Content

- Intro & motivation, getting started with Isabelle
- Foundations & Principles
  - Lambda Calculus
  - Higher Order Logic, natural deduction
  - Term rewriting
- Proof & Specification Techniques
  - Datatypes, recursion, induction
  - Inductively defined sets, rule induction
  - Calculational reasoning, mathematics style proofs
  - Hoare logic, proofs about programs

Scope

- Scope of parameters: whole subgoal
- Scope of $\forall, \exists, \ldots$ ends with $; \Rightarrow$

Example:

$$\forall x, y. [\forall y. P y \Rightarrow Q z y; \ Q x y] \Rightarrow \exists x. Q x y$$

means

$$\forall x, y. [\forall y_1. P y_1 \Rightarrow Q z y_1; \ Q x y] \Rightarrow (\exists x_1. Q x_1 y)$$
Natural deduction for quantifiers

\[ \begin{align*}
\forall x. P x & \quad \text{allI} \\
\forall x. P x & \quad \Rightarrow \\
\exists x. P x & \quad \text{exI}
\end{align*} \]

- \text{allI} and \text{exI} introduce new parameters (\( \forall x \)).
- \text{allE} and \text{exE} introduce new unknowns (\( ?x \)).

Slide 5

Instantiating Rules

\text{apply (rule tac x = "term" in rule)}

Like rule, but ?x in rule is instantiated by term before application.

Similar: \text{erule}

\[ ! \text{ x is in rule, not in goal } ! \]

Slide 6

Two Successful Proofs

1. \( \forall x. \exists y. x = y \)
   - apply (rule allI)
   - apply (rule exI)

best practice exploration
- apply (rule \text{Jac} x = "x" in exI)
- apply (rule refl)
- apply (rule refl)
- \( ?y \mapsto \lambda u. u \)

simpler & clearer shorter & trickier

Slide 7

Two Unsuccessful Proofs

1. \( \exists y. \forall x. x = y \)
   - apply (rule \text{Jac} x = ??? in exI)
   - apply (rule exI)

Principle:

\( ?f x_1 \ldots x_n \) can only be replaced by term \( t \)

If \( \text{params}(t) \subseteq x_1 \ldots x_n \)

Slide 8
Safe and Unsafe Rules

Safe  allI, exE
Unsafe  allE, exI

Create parameters first, unknowns later

Parameter names

Parameter names are chosen by Isabelle

1. \( \forall x. \exists y. x = y \)
   apply (rule allI)
1. \( \land x. \exists y. x = y \)
   apply (rule_tac x = "x" in exI)

Brittle!

Renaming parameters

1. \( \forall x. \exists y. x = y \)
   apply (rule allI)
1. \( \land x. \exists y. x = y \)
   apply (rename_tac N)
   apply (rule_tac x = "N" in exI)

In general:
\( \text{(rename_tac } x_1 \ldots x_n) \) renames the rightmost (inner) \( n \) parameters to \( x_1 \ldots x_n \)
Forward Proof: frule and drule

apply (frule \(<\ rule >\))

Rule:

\[ [A_1; \ldots; A_m] \rightarrow A \]

Subgoal:

1. \([B_1; \ldots; B_n] \rightarrow C\]

Substitution:

\[ \sigma(B_i) \equiv \sigma(A_1) \]

New subgoals:

1. \(\sigma([B_1; \ldots; B_n] \rightarrow A_2)\)

\[ \vdots \]

m-1. \(\sigma([B_1; \ldots; B_n] \rightarrow A_m)\)

m. \(\sigma([B_1; \ldots; B_n; A] \rightarrow C)\)

Like frule but also deletes \(B_i\):

apply (drule \(<\ rule >\))

Examples for Forward Rules

\[
\frac{P \land Q}{P} \quad \text{conjunct1} \quad \frac{P \land Q}{Q} \quad \text{conjunct2}
\]

\[
\frac{P \rightarrow Q}{Q} \quad P \quad \text{mp}
\]

\[
\forall x. P x \quad \frac{P z}{P / z} \quad \text{spec}
\]

Forward Proof: OF

\[ r[\text{OF } r_1 \ldots r_n] \]

Prove assumption 1 of theorem \(r\) with theorem \(r_1\), and assumption 2 with theorem \(r_2\), and...

Rule \(r\):

\[ [A_1; \ldots; A_n] \rightarrow A \]

Rule \(r_1\):

\[ [B_1; \ldots; B_n] \rightarrow B \]

Substitution:

\[ \sigma(B) \equiv \sigma(A_1) \]

\[ r[\text{OF } r_1] \quad \sigma([B_1; \ldots; B_n; A_1; \ldots; A_n] \rightarrow A) \]

Forward proofs: THEN

\[ r_1 \ [\text{THEN } r_2] \quad \text{means} \quad r_2 \ [\text{OF } r_1] \]
Hilbert’s Epsilon Operator

(David Hilbert, 1862-1943)

\( \varepsilon \, x. \, P \, x \) is a value that satisfies \( P \) (if such a value exists)

\( \varepsilon \) also known as description operator.
In Isabelle the \( \varepsilon \)-operator is written \( \text{SOME} \, x. \, P \, x \)

\[ P \, \varepsilon x \equiv P \, (\text{SOME} \, x. \, P \, x) \]

Some Automation

More Proof Methods:

- **apply** (intro <intro-rules>) repeatedly applies intro rules
- **apply** (elim <elim-rules>) repeatedly applies elim rules
- **apply** clarify applies all safe rules that do not split the goal
- **apply** safe applies all safe rules
- **apply** blast an automatic tableau prover (works well on predicate logic)
- **apply** fast another automatic search tactic
We have learned so far...

- Proof rules for negation and contradiction
- Proof rules for predicate calculus
- Safe and unsafe rules
- Forward Proof
- The Epsilon Operator
- Some automation