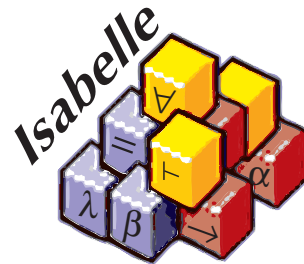


IJCAR 2004 — Tutorial 4



Introduction to the Isabelle Proof Assistant



Clemens Ballarin



Gerwin Klein

Tutorial Schedule

- ▶ Session I
 - ▶ Basics
- ▶ Session II
 - ▶ Specification Tools
 - ▶ Readable Proofs
- ▶ Session III
 - ▶ More on Readable Proofs
 - ▶ Modules
- ▶ Session IV
 - ▶ Applications
 - ▶ Q & A session with Larry Paulson

Session I

Basics

System Architecture

Isabelle — Generic, interactive theorem prover

System Architecture

Isabelle	— Generic, interactive theorem prover
Standard ML	— Logic implemented as ADT

System Architecture

HOL, ZF

— Object-logics

Isabelle

— Generic, interactive theorem prover

Standard ML

— Logic implemented as ADT

System Architecture

Proof General	— User interface
HOL, ZF	— Object-logics
Isabelle	— Generic, interactive theorem prover
Standard ML	— Logic implemented as ADT

System Architecture

User can access all layers!

Proof General — User interface

HOL, ZF — Object-logics

Isabelle — Generic, interactive theorem prover

Standard ML — Logic implemented as ADT

Documentation

Available from <http://isabelle.in.tum.de>

Documentation

Available from <http://isabelle.in.tum.de>

► Learning Isabelle

Documentation

Available from <http://isabelle.in.tum.de>

- ▶ Learning Isabelle
 - ▶ Tutorial on Isabelle/HOL (LNCS 2283)
 - ▶ Tutorial on Isar
 - ▶ Tutorial on Locales

Documentation

Available from <http://isabelle.in.tum.de>

- ▶ Learning Isabelle
 - ▶ Tutorial on Isabelle/HOL (LNCS 2283)
 - ▶ Tutorial on Isar
 - ▶ Tutorial on Locales
- ▶ Reference Manuals

Documentation

Available from <http://isabelle.in.tum.de>

- ▶ Learning Isabelle
 - ▶ Tutorial on Isabelle/HOL (LNCS 2283)
 - ▶ Tutorial on Isar
 - ▶ Tutorial on Locales
- ▶ Reference Manuals
 - ▶ Isabelle/Isar Reference Manual
 - ▶ Isabelle Reference Manual
 - ▶ Isabelle System Manual

Documentation

Available from <http://isabelle.in.tum.de>

- ▶ Learning Isabelle
 - ▶ Tutorial on Isabelle/HOL (LNCS 2283)
 - ▶ Tutorial on Isar
 - ▶ Tutorial on Locales
- ▶ Reference Manuals
 - ▶ Isabelle/Isar Reference Manual
 - ▶ Isabelle Reference Manual
 - ▶ Isabelle System Manual
- ▶ Reference Manuals for Object-Logics

Isabelle's Meta-Logic

- ▶ Intuitionistic fragment of Church's theory of simple types.

Isabelle's Meta-Logic

- ▶ Intuitionistic fragment of Church's theory of simple types.
- ▶ With type variables.

Isabelle's Meta-Logic

- ▶ Intuitionistic fragment of Church's theory of simple types.
- ▶ With type variables.
- ▶ Can be used to formalise your own object-logic.

Isabelle's Meta-Logic

- ▶ Intuitionistic fragment of Church's theory of simple types.
- ▶ With type variables.
- ▶ Can be used to formalise your own object-logic.
- ▶ Normally, use rich infrastructure of the object-logics HOL and ZF.

Isabelle's Meta-Logic

- ▶ Intuitionistic fragment of Church's theory of simple types.
- ▶ With type variables.
- ▶ Can be used to formalise your own object-logic.
- ▶ Normally, use rich infrastructure of the object-logics HOL and ZF.
- ▶ This presentation assumes HOL.

Types

Syntax

Syntax:

$$\begin{array}{l} \tau ::= (\tau) \\ \quad | \text{'}a \text{' } | \text{'}b \text{' } | \dots \end{array} \quad \text{type variables}$$

Syntax

Syntax:

$$\begin{array}{l} \tau ::= (\tau) \\ \quad | \text{'a'} \mid \text{'b'} \mid \dots \\ \quad | \tau \Rightarrow \tau \end{array}$$

type variables
total functions

Syntax

Syntax:

$\tau ::=$	(τ)	
	$'a \mid 'b \mid \dots$	type variables
	$\tau \Rightarrow \tau$	total functions
	$bool \mid nat \mid \dots$	HOL base types

Syntax

Syntax:

$\tau ::=$	(τ)	
	$'a \mid 'b \mid \dots$	type variables
	$\tau \Rightarrow \tau$	total functions
	$bool \mid nat \mid \dots$	HOL base types
	$\tau \times \tau$	HOL pairs (ascii: *)

Syntax

Syntax:

τ	$::=$	(τ)	
		$'a \mid 'b \mid \dots$	type variables
		$\tau \Rightarrow \tau$	total functions
		$bool \mid nat \mid \dots$	HOL base types
		$\tau \times \tau$	HOL pairs (ascii: *)
		$\tau \text{ list}$	HOL lists

Syntax

Syntax:

$\tau ::=$	(τ)	
	$'a \mid 'b \mid \dots$	type variables
	$\tau \Rightarrow \tau$	total functions
	$bool \mid nat \mid \dots$	HOL base types
	$\tau \times \tau$	HOL pairs (ascii: *)
	$\tau \textit{ list}$	HOL lists
	$\dots$	user-defined types

Syntax

Syntax:

$\tau ::=$	(τ)	
	$'a \mid 'b \mid \dots$	type variables
	$\tau \Rightarrow \tau$	total functions
	$bool \mid nat \mid \dots$	HOL base types
	$\tau \times \tau$	HOL pairs (ascii: *)
	$\tau \textit{ list}$	HOL lists
	$\dots$	user-defined types

Parentheses: $T1 \Rightarrow T2 \Rightarrow T3 \quad \equiv \quad T1 \Rightarrow (T2 \Rightarrow T3)$

Introducing new Types: **typedekl**

typedekl *name*

Introduces new “opaque” type *name* without definition.

Introducing new Types: **typedekl**

typedekl *name*

Introduces new “opaque” type *name* without definition.

Example:

typedekl *addr* — An abstract type of addresses.

Terms

Syntax

Syntax: (curried version)

$$term ::= (term)$$

Syntax

Syntax: (curried version)

$$\begin{array}{l} term ::= (term) \\ \quad | \quad a \end{array}$$

constant or variable (identifier)

Syntax

Syntax: (curried version)

$term ::= (term)$	
a	constant or variable (identifier)
$term\ term$	function application

Syntax

Syntax: (curried version)

$term$	$::=$	$(term)$	
		a	constant or variable (identifier)
		$term\ term$	function application
		$\lambda x. term$	function “abstraction”

Syntax

Syntax: (curried version)

$term$	$::=$	$(term)$	
		a	constant or variable (identifier)
		$term\ term$	function application
		$\lambda x. term$	function “abstraction”
		$\dots$	lots of syntactic sugar

Syntax

Syntax: (curried version)

$term$	$::=$	$(term)$	
		a	constant or variable (identifier)
		$term\ term$	function application
		$\lambda x. term$	function “abstraction”
		$\dots$	lots of syntactic sugar

Examples: $f\ (g\ x)\ y$ $h\ (\lambda x. f\ (g\ x))$

Syntax

Syntax: (curried version)

$term$	$::=$	$(term)$	
		a	constant or variable (identifier)
		$term\ term$	function application
		$\lambda x. term$	function “abstraction”
		$\dots$	lots of syntactic sugar

Examples: $f\ (g\ x)\ y$ $h\ (\lambda x. f\ (g\ x))$

Parentheses: $f\ a_1\ a_2\ a_3 \equiv ((f\ a_1)\ a_2)\ a_3$

Schematic variables

Three kinds of variables:

- ▶ bound: $\forall x. x = x$
- ▶ free: $x = x$

Schematic variables

Three kinds of variables:

- ▶ bound: $\forall x. x = x$
- ▶ free: $x = x$
- ▶ **schematic**: $?x = ?x$ (“unknown”)

Schematic variables

Three kinds of variables:

- ▶ bound: $\forall x. x = x$
- ▶ free: $x = x$
- ▶ **schematic**: $?x = ?x$ (“unknown”)
- ▶ Logically: free = schematic

Schematic variables

Three kinds of variables:

- ▶ bound: $\forall x. x = x$
- ▶ free: $x = x$
- ▶ **schematic**: $?x = ?x$ (“unknown”)
- ▶ Logically: free = schematic
- ▶ Operationally:
 - ▶ free variables are fixed
 - ▶ schematic variables are instantiated by substitutions and unification

Theorems

Connectives of the Meta-Logic

Implication $\Rightarrow$ ($==>$)

For separating premises and conclusion of theorems.

Connectives of the Meta-Logic

Implication $\Rightarrow$ ($==>$)

For separating premises and conclusion of theorems.

Equality $\equiv$ ($==$)

For definitions.

Connectives of the Meta-Logic

Implication $\Rightarrow$ ($==>$)

For separating premises and conclusion of theorems.

Equality $\equiv$ ($==$)

For definitions.

Universal quantifier $\wedge$ ($!!$)

For parameters in goals.

Connectives of the Meta-Logic

Implication $\Rightarrow$ ($==>$)

For separating premises and conclusion of theorems.

Equality $\equiv$ ($==$)

For definitions.

Universal quantifier $\wedge$ ($!!$)

For parameters in goals.

Do not use *inside* object-logic formulae.

Notation

$$\llbracket A_1; \dots ; A_n \rrbracket \Longrightarrow B$$

abbreviates

$$A_1 \Longrightarrow \dots \Longrightarrow A_n \Longrightarrow B$$

Notation

$$\llbracket A_1; \dots ; A_n \rrbracket \Longrightarrow B$$

abbreviates

$$A_1 \Longrightarrow \dots \Longrightarrow A_n \Longrightarrow B$$

$;$ $\approx$ “and”

Introducing New Theorems

- ▶ As axioms.

Introducing New Theorems

- ▶ As axioms.
- ▶ Through definitions.

Introducing New Theorems

- ▶ As axioms.
- ▶ Through definitions.
- ▶ Through proofs.

Introducing New Theorems

- ▶ As axioms.
- ▶ Through definitions.
- ▶ Through proofs.

! Axioms should mainly be used when specifying object-logics. !

Definition (non-recursive)

Declaration:

consts

$sq :: nat \Rightarrow nat$

Definition:

defs

$sq_def: sq\ n \equiv n*n$

Definition (non-recursive)

Declaration:

consts

$sq :: nat \Rightarrow nat$

Definition:

defs

$sq_def: sq\ n \equiv n*n$

Declaration + definition:

constdefs

$sq :: nat \Rightarrow nat$

$sq\ n \equiv n*n$

Proofs

General schema:

```
lemma name: <goal>  
  apply <method>  
  apply <method>  
  ⋮  
done
```

Proofs

General schema:

```
lemma name: <goal>  
  apply <method>  
  apply <method>  
  ⋮  
done
```

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

The proof state

1. $\bigwedge x_1 \dots x_p. \llbracket A_1; \dots ; A_n \rrbracket \Longrightarrow B$
2. $\bigwedge y_1 \dots y_q. \llbracket C_1; \dots ; C_n \rrbracket \Longrightarrow D$

The proof state

1. $\bigwedge x_1 \dots x_p. [A_1; \dots ; A_n] \Longrightarrow B$
2. $\bigwedge y_1 \dots y_q. [C_1; \dots ; C_n] \Longrightarrow D$

$x_1 \dots x_p$ Parameters

$A_1 \dots A_n$ Local assumptions

B Actual (sub)goal

Isabelle Theories

Theory = Source file

Syntax:

```
theory MyTh imports ImpTh1 ... ImpThn begin  
(declarations, definitions, theorems, proofs, ...)*  
end
```

- ▶ *MyTh*: name of theory. Must live in file *MyTh*.thy
- ▶ *ImpTh*_{*i*}: name of *imported* theories. Import transitive.

Theory = Source file

Syntax:

```
theory MyTh imports ImpTh1 ... ImpThn begin  
(declarations, definitions, theorems, proofs, ...)*  
end
```

- ▶ *MyTh*: name of theory. Must live in file *MyTh.thy*
- ▶ *ImpTh*_{*i*}: name of *imported* theories. Import transitive.

Unless you need something special:

```
theory MyTh imports Main begin
```

X-Symbols

Input of funny symbols in Proof General

- ▶ via menu (“X-Symbol”)
- ▶ via ascii encoding (similar to $\text{\LaTeX}$): `\<and>`, `\<or>`,
...
- ▶ via abbreviation: `/\`, `\/`, `-->`, ...

x-symbol	$\forall$	$\exists$	λ	$\neg$	$\wedge$	$\vee$	$\longrightarrow$	$\Rightarrow$
ascii (1)	<code>\<forall></code>	<code>\<exists></code>	<code>\<lambda></code>	<code>\<not></code>	<code>/\</code>	<code>\/</code>	<code>--></code>	<code>=></code>
ascii (2)	ALL	EX	%	~	&			

(1) is converted to x-symbol, (2) stays ascii.

Demo: Isabelle theories

Natural Deduction

Rules

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

$$\frac{A \wedge B}{C} \text{ conjE}$$

$$\frac{}{A \vee B} \quad \frac{}{A \vee B} \text{ disjI1/2}$$

$$\frac{A \vee B}{C} \text{ disjE}$$

$$\frac{}{A \longrightarrow B} \text{ implI}$$

$$\frac{A \longrightarrow B}{C} \text{ impE}$$

Rules

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

$$\frac{A \wedge B}{C} \text{ conjE}$$

$$\frac{A}{A \vee B} \quad \frac{B}{A \vee B} \text{ disjI1/2}$$

$$\frac{A \vee B}{C} \text{ disjE}$$

$$\frac{A \implies B}{A \longrightarrow B} \text{ impl}$$

$$\frac{A \longrightarrow B}{C} \text{ impE}$$

Rules

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

$$\frac{A \wedge B \quad \llbracket A; B \rrbracket \Longrightarrow C}{C} \text{ conjE}$$

$$\frac{A}{A \vee B} \quad \frac{B}{A \vee B} \text{ disjI1/2}$$

$$\frac{A \vee B \quad A \Longrightarrow C \quad B \Longrightarrow C}{C} \text{ disjE}$$

$$\frac{A \Longrightarrow B}{A \longrightarrow B} \text{ impl}$$

$$\frac{A \longrightarrow B \quad A \quad B \Longrightarrow C}{C} \text{ impE}$$

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

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 (backtracking!)

How to prove it by natural deduction

- ▶ **Intro** rules decompose formulae to the right of $\implies$.
apply(rule <intro-rule>)

How to prove it by natural deduction

- ▶ **Intro** rules decompose formulae to the right of $\implies$.

apply(rule <intro-rule>)

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

- ▶ Unify A and C

How to prove it by natural deduction

- ▶ **Intro** rules decompose formulae to the right of $\implies$.

apply(rule <intro-rule>)

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

- ▶ Unify A and C
- ▶ Replace C with n new subgoals $A_1 \dots A_n$

How to prove it by natural deduction

- ▶ **Intro** rules decompose formulae to the right of $\implies$.

apply(rule <intro-rule>)

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 decompose formulae on the left of $\implies$.

apply(erule <elim-rule>)

How to prove it by natural deduction

- ▶ **Intro** rules decompose formulae to the right of $\implies$.

apply(rule <intro-rule>)

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 decompose formulae on the left of $\implies$.

apply(erule <elim-rule>)

Like *rule* but also

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

Demo: natural deduction

Safe and unsafe rules

Safe rules preserve provability

Safe and unsafe rules

Safe rules preserve provability

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

Safe and unsafe rules

Safe rules preserve provability

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

Unsafe rules can turn provable goal into unprovable goal

Safe and unsafe rules

Safe rules preserve provability

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

Unsafe rules can turn provable goal into unprovable goal

disjI1, disjI2, impE,
iffD1, iffD2, notE

Safe and unsafe rules

Safe rules preserve provability

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

Unsafe rules can turn provable goal into unprovable goal

disjI1, disjI2, impE,
iffD1, iffD2, notE

Apply safe rules before unsafe ones

Predicate Logic: $\forall$ and $\exists$

Scope

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

Scope

- ▶ Scope of parameters: whole subgoal
- ▶ Scope of $\forall, \exists, \dots$: ends with $;$ or $\implies$

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

Scope

- ▶ Scope of parameters: whole subgoal
- ▶ Scope of $\forall, \exists, \dots$: ends with $;$ or $\implies$

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

means

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

Natural deduction for quantifiers

$$\frac{}{\forall x. P x} \text{allI}$$

$$\frac{}{\exists x. P x} \text{exI}$$

$$\frac{\forall x. P x}{R} \text{allE}$$

$$\frac{\exists x. P x}{R} \text{exE}$$

Natural deduction for quantifiers

$$\frac{\bigwedge x. P x}{\bigvee x. P x} \text{allI}$$

$$\frac{\bigvee x. P x}{R} \text{allE}$$

$$\frac{}{\bigvee x. P x} \text{exI}$$

$$\frac{\bigvee x. P x \quad \bigwedge x. P x \Rightarrow R}{R} \text{exE}$$

Natural deduction for quantifiers

$$\frac{\wedge x. P x}{\forall x. P x} \text{allI}$$

$$\frac{\forall x. P x \quad P ?x \Rightarrow R}{R} \text{allE}$$

$$\frac{P ?x}{\exists x. P x} \text{exI}$$

$$\frac{\exists x. P x \quad \wedge x. P x \Rightarrow R}{R} \text{exE}$$

Natural deduction for quantifiers

$$\frac{\wedge x. P x}{\forall x. P x} \text{allI}$$

$$\frac{\forall x. P x \quad P ?x \Rightarrow R}{R} \text{allE}$$

$$\frac{P ?x}{\exists x. P x} \text{exI}$$

$$\frac{\exists x. P x \quad \wedge x. P x \Rightarrow R}{R} \text{exE}$$

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

Instantiating rules

apply(*rule_tac* $x = \textit{term}$ *in rule*)

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

Instantiating rules

apply(*rule_tac* $x = \textit{term}$ *in rule*)

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

Similar: *erule_tac*

Instantiating rules

`apply(rule_tac x = "term" in rule)`

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

Similar: *erule_tac*

! *x* is in *rule*, not in the goal **!**

Safe and unsafe rules

Safe all, exE

Unsafe allE, exI

Safe and unsafe rules

Safe all, exE

Unsafe allE, exI

Create parameters first, unknowns later

Forward proofs: frule and drule

apply(*frule* *rulename*)

Forward rule: $A_1 \Longrightarrow A$

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

Forward proofs: frule and drule

apply(*frule* *rulename*)

Forward rule: $A_1 \Longrightarrow A$

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

Unifies: one B_i with A_1

New subgoal: $1. \llbracket B_1; \dots ; B_n; A \rrbracket \Longrightarrow C$

Forward proofs: frule and drule

apply(frule rulename)

Forward rule: $A_1 \Longrightarrow A$

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

Unifies: one B_i with A_1

New subgoal: $1. \llbracket B_1; \dots ; B_n; A \rrbracket \Longrightarrow C$

apply(drule rulename)

Like *frule* but also deletes B_i

Demo: quantifier proofs

Practical Session I

**In the cool morning
A man simplifies, a goal
A theorem is born.**

— Don Syme