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

\[ \tau ::= (\tau) \quad \mid \quad 'a \quad 'b \quad \ldots \quad \text{type variables} \]
Syntax:

\[
\tau ::= (\tau) \\
\mid 'a \mid 'b \mid \ldots \quad \text{type variables} \\
\mid \tau \Rightarrow \tau \quad \text{total functions}
\]
Syntax:

\[ \tau ::= (\tau) | 'a | 'b | \ldots \] type variables

\[ \tau \Rightarrow \tau \] total functions

\[ bool | nat | \ldots \] HOL base types
Syntax:

\[ \tau ::= (\tau) \]

\[ | \quad \text{'a} \quad \text{'b} \quad \ldots \quad \text{type variables} \]

\[ | \quad \tau \Rightarrow \tau \quad \text{total functions} \]

\[ | \quad \text{bool} \quad \text{nat} \quad \ldots \quad \text{HOL base types} \]

\[ | \quad \tau \times \tau \quad \text{HOL pairs (ascii: *)} \]
Syntax:

\[
\begin{align*}
\tau & ::= (\tau) \\
& \mid \ 'a \mid 'b \mid \ldots \quad \text{type variables} \\
& \mid \tau \rightarrow \tau \quad \text{total functions} \\
& \mid \text{bool} \mid \text{nat} \mid \ldots \quad \text{HOL base types} \\
& \mid \tau \times \tau \quad \text{HOL pairs (ascii: \ *} \\
& \mid \tau \text{ list} \quad \text{HOL lists}
\end{align*}
\]
Syntax

Syntax:

\[ \tau ::= (\tau) \mid \text{'a} \mid \text{'b} \mid \ldots \text{ type variables} \]
\[ \vdash \tau \mid \tau \Rightarrow \tau \text{ total functions} \]
\[ \text{bool} \mid \text{nat} \mid \ldots \text{ HOL base types} \]
\[ \tau \times \tau \text{ HOL pairs (ascii: \ast)} \]
\[ \tau \text{ list} \text{ HOL lists} \]
\[ \ldots \text{ user-defined types} \]
Syntax

Syntax:

\[ \tau ::= (\tau) \]

| \('a\) | \('b\) | ... | type variables
| \(\tau \Rightarrow \tau\) | total functions
| \(bool\) | \(nat\) | ... | HOL base types
| \(\tau \times \tau\) | HOL pairs (ascii: \(*\))
| \(\tau\ list\) | HOL lists
| ... | user-defined types

Parentheses: \(T1 \Rightarrow T2 \Rightarrow T3 \equiv T1 \Rightarrow (T2 \Rightarrow T3)\)
**Introducing new Types: typedefcl**

**typedefcl** *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)

\[ \text{term} ::= (\text{term}) \]
Syntax: (curried version)

\[ term ::= (term) \]
\[ \quad | \quad a \quad \text{constant or variable (identifier)} \]
Syntax: (curried version)

\[ \text{term} ::= (\text{term}) \]
\[ \quad | \quad a \quad \text{constant or variable (identifier)} \]
\[ \quad | \quad \text{term} \ \text{term} \quad \text{function application} \]
Syntax: (curried version)

\[
\text{term} ::= (\text{term}) \\
| \ a \quad \text{constant or variable (identifier)} \\
| \text{term} \ \text{term} \quad \text{function application} \\
| \lambda x. \ \text{term} \quad \text{function “abstraction”}
\]
Syntax

Syntax: (curried version)

\[ \text{term} ::= \ (\text{term}) \]
\[ \ | \ a \quad \text{constant or variable (identifier)} \]
\[ | \ \text{term} \ \text{term} \quad \text{function application} \]
\[ | \ \lambda x. \ \text{term} \quad \text{function "abstraction"} \]
\[ | \ \ldots \quad \text{lots of syntactic sugar} \]
Syntax

Syntax: (curried version)

\[ \text{term} ::= (\text{term}) \]
\[ \mid a \quad \text{constant or variable (identifier)} \]
\[ \mid \text{term} \ \text{term} \quad \text{function application} \]
\[ \mid \lambda x. \ \text{term} \quad \text{function “abstraction”} \]
\[ \mid \ldots \quad \text{lots of syntactic sugar} \]

Examples: \[ f (g \ x) \ y \quad h (\lambda x. f (g \ x)) \]
Syntax

Syntax: (curried version)

\[
\text{term} ::= \ (\text{term}) \\
\ | \ a \quad \text{constant or variable (identifier)} \\
\ | \ \text{term} \ \text{term} \quad \text{function application} \\
\ | \ \lambda x. \ \text{term} \quad \text{function “abstraction”} \\
\ | \ \ldots \quad \text{lots of syntactic sugar}
\]

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

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
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 $\implies (\implies)$
For separating premises and conclusion of theorems.
Connectives of the Meta-Logic

Implication \( \implies (==>) \)
For separating premises and conclusion of theorems.

Equality \( \equiv (==) \)
For definitions.
Connectives of the Meta-Logic

Implication $\implies (\implies)$
For separating premises and conclusion of theorems.

Equality $\equiv (\equiv)$
For definitions.

Universal quantifier $\forall (\forall)$
For parameters in goals.
Connectives of the Meta-Logic

Implication $\implies (\implies)$
For separating premises and conclusion of theorems.

Equality $\equiv (\equiv)$
For definitions.

Universal quantifier $\forall (\forall)$
For parameters in goals.

Do not use inside object-logic formulae.
Notation

\[ [A_1; \ldots ; A_n] \implies B \]

abbreviates

\[ A_1 \implies \ldots \implies A_n \implies B \]
Notation

\[ [ \ A_1 ; \ldots ; A_n \ ] \Rightarrow B \]

abbreviates

\[ A_1 \Rightarrow \ldots \Rightarrow A_n \Rightarrow B \]

; \approx \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 ⇒ nat
```

Definition:

```
defs
  sq_def: sq n ≡ n*n
```
**Definition (non-recursive)**

Declaration:

```plaintext
consts
  sq :: nat ⇒ nat
```

Definition:

```plaintext
defs
  sq_def: sq n ≡ n*n
```

Declaration + definition:

```plaintext
constdefs
  sq :: nat ⇒ nat
  sq n ≡ n*n
```
Proofs

General schema:

\textbf{lemma} \textit{name}: \textit{<goal>}

\begin{itemize}
  \item \textbf{apply} \textit{<method>}
  \item \textbf{apply} \textit{<method>}
  \item \textbf{<method>}
\end{itemize}

\texttt{done}
Proofs

General schema:

\textbf{lemma} \textit{name}: <\textit{goal}>

apply <\textit{method}>
apply <\textit{method}>

::
done

- Sequential application of methods until all \textit{subgoals} are solved.
The proof state

1. $\bigwedge x_1 \ldots x_p. \left[ A_1; \ldots ; A_n \right] \Rightarrow B$
2. $\bigwedge y_1 \ldots y_q. \left[ C_1; \ldots ; C_n \right] \Rightarrow D$
The proof state

1. $\forall x_1 \ldots x_p. [A_1; \ldots ; A_n] \implies B$
2. $\forall y_1 \ldots y_q. [C_1; \ldots ; C_n] \implies D$

$x_1 \ldots x_p$ Parameters
$A_1 \ldots A_n$ Local assumptions
$B$ Actual (sub)goal
Isabelle Theories
Theory = Source file

Syntax:

```
theory MyTh imports ImpTh_1 ... ImpTh_n 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
- ImpThᵢ: 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 LaTeX): \&and, \&or, ...
- via abbreviation: \&\&, \&/, \&\-->, ...

<table>
<thead>
<tr>
<th>x-symbol</th>
<th>&amp;forall</th>
<th>&amp;exists</th>
<th>&amp;lambda</th>
<th>&amp;not</th>
<th>&amp;&amp;</th>
<th>&amp;|</th>
<th>&amp;---&gt;</th>
<th>&amp;\Rightarrow</th>
</tr>
</thead>
<tbody>
<tr>
<td>ascii (1)</td>
<td>&amp;forall</td>
<td>&amp;exists</td>
<td>&amp;lambda</td>
<td>&amp;not</td>
<td>&amp;&amp;</td>
<td>&amp;|</td>
<td>&amp;---&gt;</td>
<td>&amp;\Rightarrow</td>
</tr>
<tr>
<td>ascii (2)</td>
<td>&amp;ALL</td>
<td>EX</td>
<td>%</td>
<td>~</td>
<td>&amp;</td>
<td>|</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(1) is converted to x-symbol, (2) stays ascii.
Demo: Isabelle theories
Natural Deduction
Rules

\[
\begin{align*}
A \land B & \quad \text{conjI} \\
A \lor B & \quad \text{disjI1/2} \\
A \rightarrow B & \quad \text{impl} \\
\end{align*}
\]

\[
\begin{align*}
A \land B & \quad \text{conjE} \\
A \lor B & \quad \text{disjE} \\
A \rightarrow B & \quad \text{impE} \\
\end{align*}
\]
Rules

\[ \frac{A \land B}{A \land B} \quad \text{conjI} \]

\[ \frac{A}{A \lor B} \quad \frac{B}{A \lor B} \quad \text{disjI1/2} \]

\[ \frac{A \implies B}{A \implies B} \quad \text{impl} \]

\[ \frac{A \land B}{C} \quad \text{conjE} \]

\[ \frac{A \lor B}{C} \quad \text{disjE} \]

\[ \frac{A \implies B}{C} \quad \text{implE} \]
Rules

\[
\frac{A \quad B}{A \land B} \quad \text{conjI} \quad \frac{A \land B \quad [A;B] \implies C}{C} \quad \text{conjE}
\]

\[
\frac{A}{A \lor B} \quad \frac{B}{A \lor B} \quad \text{disjI1/2} \quad \frac{A \lor B \quad A \implies C \quad B \implies C}{C} \quad \text{disjE}
\]

\[
\frac{A \implies B}{A \implies B} \quad \frac{A \implies B \quad A \quad B}{C} \quad \text{impl}\quad \frac{A \implies B \quad A \quad B}{C} \quad \text{impE}
\]
Proof by assumption

apply assumption

proves

1. $[ B_1; \ldots ; B_m ] \implies C$

by unifying $C$ with one of the $B_i$
Proof by assumption

**apply assumption**

proves

1. $\left[ B_1; \ldots; B_m \right] \Rightarrow C$

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 $\rightarrow$.

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

apply($\text{rule <intro-rule>}$)

Applying rule $[ A_1; \ldots ; A_n ] \implies A$ to subgoal $C$:

Unify $A$ and $C$
How to prove it by natural deduction

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

  \[ \text{apply}(rule \ <\text{intro-rule}>) \]

Applying rule $[ A_1; \ldots ; A_n ] \rightarrow A$ 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 $\Rightarrow$.

  ```apply(rule <intro-rule>)```

  Applying rule $[ A_1; \ldots ; A_n ] \Rightarrow A$ 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 $\Rightarrow$.

  ```apply(erule <elim-rule>)```
How to prove it by natural deduction

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

  \[ \text{apply}(\text{rule } <\text{intro-rule}>) \]

  Applying rule $[ A_1; \ldots ; A_n ] \implies A$ 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 $\implies$.

  \[ \text{apply}(\text{erule } <\text{elim-rule}>) \]

  Like \textit{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 $\forall, \exists, \ldots$: ends with $;$ or $\Rightarrow$
Scope

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

$$\land x \ y. \ [ \forall y. \ P \ y \ \rightarrow \ Q \ z \ y; \ Q \ x \ y ] \ \Rightarrow \ \exists x. \ Q \ x \ y$$
Scope

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

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

means

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

\[ \forall x. P x \quad \text{allI} \]

\[ \exists x. P x \quad \text{exI} \]

\[ \forall x. P x \]

\[ \exists x. P x \quad \text{exE} \]
Natural deduction for quantifiers

\[ \forall x. P x \quad \text{allI} \]
\[ \forall x. P x \quad \text{allE} \]
\[ \exists x. P x \quad \text{exI} \]
\[ \exists x. P x \quad \text{exE} \]
Natural deduction for quantifiers

\[ \forall x. P x \]

\[ \forall x. P x \quad \text{allI} \]

\[ \forall x. P x \quad P ?x \Longrightarrow R \quad \text{allE} \]

\[ \exists x. P x \]

\[ \exists x. P x \quad \exists x. P x \Longrightarrow R \quad \text{exE} \]

\[ \exists x. P x \quad \text{exI} \]

\[ \exists x. P x \quad \text{exE} \]
Natural deduction for quantifiers

\[ \forall x. P x \quad \text{allI} \]
\[ \forall x. P x \quad P ?x \Rightarrow R \quad \text{allE} \]
\[ P ?x \quad \exists x. P x \quad \text{exI} \]
\[ \exists x. P x \quad \exists x. P x \Rightarrow R \quad \text{exE} \]

- allI and exE introduce new parameters (\(\forall x\)).
- allE and exI introduce new unknowns (\(?x\)).
Instantiating rules

apply(rule_tac x = "term" in rule)

Like \textit{rule}, but \(?x\) in \textit{rule} is instantiated by \textit{term} before application.
Instantiating rules

\[\text{apply}(\text{rule_tac } x = "term" \text{ in } \text{rule})\]

Like \textit{rule}, but \texttt{?x} in \textit{rule} is instantiated by \textit{term} before application.

Similar: \texttt{erule_tac}
Instantiating rules

\texttt{apply}(\texttt{rule_tac x = "term" in rule})

Like \textit{rule}, but \( ?x \) in \textit{rule} is instantiated by \textit{term} before application.

Similar: \texttt{erule_tac}

\( \neg x \) is in \textit{rule}, not in the goal \( \neg \)
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: \[ A_1 \implies A \]
Subgoal: \[ 1. \[ B_1; \ldots ; B_n \] \implies C \]
Forward proofs: frule and drule

apply(frule rulename)

Forward rule: \[ A_1 \implies A \]

Subgoal: \[ 1. \[ B_1; \ldots ; B_n \] \implies C \]

Unifies: one \( B_i \) with \( A_1 \)

New subgoal: \[ 1. \[ B_1; \ldots ; B_n; A \] \implies C \]
Forward proofs: frule and drule

apply(frule rulename)

Forward rule: \( A_1 \implies A \)
Subgoal: \[ B_1; \ldots ; B_n \] \implies C

Unifies: one \( B_i \) with \( A_1 \)
New subgoal: \[ B_1; \ldots ; B_n; A \] \implies C

apply(drule rulename)

Like frule but also deletes \( B_i \)
Demo: quantifier proofs