universität freiburg

The Rules of the Game

Mathias Fleury

January 12, 2024

Low-Level Proofs

There is a divide here:

- Coq is typically taught with the low level approach first. So after 2 hours you can prove that

```
lemma
  assumes P and ‹P ⟹ Q›
  shows Q
  using assms by auto
```

- Isabelle is typically taught with automation in mind.

```
lemma
  fixes n :: nat
  shows ‹(∑ i=0..n. i*i) = n * (n+1) * (2*n+1) div 6›
  by (induction n) (auto simp: algebra_simps)
```

Low-Level Proofs

There is a divide here:

- Coq is typically taught with the low level approach first. So after 2 hours you can prove that

```
lemma
  assumes P and ‹P ⟹ Q›
  shows Q
  using assms by auto
```

- Isabelle is typically taught with automation in mind.

```
lemma
  fixes n :: nat
  shows ‹(∑i=0..n. i*i) = n * (n+1) * (2*n+1) div 6›
  by (induction n) (auto simp: algebra_simps)
```

Low-Level Proofs

There is a divide here:

- Old people (like me) explore in apply style
- Young people go for Isar directly

Low-Level Proofs

But: there is a change going on, because students do no know induction anymore or proper logic.

Finding Theorems

What is My Theorem Named?

Search in the panel. You can use

- _ to write a term
- ”name:” to restrict the names.

Alternative: find_theorems (with the same option).
Or: Use sledgehammer

Applying Theorems

Known Facts

They are applied by rule and HO unification is done: lemma ‹$P \Longrightarrow Q \Longrightarrow P \wedge (Q \wedge P)$›

```
  apply (rule conjI)
  oops
```

Known Facts

definition *P* where ‹*P _ = True*›
lemma *a: ‹P (a :: 'a :: plus)›*
  sorry
lemma shows ‹*P a*›
  supply *[[unify_trace_failure, show_sorts]]*

rule a does not apply, why?

  oops

Natural Deduction Rules

We distinguish between

- introduction rules to infer a symbol like `(?P ⟹ ?Q) ⟹ ?P ⟶ ?Q`
- elimination rule for the consequences of a symbol
  This is often a matter of point of view from the user.

Introduction Rule

lemma ‹$P \Longrightarrow Q \Longrightarrow P \wedge (Q \wedge P)$›

Elimination rule

lemma ‹$P \lor Q \Longrightarrow Q \lor P$›

This is neither an intro rule nor a dest rule.

Dest rules

thm `conjunct1 conjunct2`

Tactics

The tactics are:

- *frule* = unify with first assumption and add the assumptions
- *drule* = *frule* + remove assumption
- *intro* = *rule* repeated until fix-point

lemma "$\forall$ *P Q.* ((*P* $\longrightarrow$ *Q*) $\longrightarrow$ *P* $\wedge$ *Q*)"
  apply (*intro allI conjI impI*)
  oops

The tactics are:

- *rotate_tac* = rotate assumption

Rewriting

Rewriting

The tactics are:

- unfolding = unfolding until fix-point (command, not a tactic)
- `unfold` = substitute (that is the tactic)
- `subst` = substitute
- `hypsubst` = subsitute assumptions until fix-point, then remove the assumption

Exercise

Prove that in a low-level way:

lemma fixes *n :: nat*
  shows *‹(∑ i=0..n. i) = n * (n+1) div 2›*
  by *(induction n) auto*

Combining theorems

You can instantiate variable with of `impI [ of "2 = 2" ]`: `((2::?'a1) = (2::?'a1)` $\Longrightarrow$ `?Q)` $\Longrightarrow$ `(2::?'a1) = (2::?'a1)` $\longrightarrow$ `?Q`

And theorems with OF or THEN, `impI [ OF TrueI ]`, `TrueI [ THEN impI ]`: `?P` $\longrightarrow$ `True`, `?P` $\longrightarrow$ `True`

Conclusion

It is possible to go purely low-level... but I do not recommend it.
But it is useful for debugging sometimes (why does simp not apply my theorem?)