Getting Started With Isabelle

Lecture III: Interactive Proof

Lawrence C. Paulson
Computer Laboratory

UNIVERSITY OF
CAMBRIDGE

Lecture Outline

- Syntax of rules
- Proof states; subgoals
- Specialist tactics
- Primitive tactics
- Automatic tactics
- Simplification tactics
- The tableau prover (classical reasoner)

Expressing Inference Rules in Isabelle

`P & Q ==> P`

premises conclusion

`[| P-->Q; P |] ==> Q`

`(!!x. P x) ==> ALL x. P x`

general premise conclusion with HOL quantifier

`==>` and `!!` belong to the logical framework

An Isabelle Proof State

```
Goal "(i * j) * k = i * ((j * k)::nat)";
by (induct_tac "i" 1);

Level 1 (2 subgoals)
i * j * k = i * (j * k)
 1. 0 * j * k = 0 * (j * k)
 2. !!n. n * j * k = n * (j * k)
         ==> Suc n * j * k = Suc n * (j * k)
```

Subgoal 1 is the base case.

Subgoal 2 is the inductive step.

- The `!!n` names a natural number
- The `==>` separates the hypothesis and conclusion

The Form of a Subgoal

Each subgoal of a proof state looks like this:

$$\texttt{!!}x_1 \ldots\ x_k\texttt{.}\ \texttt{[|}\ \phi_1\texttt{;}\ \ldots\ \texttt{;}\ \phi_n\ \texttt{|]}\ \texttt{==>}\ \phi$$

parameters   assumptions   conclusion

- Parameters stand for arbitrary values
- Assumptions are typical of Natural Deduction

$\texttt{[|}\ \phi_1\texttt{;}\ \phi_2\ \texttt{|]}\ \texttt{==>}\ \psi$ is the same as

$\phi_1\ \texttt{==>}\ \texttt{(}\phi_2\ \texttt{==>}\ \psi\texttt{)}$

Specialist Tactics

`induct_tac "`x`"` i	induction over a datatype value x
`case_tac "`P`"` i	case analysis on property P
`subgoal_tac "`P`"` i	introduce P as a lemma
`Clarify_tac` i	perform all obvious steps

Replace subgoal i by new subgoals

May add new assumptions & parameters

Primitive Tactics: Single-Step Proof

Apply to subgoal i the $rule$

$$\frac{\phi_1 \quad \dots \quad \phi_n}{\psi}$$

`rtac_tac` *rule* i	replace goal ψ by subgoals $\phi_1, \dots, \phi_n$ —backward proof
`dtac_tac` *rule* i	replace assumption ϕ_1 by ψ —new subgoals $\phi_2, \dots, \phi_n$ —forward proof
`etac_tac` *rule* i	apply an elimination rule — new subgoals $\phi_2, \dots, \phi_n$

“Try Everything” Tactics

`Auto_tac`	break up & try to prove all subgoals — may leave many subgoals
`Force_tac` i	prove subgoal i using everything — or give up

These call the simplifier and the classical reasoner.

Simplification Tactics

`Simp_tac` i	simplify conclusion
`Asm_simp_tac` i	... using assumptions as extra rewrite rules
`Full_simp_tac` i	simplify assumptions and conclusion
`Asm_full_simp_tac` i	... using assumptions as extra rewrite rules

These apply rewrite rules and specialized proof procedures to subgoal i.

Using Your Own Simplification Rules

Add them globally:

```
Addsimps [my_thm];
```

Or add them locally:

```
by (simp_tac (simpset() addsimps [my_thm]) 2);
      !                                   !     note lower case!
```

- Try conditional rules like `m<n ==> m mod n = m`.
- To sort, use permutative rules like `m*n = n*m`.

Using the Tableau Prover

`Blast_tac` i search for a proof of subgoal i

Some rules that work with `Blast_tac`:

```
[| x<=y;  y <=x |] ==> x=y
```

Introduction rule: backward proof

```
{x} = {y} ==> x=y
```

Destruction rule: forward proof

```
[| P|Q;  P ==> R;  Q ==> R |] ==> R
```

Elimination rule

Using Your Own Tableau Rules

Easy way: prove an equivalence like `finite_Un`:

```
finite (A Un B) = (finite A & finite B)
```

Then install it—to simplifier also—by

```
AddIffs [finite_Un];
```

Or add them locally:

```
by (blast_tac (claset() addIs intro_rules
                        addDs destruction_rules
                        addEs elim_rules) 2);
```

Rules are used to break down formulas

Finding Theorems in the Library

```
thms_containing ["map", "rev"];

[("List.rev_concat",
  "rev (concat ?xs) = concat (map rev (rev ?xs))"),
 ("List.rev_map",
  "rev (map ?f ?xs) = map ?f (rev ?xs)")]
: (string * thm) list

thms_containing ["op div", "op mod", "op <"];

[("IntDiv.zmod_zmult2_eq",
  "#0 < ?c ==> ?a mod (?b * ?c) =
              ?b * (?a div ?b mod ?c) + ?a mod ?b")]
```

Result is a list of names and theorems — as ML values.

An infix has a declared name or the default op-form.
See theory file!