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:

$$\begin{array}{lcll} \tau & ::= & (\tau) & \\ & \mid & \textit{'a} \mid \textit{'b} \mid \ldots & \text{type variables} \end{array}$$

Syntax

Syntax:

τ	$::=$	(τ)	
	$\mid$	*'a* $\mid$ *'b* $\mid \ldots$	type variables
	$\mid$	$\tau \Rightarrow \tau$	total functions

Syntax

Syntax:

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

Syntax

Syntax:

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

Syntax

Syntax:

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

Syntax

Syntax:

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

Syntax

Syntax:

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

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

Introducing new Types: typedecl

typedecl $name$

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

Introducing new Types: typedecl

typedecl $name$

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

Example:

typedecl *addr* —An abstract type of addresses.

Terms

Syntax

Syntax: (curried version)

$$term \quad ::= \quad (term)$$

Syntax

Syntax: (curried version)

$$
\begin{array}{lcll}
term & ::= & (term) & \\
 & | & a & \text{constant or variable (identifier)}
\end{array}
$$

Syntax

Syntax: (curried version)

$$
\begin{array}{lcll}
term & ::= & (term) & \\
 & | & a & \text{constant or variable (identifier)} \\
 & | & term\ term & \text{function application}
\end{array}
$$

Syntax

Syntax: (curried version)

$$
\begin{array}{lcll}
term & ::= & (term) & \\
 & | & a & \text{constant or variable (identifier)} \\
 & | & term\ term & \text{function application} \\
 & | & \lambda x.\ term & \text{function “abstraction”}
\end{array}
$$

Syntax

Syntax: (curried version)

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

Syntax

Syntax: (curried version)

$$
\begin{array}{lcll}
term & ::= & (term) & \\
 & | & a & \text{constant or variable (identifier)} \\
 & | & term\ term & \text{function application} \\
 & | & \lambda x.\ term & \text{function “abstraction”} \\
 & | & \ldots & \text{lots of syntactic sugar}
\end{array}
$$

Examples: *f (g x) y* *h (*λ*x. f (g x))*

Syntax

Syntax: (curried version)

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

Examples: *f (g x) y*   *h (*λ*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 $\Longrightarrow$ (==>)

For separating premises and conclusion of theorems.

Connectives of the Meta-Logic

Implication $\Longrightarrow$ (==>)

For separating premises and conclusion of theorems.

Equality $\equiv$ (==)

For definitions.

Connectives of the Meta-Logic

Implication $\Longrightarrow$ (==>)

For separating premises and conclusion of theorems.

Equality $\equiv$ (==)

For definitions.

Universal quantifier $\bigwedge$ (!!)

For parameters in goals.

Connectives of the Meta-Logic

Implication $\Longrightarrow$ (==>)

For separating premises and conclusion of theorems.

Equality $\equiv$ (==)

For definitions.

Universal quantifier $\bigwedge$ (!!)

For parameters in goals.

Do not use *inside* object-logic formulae.

Notation

$$[\![\ A_1;\ \ldots\ ;A_n\ ]\!] \Longrightarrow B$$

abbreviates

$$A_1 \Longrightarrow \ldots \Longrightarrow A_n \Longrightarrow B$$

Notation

$$[\![A_1; \ldots ; A_n]\!] \Longrightarrow B$$

abbreviates

$$A_1 \Longrightarrow \ldots \Longrightarrow A_n \Longrightarrow B$$

$$; \quad \approx \quad \text{“and”}$$

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

The proof state

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

$x_1 \ldots x_p$	Parameters
$A_1 \ldots A_n$	Local assumptions
B	Actual (sub)goal

Isabelle Theories

Theory = Source file

Syntax:

`theory` $MyTh$ `=` $ImpTh_1$ `+ ... +` $ImpTh_n$ `:`
(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 $MyTh$ = $ImpTh_1$ + ... + $ImpTh_n$:
(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 $MyTh$ = Main:

X-Symbols

Input of funny symbols in Proof General

- via menu ("X-Symbol")
- via ascii encoding (similar to LaTeX): \<and>, \<or>, ...
- via abbreviation: /\, \/, -->, ...

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

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

Demo: Isabelle theories

Natural Deduction

Rules

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

$$\frac{}{A \vee B} \; \frac{}{A \vee B} \text{ disjI1/2} \qquad \frac{A \vee B}{C} \text{ disjE}$$

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

Rules

$$\frac{A \quad B}{A \wedge B} \text{ conjI} \qquad \frac{A \wedge B}{C} \text{ conjE}$$

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

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

Rules

$$\frac{A \quad B}{A \wedge B}\ \text{conjI} \qquad \frac{A \wedge B \quad [\![A;B]\!] \Longrightarrow C}{C}\ \text{conjE}$$

$$\frac{A}{A \vee B}\ \frac{B}{A \vee B}\ \text{disjI1/2} \qquad \frac{A \vee B \quad A \Longrightarrow C \quad B \Longrightarrow C}{C}\ \text{disjE}$$

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

Proof by assumption

apply *assumption*

proves

1. $[\![\, B_1; \ldots ; B_m \,]\!] \Longrightarrow C$

by unifying C with one of the B_i

Proof by assumption

apply *assumption*

proves

$$1.\ [\![\, B_1; \ldots ; B_m \,]\!] \Longrightarrow C$$

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 $\Longrightarrow$.

  apply_(rule <intro-rule>)_

How to prove it by natural deduction

- Intro rules decompose formulae to the right of $\Longrightarrow$.

  apply(*rule <intro-rule>*)

  Applying rule $[\![\, A_1; \ldots ; A_n \,]\!] \Longrightarrow A$ to subgoal C:

  - Unify A and C

How to prove it by natural deduction

- Intro rules decompose formulae to the right of $\Longrightarrow$.

  apply(*rule <intro-rule>*)

  Applying rule $[\![A_1; \ldots ; A_n]\!] \Longrightarrow A$ to subgoal C:
  - Unify A and C
  - Replace C with n new subgoals $A_1 \ldots A_n$

How to prove it by natural deduction

- Intro rules decompose formulae to the right of $\Longrightarrow$.

  apply(*rule <intro-rule>*)

  Applying rule $[\![A_1; \ldots ; A_n]\!] \Longrightarrow A$ to subgoal C:
  - Unify A and C
  - Replace C with n new subgoals $A_1 \ldots A_n$
- Elim rules decompose formulae on the left of $\Longrightarrow$.

  apply(*erule <elim-rule>*)

How to prove it by natural deduction

- Intro rules decompose formulae to the right of $\Longrightarrow$.

  apply(*rule <intro-rule>*)

  Applying rule $[\![A_1; \ldots ; A_n]\!] \Longrightarrow A$ to subgoal C:
  - Unify A and C
  - Replace C with n new subgoals $A_1 \ldots A_n$
- Elim rules decompose formulae on the left of $\Longrightarrow$.

  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 $\Longrightarrow$

Scope

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

$$\bigwedge x\, y.\ [\![\ \forall y.\ P\ y \longrightarrow Q\ z\ y;\ \ Q\ x\ y\ ]\!] \Longrightarrow \exists x.\ Q\ x\ y$$

Scope

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

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

means

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

Natural deduction for quantifiers

$$\frac{}{\forall x.\ P\ x}\ \texttt{allI} \qquad \frac{\forall x.\ P\ x}{R}\ \texttt{allE}$$

$$\frac{}{\exists x.\ P\ x}\ \texttt{exI} \qquad \frac{\exists x.\ P\ x}{R}\ \texttt{exE}$$

Natural deduction for quantifiers

$$\frac{\bigwedge x.\ P\ x}{\forall x.\ P\ x}\ \texttt{allI} \qquad \frac{\forall x.\ P\ x \quad \quad}{R}\ \texttt{allE}$$

$$\frac{}{\exists x.\ P\ x}\ \texttt{exI} \qquad \frac{\exists x.\ P\ x \quad \bigwedge x.\ P\ x \Longrightarrow R}{R}\ \texttt{exE}$$

Natural deduction for quantifiers

$$\frac{\bigwedge x.\ P\ x}{\forall x.\ P\ x}\ \texttt{allI} \qquad \frac{\forall x.\ P\ x \quad P\ ?x \Longrightarrow R}{R}\ \texttt{allE}$$

$$\frac{P\ ?x}{\exists x.\ P\ x}\ \texttt{exI} \qquad \frac{\exists x.\ P\ x \quad \bigwedge x.\ P\ x \Longrightarrow R}{R}\ \texttt{exE}$$

Natural deduction for quantifiers

$$\frac{\bigwedge x.\ P\ x}{\forall x.\ P\ x}\ \texttt{allI} \qquad \frac{\forall x.\ P\ x \quad P\ ?x \Longrightarrow R}{R}\ \texttt{allE}$$

$$\frac{P\ ?x}{\exists x.\ P\ x}\ \texttt{exI} \qquad \frac{\exists x.\ P\ x \quad \bigwedge x.\ P\ x \Longrightarrow R}{R}\ \texttt{exE}$$

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

Instantiating rules

apply(*rule_tac x* = "$term$" *in* $rule$)

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

Instantiating rules

apply(*rule_tac x* = "$term$" *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 allI, exE

Unsafe allE, exI

Safe and unsafe rules

Safe allI, 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. $\llbracket B_1; \ldots ; B_n \rrbracket \Longrightarrow C$

Forward proofs: frule and drule

apply(*frule* $rulename$)

Forward rule: $A_1 \Longrightarrow A$

Subgoal: *1.* $[\![B_1; \ldots ; B_n]\!] \Longrightarrow C$

Unifies: one B_i with A_1

New subgoal: *1.* $[\![B_1; \ldots ; B_n; A]\!] \Longrightarrow C$

Forward proofs: frule and drule

apply(*frule* $rulename$)

Forward rule: $A_1 \Longrightarrow A$

Subgoal: 1. $[\![B_1; \ldots ; B_n]\!] \Longrightarrow C$

Unifies: one B_i with A_1

New subgoal: 1. $[\![B_1; \ldots ; B_n; A]\!] \Longrightarrow C$

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