Isabelle Tutorial:
System, HOL and Proofs

Burkhart Wolff

Université Paris-Sud

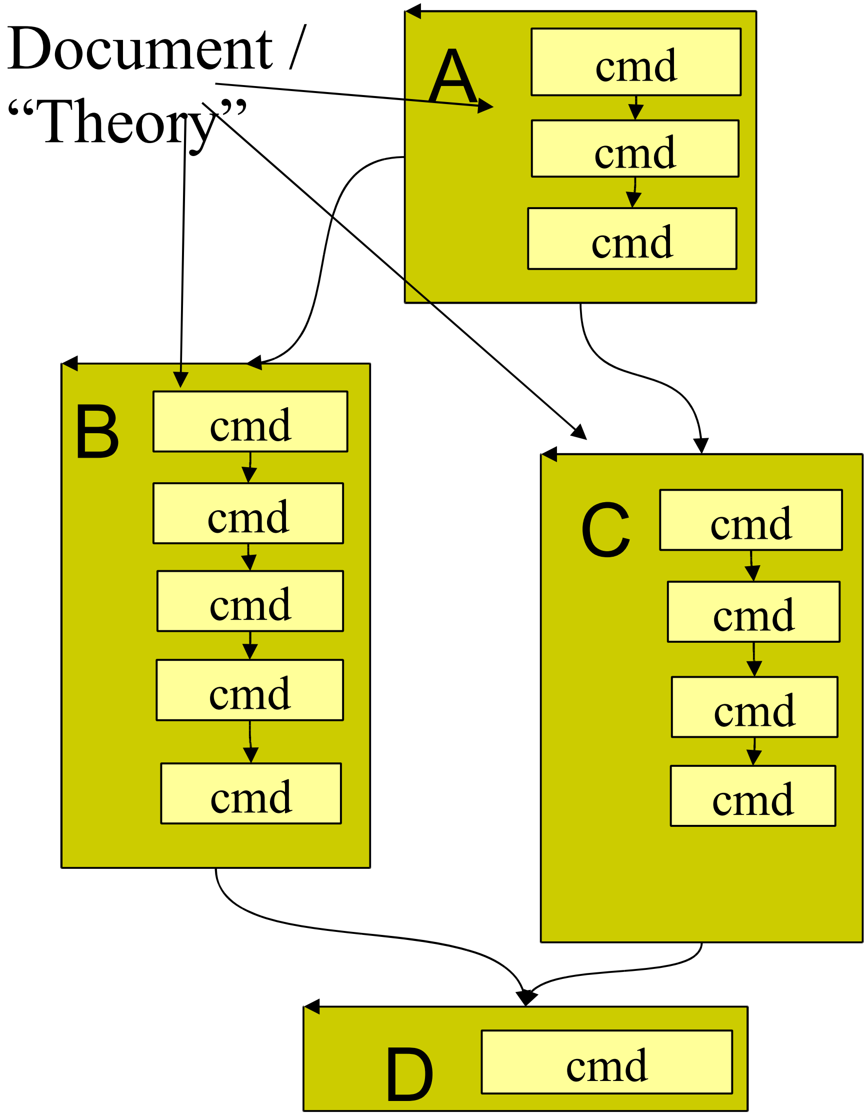
The Isabelle System Framework

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 hierachy) 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

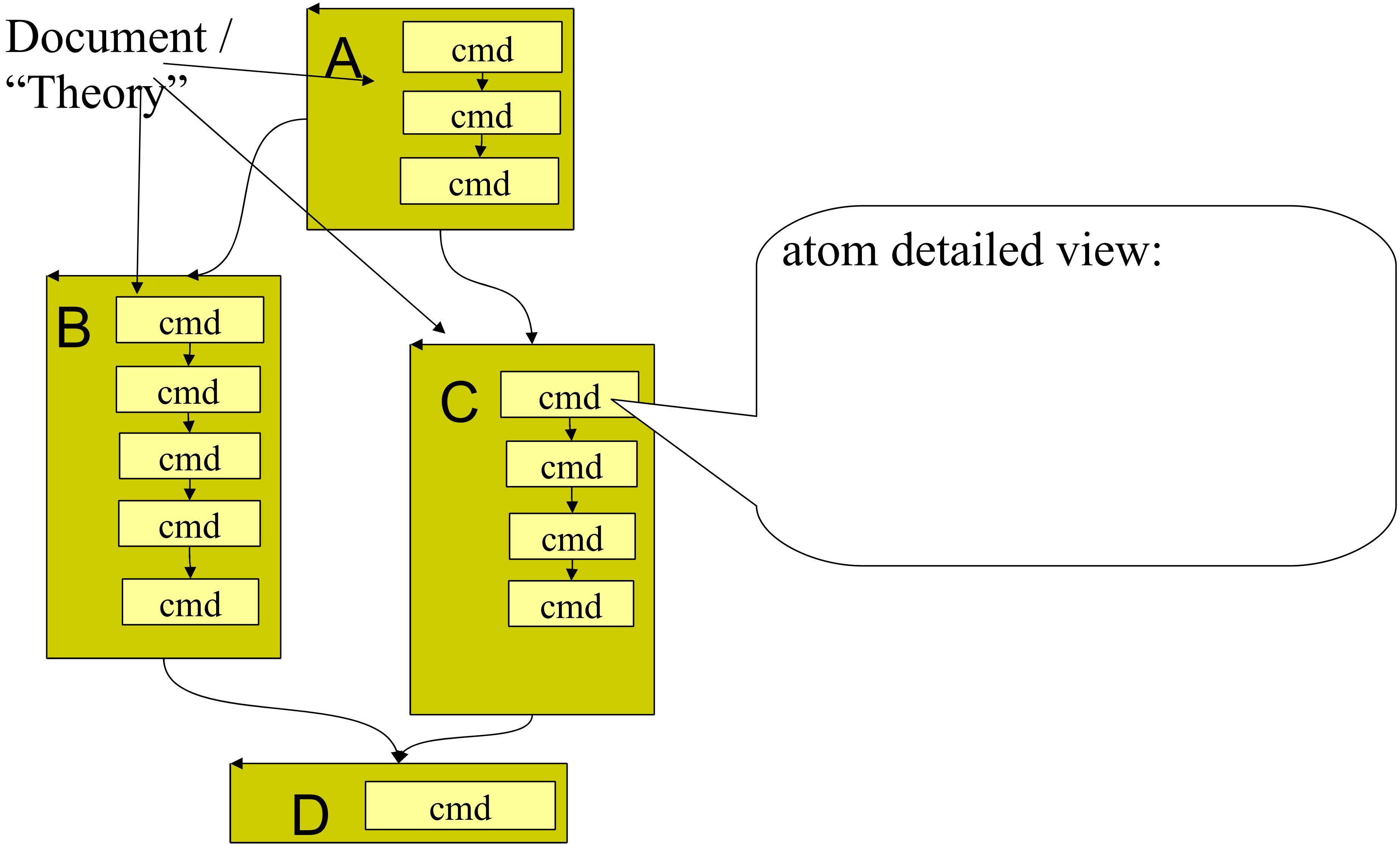
What is Isabelle as a System ?

- Global View of a “session“



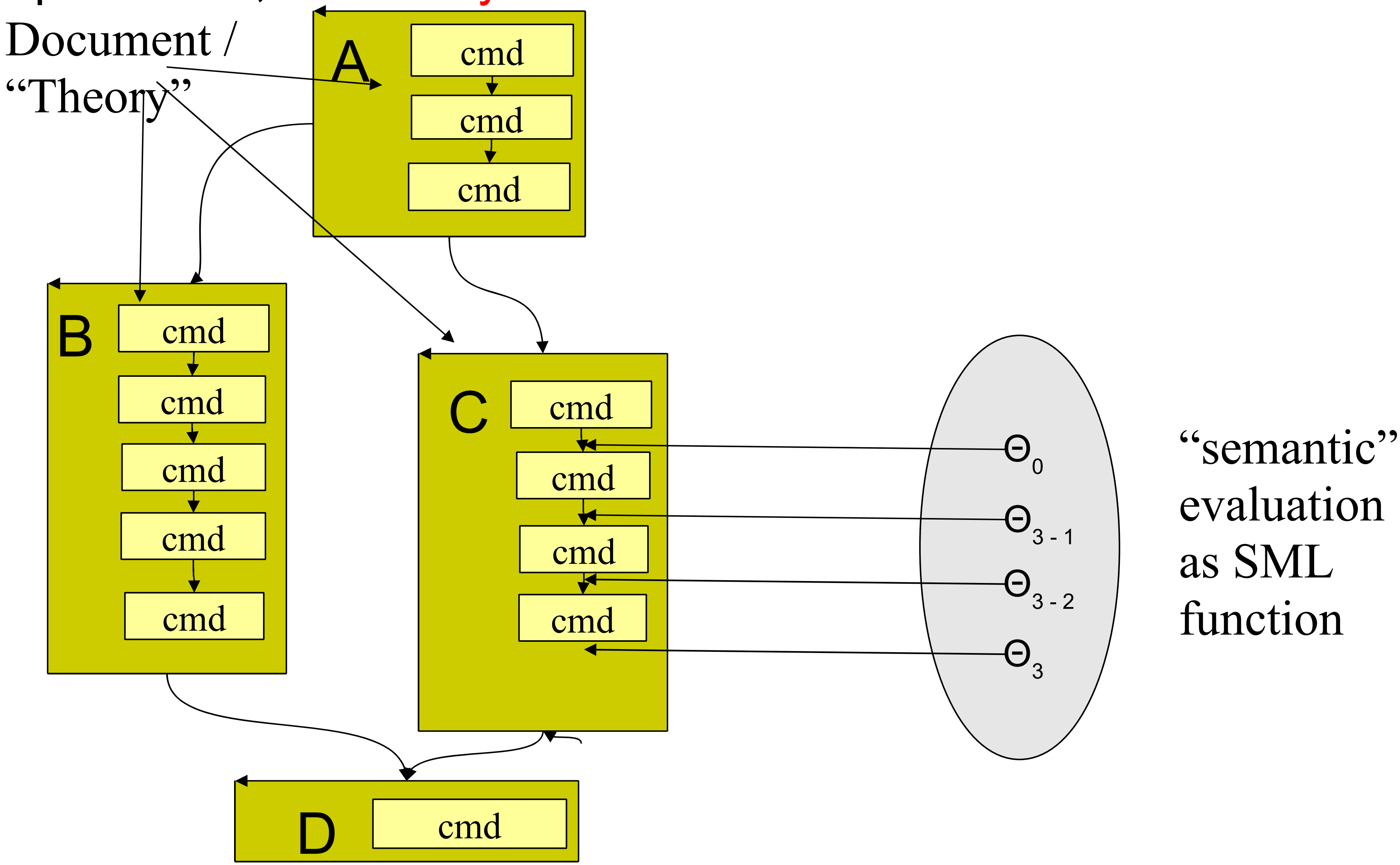
What is Isabelle as a System ?

- Global View



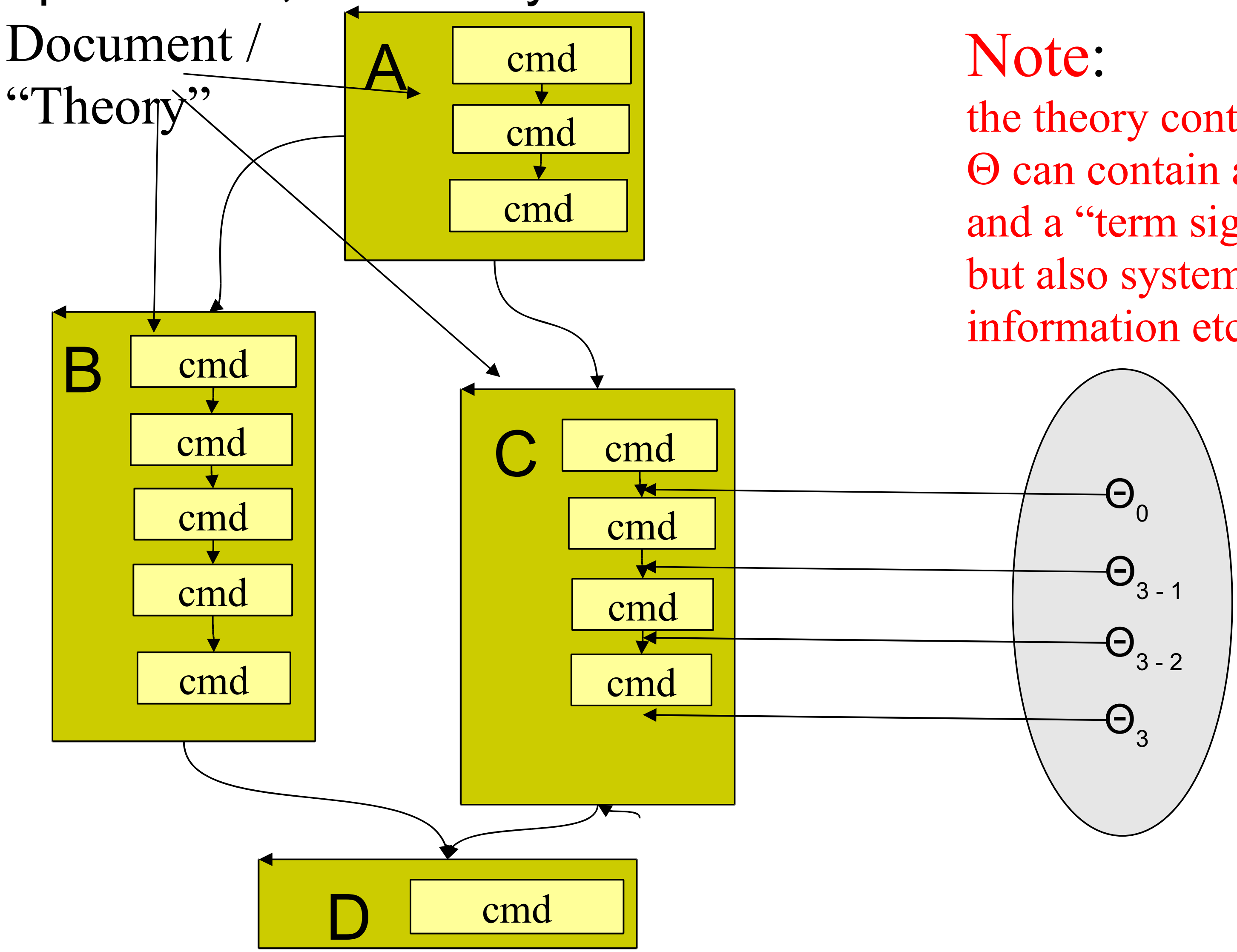
What is Isabelle as a System ?

- Document “positions” were evaluated to an implicit state, the theory context Θ



What is Isabelle as a System ?

- Document “positions” were evaluated to an implicit state, the theory context Θ



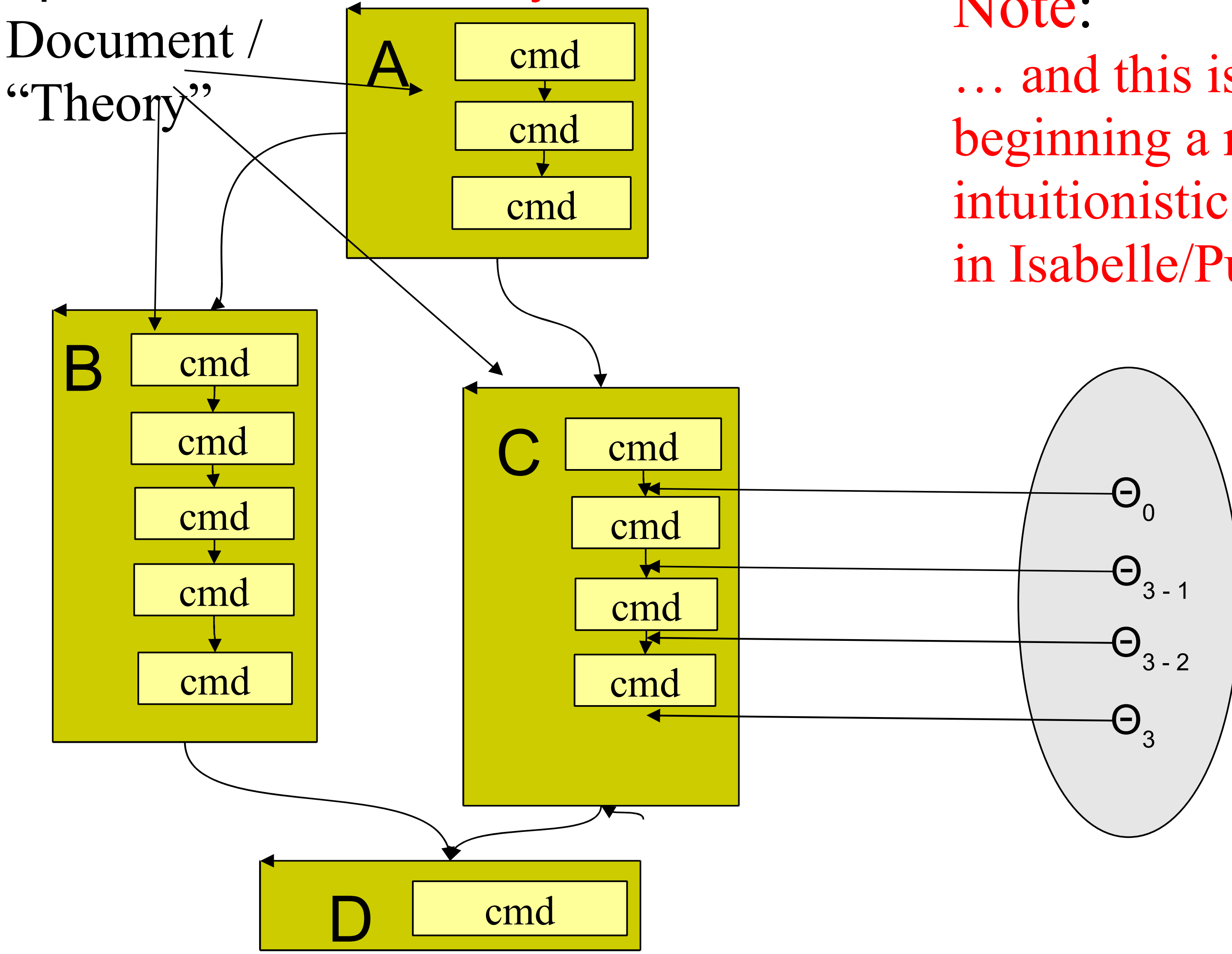
Note:
the theory context
Θ can contain a “type signature” and a “term signature”, “axioms”, but also system configuration information etc.

“semantic” evaluation as SML function

What is Isabelle as a System ?

- Document "positions" were evaluated to an implicit state, the theory context Θ

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

```
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

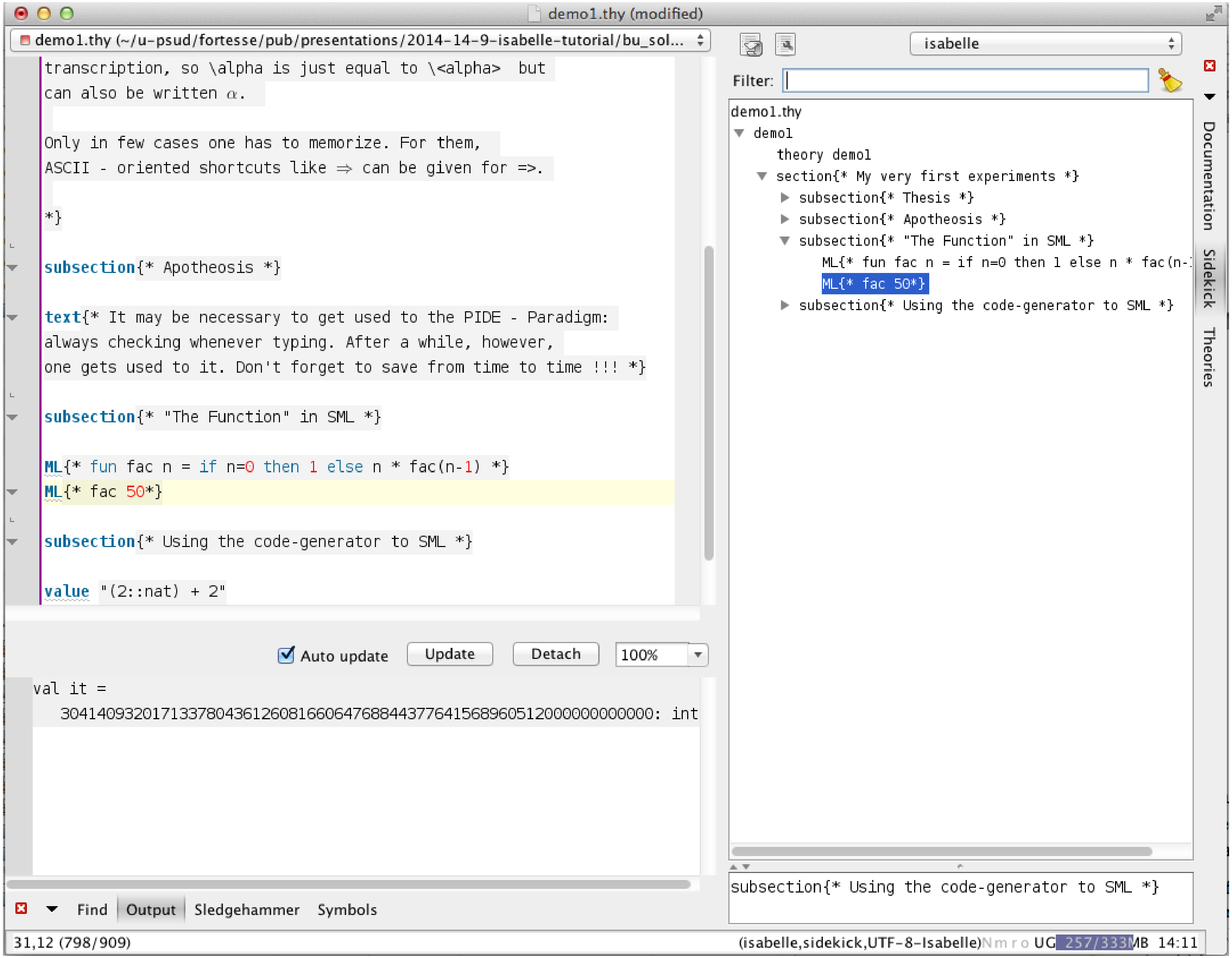
```
theory D
imports B C
begin
section{* First Section *}
text{* Some mathematical text: @{text \<alpha>}.*}
ML{* fun fac x = if x = 0 then 1 else x*fac(x-1) *}
ML{* fac 10 *}
end
```

"fac" visible here because the ML environment is part of Θ !!

Demo I

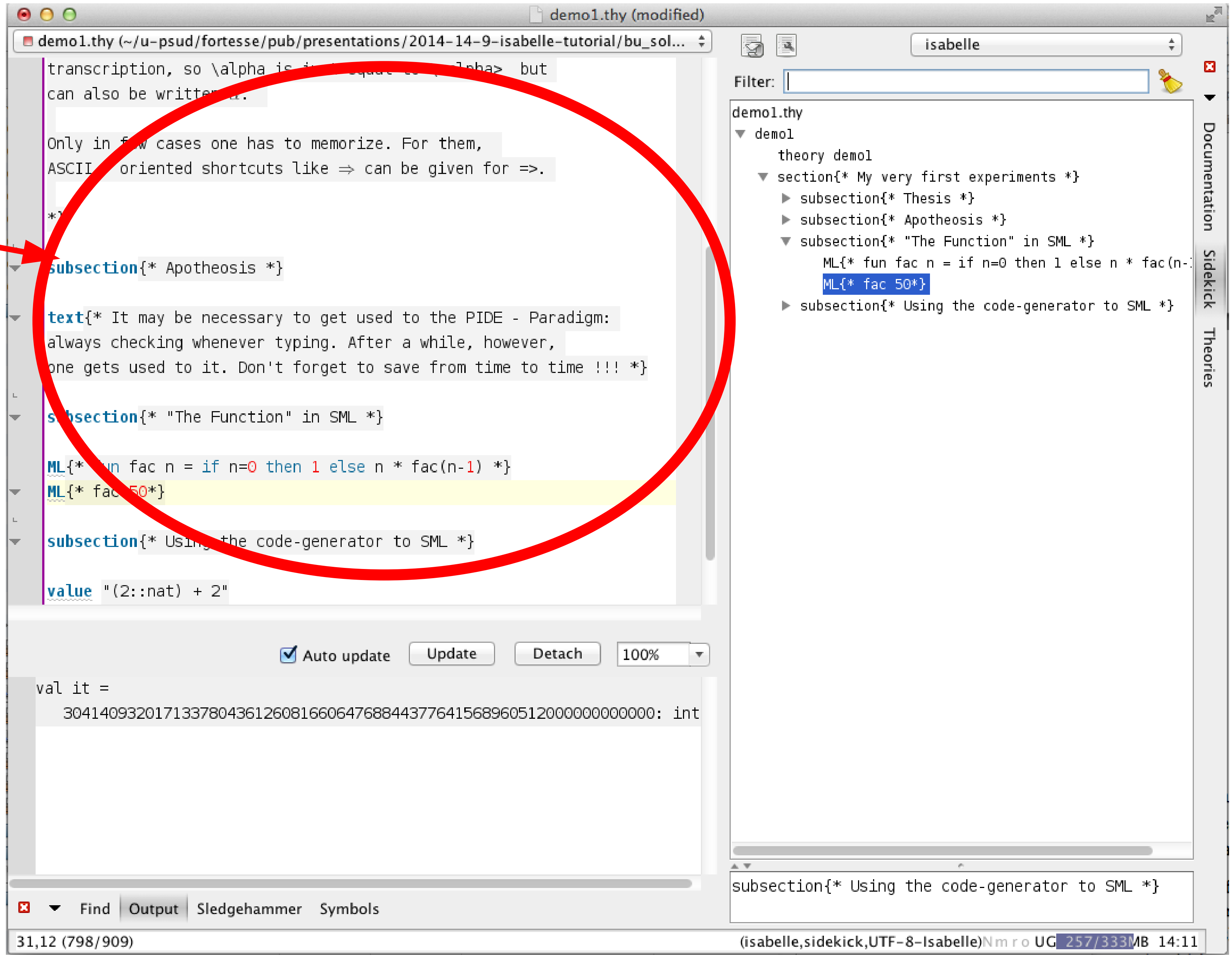
- Start Isabelle (via the PIDE jEdit)
- Browse „demo1.thy“
- Commands:
  - text, section, subsection
  - ML
  - value
  - a browser for theorems: find_theorems
- Capabilities:
  - hovering, jump-link,

Demo I

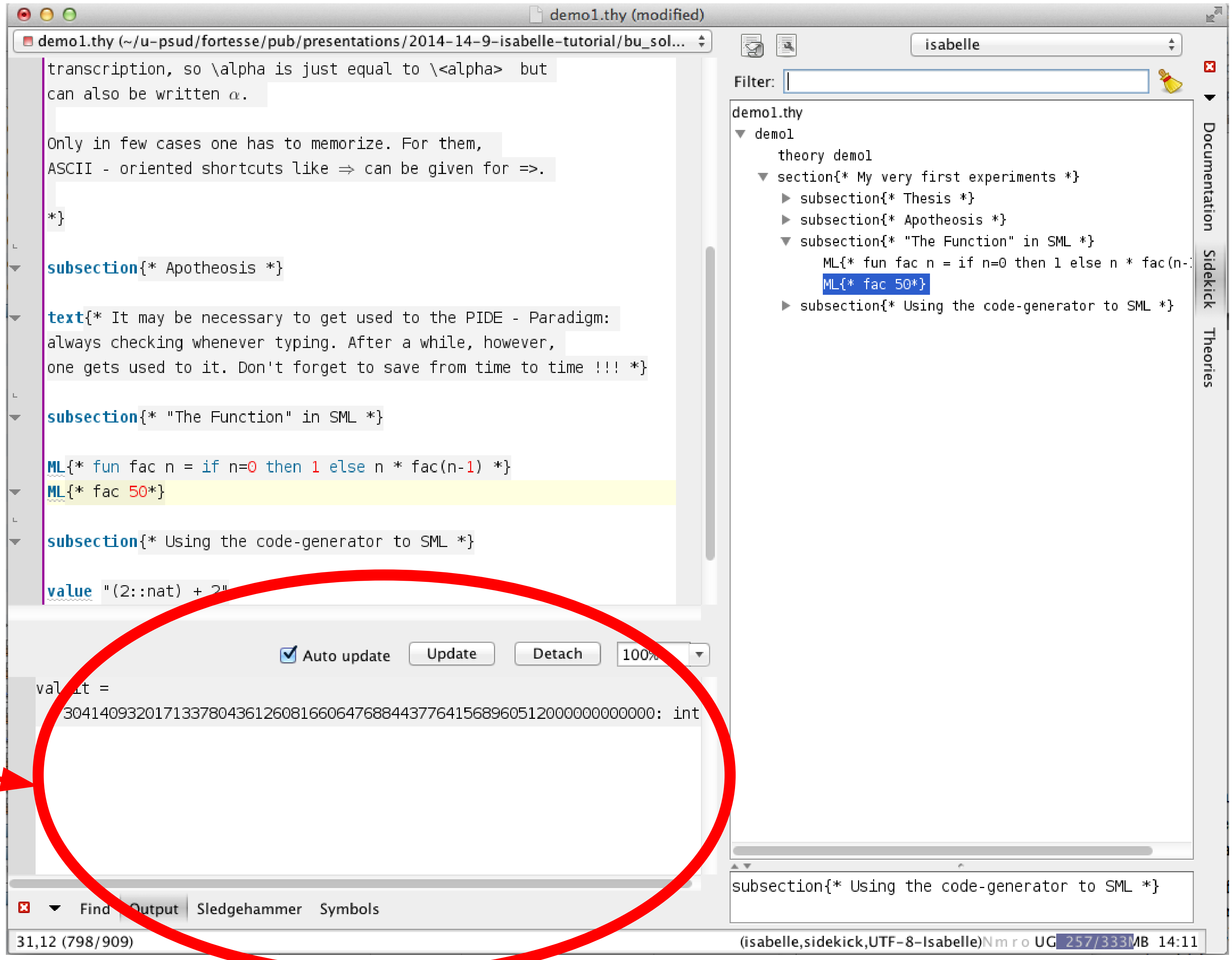

demo1.thy (modified)
demo1.thy (~/u-psud/fortesse/pub/presentations/2014-14-9-isabelle-tutorial/bu_sol...
transcription, so \alpha is just equal to \<alpha> but
can also be written α.
Only in few cases one has to memorize. For them,
ASCII - oriented shortcuts like ⇒ can be given for =>.
*}
subsection{* 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"
Auto update
Update
Detach
100%
val it =
30414093201713378043612608166064768844377641568960512000000000000: int
Find
Output
Sledgehammer
Symbols
31,12 (798/909)
isabelle
Filter:
demo1.thy
demo1
theory demo1
section{* My very first experiments *}
subsection{* Thesis *}
subsection{* Apotheosis *}
subsection{* "The Function" in SML *}
ML{* fun fac n = if n=0 then 1 else n * fac(n-
ML{* fac 50*}
subsection{* Using the code-generator to SML *}
subsection{* Using the code-generator to SML *}
Documentation
Sidekick
Theories
(isabelle,sidekick,UTF-8-Isabelle)Nmro UG 257/333MB 14:11

Demo I

Main (Editing) Panel

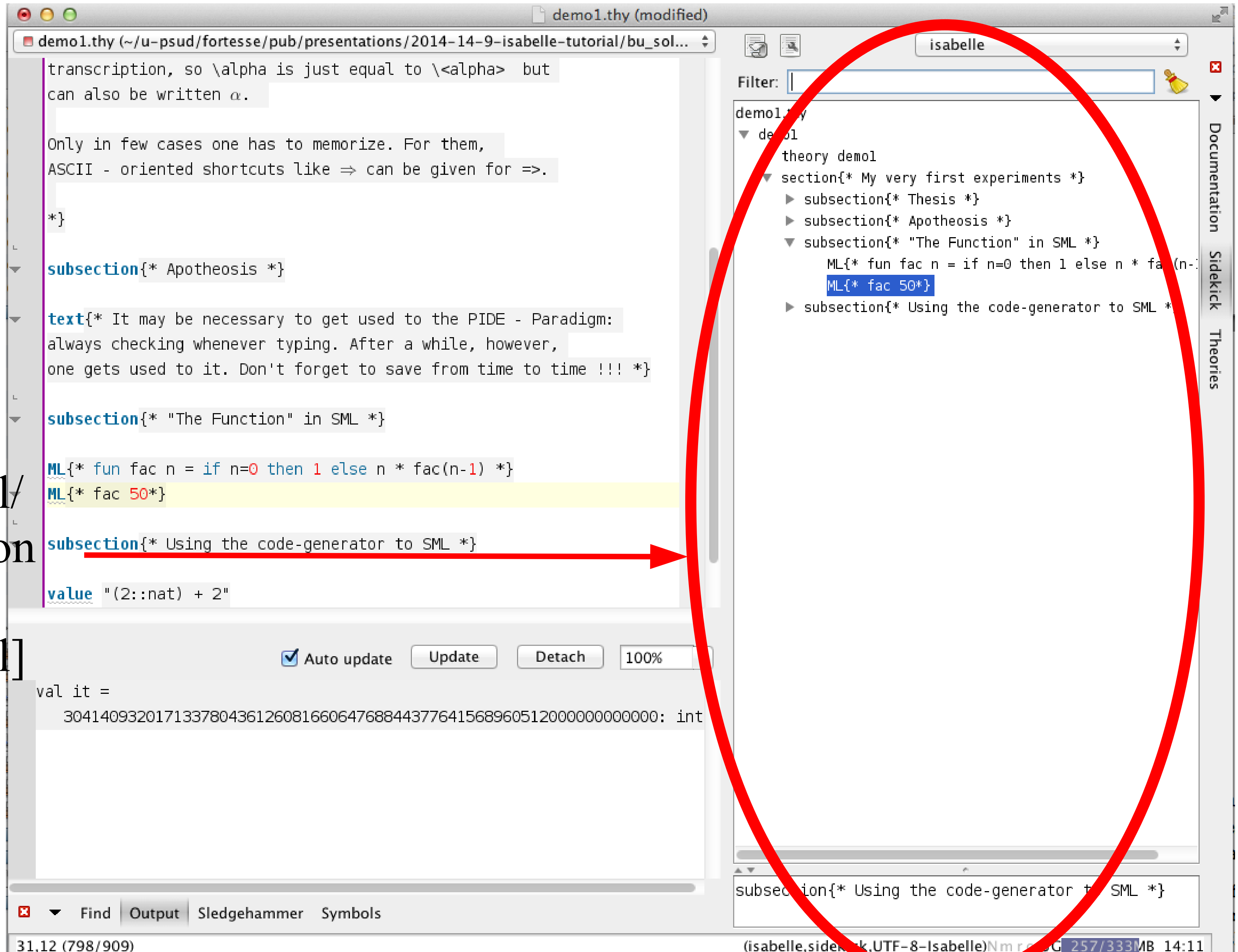


Demo I



Output Panel

Demo I

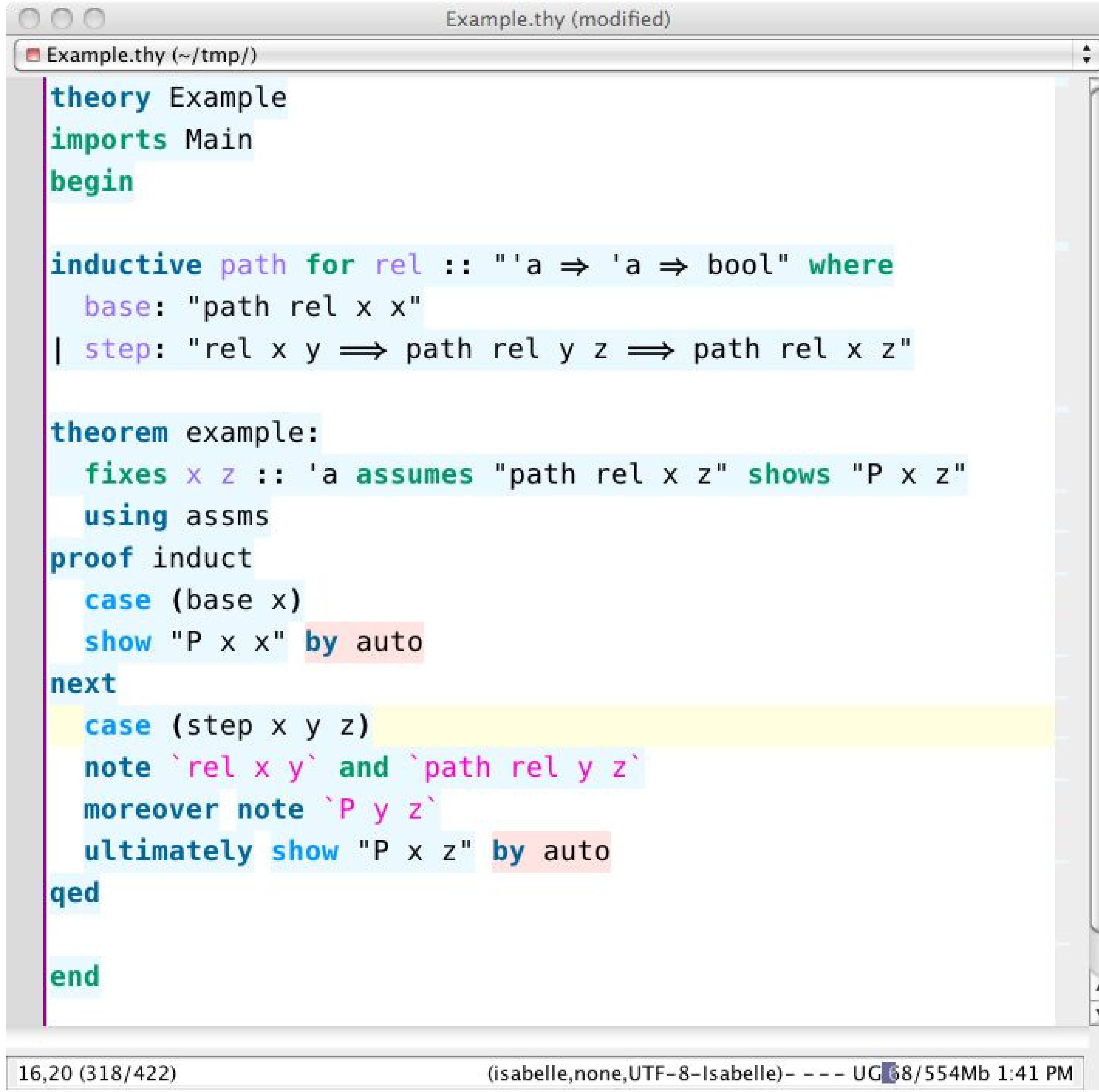


Sidekick Panel/
[Documentation
Panel |
Theories Panel]

Parallel
Nano-Kernel
LCF-Archi-
tecture

in the

jEdit - GUI
(PIDE)



fine-grained,
asynchronous
parallelism
(Isabelle2009-2)

What is Isabelle as a System ?

- Example with definitions and proofs:

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

```
definition H : “bool \<Rightarrow> bool \<Rightarrow> bool”
where  “H x y == (x \<or> y) \<and> (x \<noteq> y)”
```

```
lemma <SomeName> : “A \<and> B \<longrightarrow> B”
<tactical proof or declarative proof>
done
```

What is Isabelle as a System ?

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

```
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“:

  $\Sigma_{Pure} = \{$ (all, $(\alpha \to Prop) \to Prop$), (* !! *)

  ($\_ \Longrightarrow \_$, $Prop \to Prop \to Prop$), (* ==> *)

  ($\_ \equiv \_$, $\alpha \to \alpha \to Prop$)$\}$ (* == *)

- Note that we use schematic type variables to denote conceptually infinite signatures :

  ($\_ \equiv \_$, $Prop \to Prop \to Prop$), ($\_ \equiv \_$, $bool \to bool \to Prop$),

  ($\_ \equiv \_$, $nat \to nat \to Prop$), ...

- Caveat: Isabelle uses $\Rightarrow$ instead of $\to$ in types, sorry for the confusion.

Simple Proof Commands

- Simple (Backward) Proofs:

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

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

apply(<$method_1$>) … apply(<$method_n$>) done

Simple Proof Commands

- Simple (Backward) Proofs:

```
lemma  <thmname> :
  [<contextelem>+ shows] “<phi>”
   <proof>
```

example:

```
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="ϕ_1" and x_n="ϕ_n"

    and the xi 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 β and α 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)