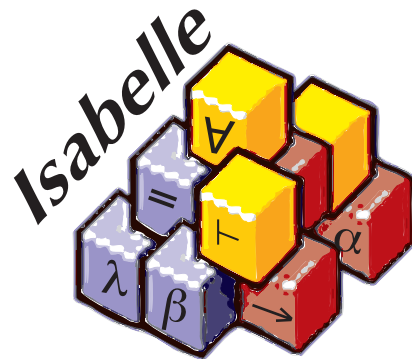


Introduction to Isabelle

Clemens Ballarin
Universität Innsbruck



HOL

Contents

- ▶ Intro & motivation, getting started with Isabelle
- ▶ Foundations & Principles
 - ▶ Lambda Calculus
 - ▶ Types & Classes
 - ▶ Natural Deduction
- ▶ Proof & Specification Techniques
 - ▶ Isar: mathematics style proofs
 - ▶ Inductively defined sets, rule induction
 - ▶ Datatypes, structural induction
 - ▶ Recursive functions & code generation

Types

Types in Isabelle

$$\tau ::= B \mid '\nu \mid '?\nu \mid (\tau, \dots, \tau) K \mid \tau :: C$$

B base types

ν type variables

K type constructors

C sorts

- ▶ **Base types:** `bool, int, ...`
- ▶ **Type variables:** `'a, 'a1, 'name, '?a, ...`
- ▶ **Type constructors:** `int list, 'a list, 'a \Rightarrow 'b, ...`
- ▶ **Sorts:** `'a :: order, 'a :: {plus, order}, ...`
Restrict a type to one or more classes.

Terms in Isabelle

$$t ::= v \mid ?v \mid c \mid (t\ t) \mid (\lambda x. t) \mid (t :: \tau)$$

v, x variable names

c constants

- ▶ **Variables & constants:** $a, a1, \text{name}, \dots$
- ▶ **Type constraints:** $f :: 'a \Rightarrow 'b$
Restrict a term to a type.
- ▶ **Schematic variables:** variables that can be instantiated.

Type Classes

Similar to Haskell's type classes, but with semantic properties

```
class order =  
  fixes less_eq (infix " ≤ " 50)  
    and less (infix " < " 50)  
  assumes order_refl: "  $x \leq x$  "  
    and order_trans: "  $\llbracket x \leq y; y \leq z \rrbracket \implies x \leq z$  "  
  and ...
```

Theorems can be proved in the abstract

```
lemma (in order) order_less_trans:  
"  $\bigwedge x. \llbracket x < y; y < z \rrbracket \implies x < z$  "
```

Here x, y and z have type $'a :: order$.

Type Classes

Can be used for subtyping

```
class linorder = order +  
  assumes linorder_linear: " $x \leq y \vee y \leq x$ "
```

Can be instantiated

```
instance nat :: "{order, linorder}" by ...
```

Schematic Variables

Two operational roles of variables.

- ▶ In lemmas they must be **instantiated** when applied.

$$\llbracket X; Y \rrbracket \implies X \wedge Y$$

- ▶ During proofs they must not be instantiated.

lemma " $x + 0 = 0 + x$ "

Convention: lemma must be true for all x .

Isabelle has **free** (x), **bound** (x), and **schematic** ($?x$) variables.

Only schematic variables can be instantiated.

Free converted into schematic after proof is finished.

Higher-Order Unification

Unification:

Find substitution σ on variables for terms s, t such that $\sigma(s) = \sigma(t)$

In Isabelle:

Find substitution σ on schematic variables such that $\sigma(s) =_{\alpha\beta\eta} \sigma(t)$

Examples:

$$\begin{array}{llll} ?X \wedge ?Y & =_{\alpha\beta\eta} & x \wedge x & [?X \mapsto x, ?Y \mapsto x] \\ ?P \ x & =_{\alpha\beta\eta} & x \wedge x & [?P \mapsto \lambda x. x \wedge x] \\ P \ (?f \ x) & =_{\alpha\beta\eta} & ?Y \ x & [?f \mapsto \lambda x. x, ?Y \mapsto P] \end{array}$$

Higher-Order: schematic variables can be functions.

Higher-Order Unification

- ▶ Unification modulo $\alpha\beta$ is semi-decidable
- ▶ Unification modulo $\alpha\beta\eta$ is undecidable
- ▶ Higher-Order Unification has possibly infinitely many most general solutions

But:

- ▶ Most cases are well-behaved
- ▶ Important fragments (like Higher-Order Patterns) are decidable

Higher-Order Patterns

Higher-Order Pattern:

- ▶ is a term in β -normal form where
- ▶ each occurrence of a schematic variable is of the form $f t_1 \dots t_n$
- ▶ and the $t_1 \dots t_n$ are η -convertible into n distinct bound variables

Preview: Proofs in Isabelle

Proofs in Isabelle

General schema

```
lemma name: "⟨goal⟩"  
  apply ⟨method⟩  
  apply ⟨method⟩  
  ...  
done
```

- ▶ Sequential application of methods until all **subgoals** are solved.

The Proof State

1. $\bigwedge x_1 \dots x_p. \llbracket A_1; \dots; A_n \rrbracket \implies B$

2. $\bigwedge y_1 \dots y_q. \llbracket C_1; \dots; C_m \rrbracket \implies D$

$x_1 \dots x_p$ Parameters

$A_1 \dots A_n$ Local assumptions

B Current (sub)goal

Isabelle Theories

Syntax

```
theory  $\langle name \rangle$   
imports  $\langle import_1 \rangle \dots \langle import_n \rangle$   
begin  
(declarations, definitions, theorems, proofs, ...)*  
end
```

- ▶ $\langle name \rangle$: name of theory. Must live in file $\langle name \rangle.thy$
- ▶ $\langle import_i \rangle$: name of **imported** theory. Import transitive.

Unless you need something special:

```
theory  $\langle name \rangle$   
imports Main  
begin
```

Natural Deduction

Natural Deduction Rules

$$\begin{array}{l} \frac{A \quad B}{A \wedge B} \text{ conjI} \qquad \frac{A \wedge B \quad \llbracket A; B \rrbracket \Longrightarrow C}{C} \text{ conjE} \\ \\ \frac{A}{A \vee B} \quad \frac{B}{A \vee B} \text{ disjI1/2} \qquad \frac{A \vee B \quad A \Longrightarrow C \quad B \Longrightarrow C}{C} \text{ disjE} \\ \\ \frac{A \Longrightarrow B}{A \longrightarrow B} \text{ disjE} \qquad \frac{A \longrightarrow B \quad A \quad B \Longrightarrow C}{C} \text{ impE} \end{array}$$

For each connective (\wedge , \vee , etc):
introduction and **elimination** rules

Proof by Assumption

apply assumption

proves

$$1. \llbracket B_1; \dots; B_m \rrbracket \Longrightarrow C$$

by unifying C with one of the B_i

There may be more than one matching B_i
and multiple unifiers.

Backtracking!

Explicit backtracking command: **back**

Intro Rules

Intro rules decompose formulae to the right of \implies .

apply (rule $\langle intro-rule \rangle$)

Intro rule $\llbracket A_1; \dots; A_n \rrbracket \implies A$ means

- ▶ To prove A it suffices to show $A_1 \dots A_n$

Applying rule $\llbracket A_1; \dots; A_n \rrbracket \implies A$ to subgoal C :

- ▶ unify A and C
- ▶ replace C with n new subgoals $A_1 \dots A_n$

Elim Rules

Elim rules decompose formulae on the left of \implies .

apply (erule <elim-rule>)

Elim rule $\llbracket A_1; A_2; \dots; A_n \rrbracket \implies A$ means

- ▶ If I know A_1 and want to prove A it suffices to show $A_2 \dots A_n$

Applying rule $\llbracket A_1; \dots; A_n \rrbracket \implies A$ to subgoal C :

Like **rule** but also

- ▶ unifies first premise of rule with an assumption
- ▶ eliminates that assumption

Demo: Propositional Reasoning

Iff, Negation, True and False

$$\frac{A \implies B \quad B \implies A}{A = B} \text{ iffI}$$

$$\frac{A = B \quad \llbracket A \longrightarrow B; B \longrightarrow A \rrbracket \implies C}{C} \text{ iffE}$$

$$\frac{A = B}{A \implies B} \text{ iffD1}$$

$$\frac{A = B}{B \implies A} \text{ iffD2}$$

$$\frac{A \implies \text{False}}{\neg A} \text{ notI}$$

$$\frac{\neg A \quad A}{P} \text{ notE}$$

$$\frac{}{\text{True}} \text{ TrueI}$$

$$\frac{\text{False}}{P} \text{ FalseE}$$

Equality

$$\frac{}{t = t} \text{ refl} \qquad \frac{s = t}{t = s} \text{ sym} \qquad \frac{r = s \quad s = t}{r = t} \text{ trans}$$

$$\frac{s = t \quad P \ s}{P \ t} \text{ subst}$$

Rarely needed explicitly — used implicitly by term rewriting.

Classical

$$\frac{}{P = \text{True} \vee P = \text{False}} \text{True_or_False}$$

$$\frac{}{P \vee \neg P} \text{excluded_middle}$$

$$\frac{\neg A \implies \text{False}}{A} \text{ccontr} \qquad \frac{\neg A \implies A}{A} \text{classical}$$

- ▶ excluded_middle, ccontr and classical not derivable from the other rules.
- ▶ If we include True_or_False, they are derivable.

They make the logic **classical**, **non-constructive**.

Cases

$$\frac{}{P \vee \neg P} \text{excluded_middle}$$

is a case distinction on type *bool*.

Isabelle can do case distinctions on arbitrary terms:

apply (case_tac $\langle term \rangle$)

Safe and Not so Safe

Safe rules preserve provability:

conjI, impl, notI, iffI, refl, ccontr, classical, conjE, disjE

$$\frac{A \quad B}{A \wedge B} \text{conjI}$$

Unsafe rules can turn a provable goal into an unprovable one:

disjI1, disjI2, impE, iffD1, iffD2, notE

$$\frac{A}{A \vee B} \text{disjI1}$$

Apply safe rules before unsafe ones.

Demo: More Rules

Quantifiers

Scope

- ▶ Scope of parameters: whole subgoal
- ▶ Scope of \forall, \exists, \dots : ends with meta-level connective:
 \implies, \equiv or $;$.

Example:

$$\bigwedge x y. \llbracket \forall y. P y \longrightarrow Q z y; Q x y \rrbracket \implies \exists x. Q x y$$

means

$$\bigwedge x y. \llbracket (\forall y_1. P y_1 \longrightarrow Q z y_1); Q x y \rrbracket \implies (\exists x_1. Q x_1 y)$$

Natural Deduction for Quantifiers

$$\frac{\bigwedge x. P x}{\forall x. P x} \text{allI} \qquad \frac{\forall x. P x \quad P ?x \implies R}{R} \text{allE}$$
$$\frac{P ?x}{\exists x. P x} \text{exI} \qquad \frac{\exists x. P x \quad \bigwedge x. P x \implies R}{R} \text{exE}$$

- ▶ **allI** and **exE** introduce new parameters ($\bigwedge x$).
- ▶ **allE** and **exI** introduce new unknowns ($?x$).

Instantiating Rules

apply (rule_tac x = " $\langle term \rangle$ " in $\langle rule \rangle$)

Like `rule`, but $?x$ in $\langle rule \rangle$ is instantiated by $\langle term \rangle$ before application.

Similar: `erule_tac`

- ▶ x is in $\langle rule \rangle$, not in goal.
- ▶ $\langle term \rangle$ may contain parameters from the goal and those introduced in Isar texts (later).

Two Successful Proofs

1. $\forall x. \exists y. x = y$

apply (rule allI)

1. $\bigwedge x. \exists y. x = y$

Best practice

apply (rule_tac x = "x" in exI)

1. $\bigwedge x. x = x$

apply (rule refl)

simpler & clearer

Exploration

apply (rule exI)

1. $\bigwedge x. x = ?y\ x$

apply (rule refl)

$?y \mapsto \lambda u. u$

shorter & trickier

Two Unsuccessful Proofs

1. $\exists y. \forall x. x = y$

apply (rule_tac x = ??? in exI)

apply (rule exI)

1. $\forall x. x = ?y$

apply (rule allI)

1. $\bigwedge x. x = ?y$

apply (rule refl)

$?y \mapsto x$ yields $\bigwedge x'. x' = x$

Principle

$?f\ x_1 \dots x_n$ can only be replaced by term t

if $\text{params}(t) \subseteq x_1, \dots, x_n$.

Safe and Unsafe Rules

Safe allI, exE

Unsafe allE, exI

Create parameters first, unknowns later

Demo: Quantifier Proofs

Parameter Names

Parameter names are chosen by Isabelle

1. $\forall x. \exists y. x = y$

apply (rule all)

1. $\wedge x. \exists y. x = y$

apply (rule_tac x = "x" in exI)

Brittle!

Renaming Parameters

1. $\forall x. \exists y. x = y$

apply (rule allI)

1. $\bigwedge x. \exists y. x = y$

apply (rename_tac N)

1. $\bigwedge N. \exists y. N = y$

apply (rule_tac x = "N" in exI)

In general

(rename_tac $x_1 \dots x_n$) renames the rightmost (inner) n parameters to $x_1 \dots x_n$.

Forward Proof: frule and drule

apply (frule $\langle rule \rangle$)

Rule: $\llbracket A_1; \dots; A_m \rrbracket \Longrightarrow A$

Subgoal: 1. $\llbracket B_1; \dots; B_n \rrbracket \Longrightarrow C$

Substitution: $\sigma(B_i) \equiv \sigma(A_1)$

Unifiable assumption B_i is chosen.

New subgoals: 1. $\sigma(\llbracket B_1; \dots; B_n \rrbracket \Longrightarrow A_2)$

\vdots

m-1. $\sigma(\llbracket B_1; \dots; B_n \rrbracket \Longrightarrow A_m)$

m. $\sigma(\llbracket B_1; \dots; B_n; A \rrbracket \Longrightarrow C)$

Like **frule** but also deletes B_i : **apply** (drule $\langle rule \rangle$)

Examples for Forward Rules

$$\frac{P \wedge Q}{P} \text{ conjunct1} \quad \frac{P \wedge Q}{Q} \text{ conjunct2}$$

$$\frac{P \longrightarrow Q \quad P}{Q} \text{ mp}$$

$$\frac{\forall x. P \ x}{P \ ?x} \text{ spec}$$

Forward Proof: OF

$$r \text{ [OF } r_1 \dots r_n]$$

Prove assumption 1 of theorem r with theorem r_1 , and assumption 2 with theorem r_2 , etc ...

$$\text{Rule } r \quad \llbracket A_1; \dots; A_m \rrbracket \implies A$$

$$\text{Rule } r_1 \quad \llbracket B_1; \dots; B_n \rrbracket \implies B$$

$$\text{Substitution} \quad \sigma(B) \equiv \sigma(A_1)$$

$$r \text{ [OF } r_1] \quad \sigma(\llbracket B_1; \dots; B_n; A_2; \dots; A_m \rrbracket \implies A)$$

May use underscore to omit an argument:

$r \text{ [OF } _ r_2]$ proves assumption 2 with theorem r_2 .

Forward proofs: THEN

r_1 [THEN r_2] means r_2 [OF r_1]

Demo: Forward Proofs

Hilbert's Epsilon Operator



(David Hilbert, 1862-1943)

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

ε also known as **description operator**.

In Isabelle the ε -operator is written $\text{SOME } x. P x$

$$\frac{P ?x}{P (\text{SOME } x. P x)} \text{someI}$$

More Epsilon

ε implies Axiom of Choice:

$$\forall x. \exists y. Q\ x\ y \implies \exists f. \forall x. Q\ x\ (f\ x)$$

Existential and universal quantification can be defined with ε .

Isabelle also knows the **definite description operator** ι :

$$\frac{}{(\text{THE } x. x = a) = a} \text{the_eq_trivial}$$

More Proof Methods

apply (intro $\langle intro-rules \rangle$)	repeatedly applies intro rules
apply (elim $\langle elim-rules \rangle$)	repeatedly applies elim rules
apply clarify	applies all safe rules that do not split the goal
apply safe	applies all safe rules
apply fast	sequent based automatic
apply best	search tactics
apply blast	an automatic tableaux prover (works well on predicate logic)
apply metis	resolution prover for first-order logic with equality

Epsilon and Automation Demo

More on Automation

Review:

Safe and unsafe rule; heuristics: use safe before unsafe

This can be automated

Automated methods (fast, blast, clarify etc) are not hardwired.

Safe and unsafe intro and elim rules can be declared.

Syntax:

[<kind>!] for safe rules (<kind> one of intro, elim, dest)

[<kind>] for unsafe rules

More on Automation

Application (roughly):

do safe rules first, search/backtrack on unsafe rules only

Example:

declare attribute globally
remove attribute globally
use locally
delete locally

declare conj1 [intro!] allE [elim]
declare allE [rule del]
apply (blast intro: some1)
apply (blast del: conj1)

Demo: Attributes

We Have Learned so far...

- ▶ Proof rules propositional logic
- ▶ Proof rules for predicate calculus
- ▶ Safe and unsafe rules
- ▶ Forward proof
- ▶ The Epsilon Operator
- ▶ Some automation (classical reasoner)