The Isabelle System Framework
What is Isabelle as a System?

• A Document Processor
  • ... where documents have a unique name
  • ... may acyclicly import documents
  • ... and consists of an command sequence
  • ... where new commands may be introduced on the fly (i.e. the system framework is extensible).
  • A session (a collection of documents organized in a hierarchy) may be “frozen” to a session (or configuration)
  • A session is evaluated concurrently and asynchronisly on all what the “user sees”, its jEdit editor is an IDE
What is Isabelle as a System?

- Global View of a “session”

Diagram:

- Document / “Theory”
- A
  - cmd
  - cmd
  - cmd
- B
  - cmd
  - cmd
  - cmd
  - cmd
- C
  - cmd
  - cmd
  - cmd
  - cmd
- D
  - cmd
What is Isabelle as a System?

- Global View

Document / “Theory”

Atom detailed view:
What is Isabelle as a System?

- Document “positions” were evaluated to an implicit state, the theory context $\Theta$

```
Document / “Theory”
```

```
A
  cmd
  cmd
  cmd
C
  cmd
  cmd
  cmd
B
  cmd
  cmd
  cmd
D
  cmd
```

“semantic” evaluation as SML function

$\Theta_0$
$\Theta_{3-1}$
$\Theta_{3-2}$
$\Theta_3$
What is Isabelle as a System?

- Document “positions” were evaluated to an implicit state, the theory context $\Theta$

Note:
The theory context $\Theta$ can contain a “type signature” and a “term signature”, “axioms”, but also system configuration information etc.

“semantic” evaluation as SML function
What is Isabelle as a System?

- Document “positions” were evaluated to an implicit state, the theory context $\Theta$

Note:
- ... and this is at the beginning a minimal intuitionistic logic called in Isabelle/Pure

“semantic” evaluation as SML function
What is Isabelle as a System?

• Example

theory D
imports B C
begin

section{* First Section *}
text{* Some mathematical text: @{text \<alpha>}. *}
ML{* fun fac x = if x = 0 then 1 else x*fac(x-1) *}
ML{* fac 10 *}
end
What is Isabelle as a System?

- Example

```plaintext
theory D
imports B C
begin

section{* First Section *}

text{* Some mathematical text: @{text \alpha}. *}

ML{* fun fac x = if x = 0 then 1 else x*fac(x-1) *}

ML{* fac 10 *}

end
```
What is Isabelle as a System?

• Example

theory D
imports B C
begin

section{* First Section *}

text{* Some mathematical text: @{text \(<alpha>\).*}

ML{* fun fac x = if x = 0 then 1 else x*fac(x-1) *}

ML{* fac 10 *}
end

“fac” visible here because the ML environment is part of Θ !!
Demo I

- Start Isabelle (via the PIDE jEdit)
- Browse „demo1.thy“
- Commands:
  - text, section, subsection
  - ML
  - value
  - a browser for theorems: find_theorems
- Capabilities:
  - hovering, jump-link,
Demo I

transcription, so $\alpha$ is just equal to $\mu$ but can also be written $\alpha$.

Only in few cases one has to memorize. For them, ASCII - oriented shortcuts like $\Rightarrow$ can be given for $\rightarrow$.

substitution{"Apotheosis"}

text{"It may be necessary to get used to the PIDE - Paradigm: always checking whenever typing. After a while, however, one gets used to it. Don't forget to save from time to time !!!"}

substitution{"The Function" in SML}

ML{"fun fac n = if n=0 then 1 else n * fac(n-1)"}

ML{"fac 50"}

substitution{"Using the code-generator to SML"}

value '{2::nat} + 2'

val it =

304140932017337804361260581655064768844377641568960512000000000000000000: int
Demo I

Main (Editing) Panel

transcription, so \alpha is in \textit{roman}, \textit{but} can also be written \textit{roman}.

Only in few cases one has to memorize. For them, ASCII-oriented shortcuts like $\rightarrow$ can be given for $\Rightarrow$.

\begin{verbatim}
subsection{* Apotheosis *}

\textit{text}[* It may be necessary to get used to the PIDE - Paradigm: always checking whenever typing. After a while, however, one gets used to it. Don’t forget to save from time to time !!! *]

subsection{* The Function in SML *}

ML[* fun fac n = if n=0 then 1 else n * fac(n-1) *]

subsection[* Using the code-generator to SML *]

value '[(2::nat) + 2]
\end{verbatim}
transcription, so $\alpha$ is just equal to $<\alpha>$ but can also be written as $\alpha$.

Only in few cases one has to memorize. For them, ASCII-oriented shortcuts like $\Rightarrow$ can be given for $\rightarrow$.

subsection{* Apotheosis *}

text[* It may be necessary to get used to the PIDE - Paradigm: always checking whenever typing. After a while, however, one gets used to it. Don’t forget to save from time to time !!! *]

subsection{* ‘The Function’ in SML *}

ML[* fun fac n = if n=0 then 1 else n * fac(n-1) *]
ML[* fac 50*]

subsection[* Using the code-generator to SML *]

value ‘[2::nat] + 2’
Demo I

transcription, so $\alpha$ is just equal to $\langle\alpha\rangle$ but can also be written $\alpha$.

Only in few cases one has to memorize. For them, ASCII - oriented shortcuts like $\Rightarrow$ can be given for $\Rightarrow$.

subsection{* Apothesis *}

text[* It may be necessary to get used to the PIDE - Paradigm: always checking whenever typing. After a while, however, one gets used to it. Don’t forget to save from time to time !!! *]

subsection{* 'The Function' in SML *}

ML[* fun fac n = if n=0 then 1 else n * fac(n-1) *]

subsection{* Using the code-generator to SML *}

value '[(2::nat) + 2]
Parallel Nano-Kernel LCF-Architecture in the jEdit - GUI (PIDE)

fine-grained, asynchronous parallelism (Isabelle2009-2)
What is Isabelle as a System?

• Example with definitions and proofs:

```plaintext
theory Test
imports Main (* = HOL Library *)
begin

definition H : "bool \longrightarrow bool \longrightarrow bool"
where "H x y == (x \or y) \and (x \noteq y)"

lemma <SomeName> : "A \and B \longrightarrow B"
<\texttt{tactical proof or declarative proof}>
done
```
What is Isabelle as a System?

- The jEdit - IDE will parse and print this to:

```isar
theory Test
imports Main (* = HOL Library *)
begin

definition H : "bool ⇒ bool ⇒ bool"
where "H x y == (x ∨ y) ∧ x ≠ y"

lemma <SomeName> : "A ∧ B → B"
<tactical proof or declarative proof>
done

Use completion and tooltips!
```
Revision: Pure Syntax
(the syntax for "rule" formation)

• Example: The language "Pure":
  \[ \Sigma_{\text{Pure}} = \{ \text{(all, } (\alpha \to \text{Prop}) \to \text{Prop}), \quad (* \text{ !! *)} \]
  \[ (\_ \iff \_, \text{Prop} \to \text{Prop} \to \text{Prop}), \quad (* \text{ ==> *)} \]
  \[ (\_ \equiv \_, \alpha \to \alpha \to \text{Prop}) \} \quad (* \text{ == *)} \]

• Note that we use schematic type variables to denote conceptually infinite signatures:
  \[ (\_ \equiv \_, \text{Prop} \to \text{Prop} \to \text{Prop}), \quad (\_ \equiv \_, \text{bool} \to \text{bool} \to \text{Prop}), \]
  \[ (\_ \equiv \_, \text{nat} \to \text{nat} \to \text{Prop}), \ldots \]

• Caveat: Isabelle uses \( \Rightarrow \) instead of \( \to \) in types, sorry for the confusion.
Simple Proof Commands

• Simple (Backward) Proofs:

\[
\text{lemma } \langle \text{thmname} \rangle : \ \\
[\langle \text{contextelem} \rangle^+ \text{ shows}] \ \langle \text{phi} \rangle \\
\langle \text{proof} \rangle
\]

There are different formats of proofs, we concentrate on the simplest one:

\[
\text{apply(}\langle \text{method}_1 \rangle \rangle \ldots \text{apply(}\langle \text{method}_n \rangle \rangle \ \text{done}
\]
Simple Proof Commands

• Simple (Backward) Proofs:

```
lemma <thmname> : 
[<contextelem>^+ shows] "<phi>"
<proof>
```

element:

```
lemma m : "conc (Seq a (Seq b Empty)) (Seq c Empty) = 
    Seq a (Seq b (Seq c Empty))"
apply(simp) done
```

This type of proof evolves “bottom up” from the conclusion to
the assumptions.
apply(bla) done is syntactically equivalent to by bla.
A Summary of Proof Methods

• The most elementary proof method is the rule <thmname> method. It is used for introduction rules. It proceeds in three phases:
  – lifting of <thmname> over the parameters of the current (first) goal (fiddling with quantifiers)
  – lifting of <thmname> over the assumptions of the current (first) goal (see pp. 25)
  – constructing an instance of <thmname> by unification; this means that the conclusion of <thmname> must finally match (modulo β and α red.) against the conclusion of the current (first) goal.

• The user can help this process by using the variant:
  – rule_tac <subst> in <thmname>
  – … where <subst> is of the form:
    \[ x_1 = \phi_1 \text{ and } x_n = \phi_n \]
    and the \( x_i \) are the variables of <thmname>
A Summary of Proof Methods

- An important variant is erule <thmname> method. It is used for elimination rules. It proceeds in three phases:
  - lifting of <thmname> over the assumptions of the current (first) goal (see pp. 25)
  - lifting of <thmname> over the parameters of the current (first) goal (fiddling with quantifiers)
  - constructing an instance of <thmname> by unification; this means that the conclusion of <thmname> must finally match (modulo β and α red.) against the conclusion of the current (first) goal, moreover, the first premise of <thmname> must match (modulo β and α red.) against one of the assumptions of the current goal.

- The user can help this process by using the variant:
  - erule_tac <subst> in <thmname>
A Summary of Proof Methods

• An important method the assumption method.

It is used for final situations, where the conclusion of a goal can be discharged by one of the assumptions. It suffices that one of the assumptions match (modulo \( \beta \) and \( \alpha \) red.) against the conclusion.
At a Glance

- **low-level methods (without substitution)**
  - assumption (unifies conclusion vs. a premise)
  - subst `<thmname>`
    - does one rewrite-step
      - (by instantiating the HOL subst-rule)
  - `rule[_tac <subst> in] <thmname>`
    - PROLOG - like resolution step using HO-Unification
  - `erule[_tac <subst> in] <thmname>`
    - elimination resolution (for ND elimination rules)
  - `drule[_tac <subst> in] <thmname>`
    - destruction resolution (for ND destruction rules)