. Constant sum is declared with a curried function type. Recursion equations make it a summation functional.
Arithmetic Reasoning II: Proving

\[ 1 + 3 + \cdots + (2n - 1) = n^2 \]

Goal "sum (%i. Suc(i+i)) n = n*n"
by (induct_tac "n" 1);

Level 1 (2 subgoals)
sum (%i. Suc (i + i)) n = n * n
1. sum (%i. Suc (i + i)) 0 = 0 * 0
2. !!n. sum (%i. Suc (i + i)) n = n * n
   ==> sum (%i. Suc (i + i)) (Suc n) = Suc n * Suc n

by Auto_tac;
qed "sum_of_odds";

The tactic induct_tac applies structural induction,
while Auto_tac simplifies and breaks up all subgoals.
**Datatypes I: Specifying Boolean Expressions**

\[
\text{datatype boolex} = \text{Const bool} \\
\quad \mid \text{Neg boolex} \\
\quad \mid \text{And boolex boolex}
\]

\text{consts} \quad \text{value} :: "boolex \Rightarrow \text{bool}"

\text{primrec}

"\text{value (Const b) = b}"

"\text{value (Neg b) = (\sim \text{value b})}"

"\text{value (And b c) = (value b \& value c)}"

Type \text{boolex} has three constructors.

Constant \text{value} maps these Boolean expressions to truth values.

It is declared \textit{primitive recursive}.
Datatypes II: Specifying If-Expressions

datatype ifex = CIF bool | IF ifex ifex ifex

consts valif :: "ifex => bool"
primrec
  "valif(CIF b) = b"
  "valif(IF b t e) = (if valif b then valif t else valif e)"

consts bool2if :: "boolex => ifex"
primrec
  "bool2if(Const b) = CIF b"
  "bool2if(Neg b) = IF (bool2if b) (CIF False) (CIF True)"
  "bool2if(And b c) = IF (bool2if b) (bool2if c) (CIF False)"

Functions valif and bool2if relate types ifex, bool and boolex.
Datatypes III: Proving \texttt{bool2if} Correct

Goal "\texttt{valif (bool2if b) = value b}";
by (induct_tac "b" 1);

Level 1 (3 subgoals)
valif (bool2if b) = value b
1. \texttt{!!bool. valif (bool2if (Const bool)) = value (Const bool)}
2. \texttt{!!boolex. valif (bool2if boolex) = value boolex} 
   \texttt{==> valif (bool2if (Neg boolex)) = value (Neg boolex)}
3. \texttt{!!boolex1 boolex2. [| valif (bool2if boolex1) = value boolex1; valif (bool2if boolex2) = value boolex2 |]}
   \texttt{==> valif (bool2if (And boolex1 boolex2)) = value (And boolex1 boolex2)}

by \texttt{Auto_tac};
Qsort = Sorting +
consts quickSort :: "'a::{linorder} list => 'a list"

recdef quickSort "measure size"
  simpset
   "simpset() addsimps [length_filter RS le_less_trans]"

  "quickSort [] = []"

  "quickSort (x#l) = quickSort [y:l. ~ x<=y] @
   (x # quickSort [y:l. x<=y])"

Parent theory Sorting defines sorted and multiset.
Function quickSort is recursive in the size of its argument.
Goal "multiset (quickSort xs) z = multiset xs z";
by (res_inst_tac ["u","xs"] quickSort.induct 1);

multiset (quickSort xs) z = multiset xs z
  1. multiset (quickSort []) z = multiset [] z
  2. !!x l.
      [ | multiset (quickSort (filter (op <= x) l)) z =
          multiset (filter (op <= x) l) z;
          multiset (quickSort [y:l . ~ x <= y]) z =
          multiset [y:l . ~ x <= y] z | ]
      ==> multiset (quickSort (x # l)) z =
          multiset (x # l) z
by Auto_tac;
Summary

Commands for managing an interactive proof:

- **Goal**: start it
- **by**: apply a tactic
- **qed**: name & store the proved theorem

Tactics for the reasoning itself

- **induct_tac i**: structural induction on subgoal $i$
- **Blast_tac i**: classical reasoning on subgoal $i$
- **Auto_tac**: tackle all subgoals

Theory file elements **consts, datatype, primrec, ...**