Preuves Interactives et Applications

Burkhart Wolff

http://www.lri.fr/~wolff/teach-material/2020-2021/M2-CSMR

Université Paris-Saclay

Foundations: HOL Semantics and Specification Constructs

Overview

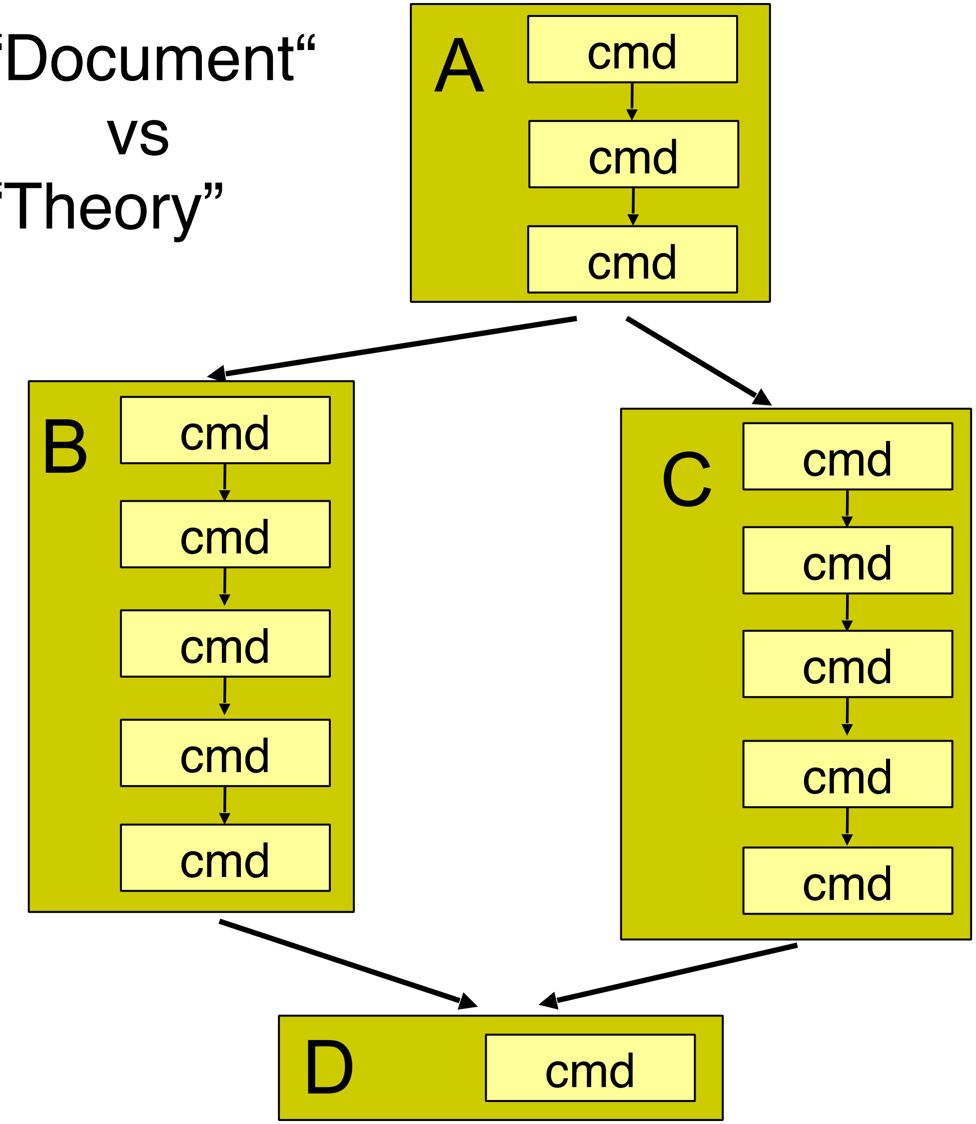
- Front-End: Isabelle's Document Model
- Back-End: Global/Local Contexts
- HOL Semantics and Foundations
- Conservative Extensions of Contexts
- Specification Constructs in Isabelle/HOL
- More on Proof Automation

Isabelle Document Model
and
Global/Local Contexts

What is Isabelle as a System ?

- Global View of a "session"

"Document"
vs
"Theory"

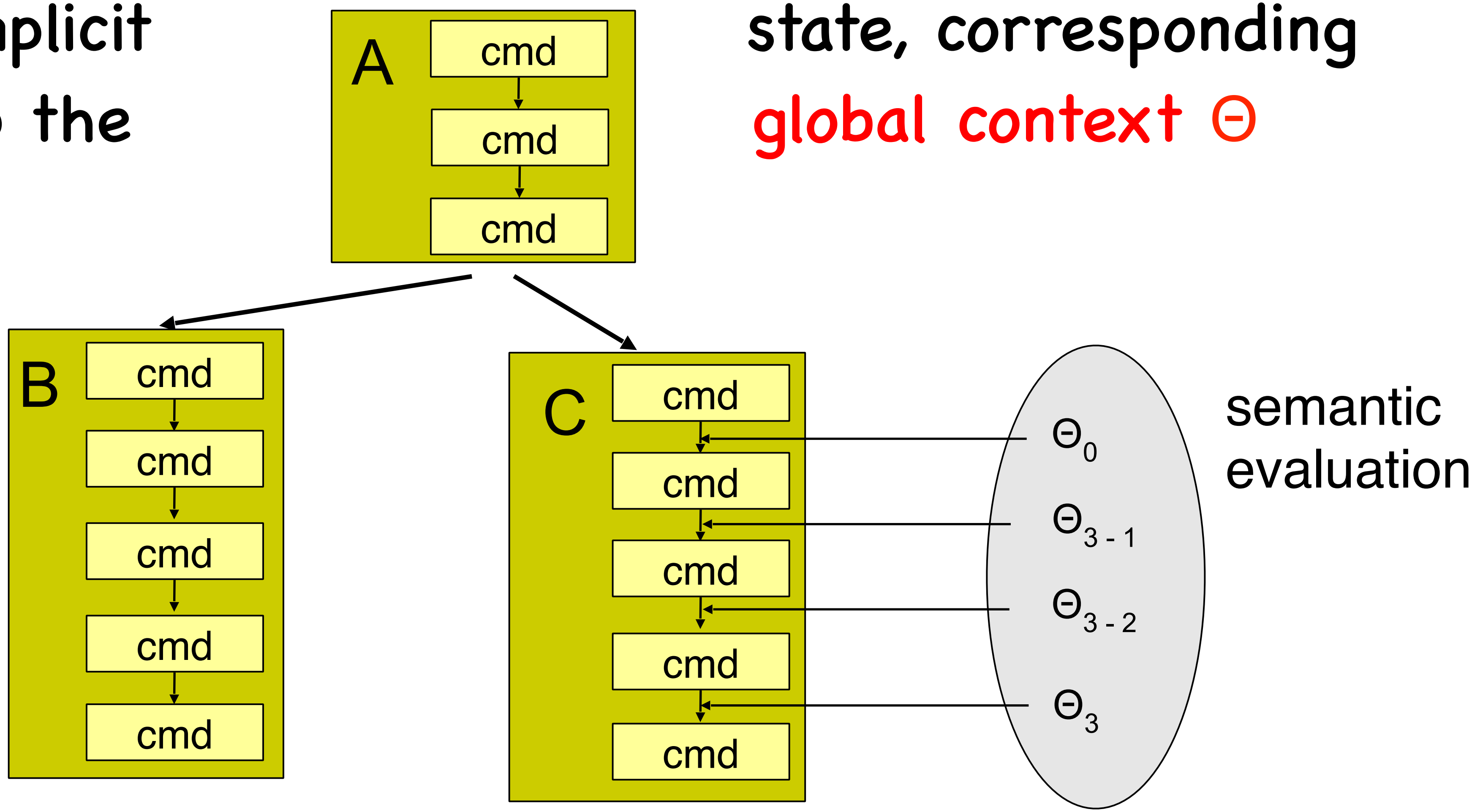


Revision: Documents and Commands

- Each position in document corresponds
  - to a "global context" Θ (containing a signature Σ and a set of axioms A)
  - to a "local context" Θ, Γ
  - [reminder] composing a thm $\Gamma \vdash_{\Theta} \varphi$
- There are specific „Inspection Commands" that give access to information in the contexts
  - thm, term, typ, value, prop : **global context**
  - thm, print_cases, facts, ... , : **local context**

What is Isabelle as a System ?

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



Commands for Basic Theory Extensions

- Isabelle has (similar to Eclipse) a „document-centric" view of development: there is a notion on an entire "project" which is processed globally.

- Documents (projects in Eclipse) consists of files (with potentially different file-type); .thy files consists of headers commands.

- A Document Configuration is specified in ROOT file

Theory Extensions and Global/Local Contexts

Commands for Basic Theory Extensions

- **Type Declaration**

typedecl "$(\alpha_1,\ldots,\alpha_n)$<typconstructor-id>"

example: typedecl "L"

- **(Unspecified) Constant Declaration:**

consts c :: „τ"

example: consts True :: "bool"

Commands for Basic Theory Extensions

- Constant Declaration "Semantics":

$$(\Sigma, A) \text{ "}\in\text{" } \Theta$$

$$\Downarrow$$

consts c :: „τ"

$$(\Sigma \oplus (c \mapsto \tau) , A) \text{ "}\in\text{" } \Theta'$$

- where the constant c is "fresh" in S

Commands for Basic Theory Extensions

- Constant Declaration "Semantics":

$$(\Sigma, A) \text{ ``}\in\text{''} \; \Theta$$

axiomatization c :: „τ"
where <name> : "<prop>"

$$(\Sigma , A \oplus (\langle name\rangle \mapsto \langle prop\rangle)) \text{ ``}\in\text{''} \; \Theta'$$

- where the constant c may be arbitrary.

Foundation:

Introduction to HOL Semantics

A Critique on Axioms

- In general, theory extensions are problematic
- In particular, axioms are extremely dangerous.
  Consider:

```
axiomatization  Y :: „('α ⇒ 'α) ⇒ 'α“
    where  rec : “Y f = f(Y f)”
```

- Wouldn't be dead useful, n'est-ce pas ?
- But is inconsistent:
  Consider the instance: $Y(\neg) = \neg(Y(\neg))$

How to built theories in a logically safe manner ?

- This leads to are a number of questions:
  - Is the logic HOL consistent ?
  - Is HOL correctly implemented in Isabelle ?
  - How to extend HOL in a logically safe way ?
  - Is there a **method** that scales to the entire HOL library, i.e. to „Main" ?

We will address these questions one by one ...

How to built theories in a logically safe manner ?

- HOL consistency
  - … can only be answered relatively,
    i.e. relative to a logical system which gives
    a formal „interpretation" of HOL terms.
  - the gold-standard for mathematicians and
    logicians is „Zermelo-Fraenkel Set Theory"
    plus „axiom of choice", called ZFC.
  - it is possible to give several interpretations of HOL
    in ZFC and prove the validity of HOL's core axioms
    relative to these interpretations.

How to built theories in a logically safe manner ?

- HOL consistency
  - ZFC gives a kind of „universe of sets“ V with the properties:
    - an infinite set I is part of V
    - any product V‘× V“ is part of V, if V‘ and V“ are
    - any potence set $\mathscr{P}$(V‘) is part of V provided that V‘ is. (this is not possible in a typed set-theory)

  - Since relations $\mathscr{P}$(V‘× V“) are part of V, it is possible to express in V function spaces.
  - ZFC gives us an “untyped set-theory”

How to built theories in a logically safe manner ?

- HOL consistency
  - Since relations $\mathscr{P}$(V'× V") are part of V, it is possible to define in V the following function spaces:

    - $A \Rightarrow_{standard} B = \{f: \mathscr{P}(V' \times V'') \mid f \neq \emptyset$ and f is function$\}$
    - $\emptyset \subset A \Rightarrow_{henkin} B \subseteq \{f: \mathscr{P}(V' \times V'') \mid f \neq \emptyset$ and f is function$\}$
    - $A \Rightarrow_{construct} B = \{f: \mathscr{P}(V' \times V'') \mid f \neq \emptyset$ and f is a computable function$\}$

How to built theories in a logically safe manner ?

- HOL consistency
  - On this basis, we can give a standard / Henkin-style / constructivist interpretation of HOL types τ into V:
    - $I_{standard}$, the “standard model”
    - I_{henkin} , the Henkin-model
    - $I_{construct}$, the constructivist model

How to built theories in a logically safe manner ?

- HOL consistency
  - On this basis, we can give a standard interpretation of HOL core types into V
    - $I_{standard}$ ⟦bool⟧ = {a,b}   (where a,b are some distinct elements from the infinite set I)
    - $I_{standard}$ ⟦ind⟧ = I
    - $I_{standard}$ ⟦τ ⇒ τ'⟧ = $I_{standard}$⟦τ⟧ $\Rightarrow_{standard}$ $I_{standard}$⟦τ'⟧

How to built theories in a logically safe manner ?

- HOL consistency
  - On this basis, we can give a Henkin interpretation of HOL core types into V
    - I_{henkin} ⟦bool⟧ = {a,b} (where a,b are some distinct elements from the infinite set I)
    - I_{henkin} ⟦ind⟧ = I
    - I_{henkin} ⟦$\tau \Rightarrow \tau'$⟧ = (I_{henkin}⟦τ⟧) $\Rightarrow_{henkin}$ (I_{henkin}⟦τ'⟧)

How to built theories in a logically safe manner ?

- HOL consistency
  - On this basis, we can give a standard interpretation of HOL core types into V
    - $I_{construct}$ ⟦bool⟧ = {a,b} (where a,b are some distinct elements from the infinite set I)
    - $I_{construct}$ ⟦ind⟧ = I
    - $I_{construct}$ ⟦τ ⇒ τ'⟧ = $I_{construct}$⟦τ ⟧ $\Rightarrow_{construct}$ $I_{construct}$⟦τ'⟧
  - It is easy to show that our typing rules are consistent with $I_{standard}$, I_{henkin} , $I_{construct}$.

How to built theories in a logically safe manner ?

- HOL consistency
  - Core HOL needs a small number of axioms.
  - Traditional papers [Andrews86] reduce it to 6 axioms plus the axiom of infinity:

$$\frac{}{\exists\, f{::}ind \Rightarrow ind.\ injective\ f \wedge \neg surjective\ f}$$

  - The presentation of the axiomatic core in Isabelle/HOL looks as follows:

How to built theories in a logically safe manner ?

- **The presentation in Isabelle/HOL looks as follows:**
  - refl: "t = (t::'a)"
  - subst: "s = t $\Longrightarrow$ P s $\Longrightarrow$ P t"
  - ext: "($\bigwedge$x::'a. (f x ::'b) = g x) $\Longrightarrow$ (λx. f x) = (λx. g x)"
  - the_eq_trivial: "(THE x. x = a) = (a::'a)"
  - impI:"(P $\Longrightarrow$ Q) $\Longrightarrow$ P$\longrightarrow$ Q"
  - mp: "P $\longrightarrow$ Q $\Longrightarrow$ P $\Longrightarrow$ Q"
  - iff: "(P $\longrightarrow$ Q) $\longrightarrow$ (Q $\longrightarrow$ P) $\longrightarrow$ (P = Q)
  - True_or_False: "(P = True) $\vee$ (P = False)"

How to built theories in a logically safe manner ?

- where:
  - True is an abbreviation for $((\lambda x::bool.\ x) = (\lambda x.\ x))$
  - All(P) for $(P = (\lambda x.\ True))$
  - False for $(\forall P.\ P)$
  - Not P for $P \longrightarrow False$
  - and for $\forall R.\ (P \longrightarrow Q \longrightarrow R) \longrightarrow R$
  - or for $\forall R.\ (P \longrightarrow R) \longrightarrow (Q \longrightarrow R) \longrightarrow R$

How to built theories in a logically safe manner ?

- It is straight-forward to prove for the semantic interpretations $I_{standard}$, I_{henkin} , $I_{construct}$ for HOL types, terms and formulas in ZFC

- (Meta) Theorem: Consistency relative to ZFC

  $I_{standard} : \tau => V$ and $I_{standard} : T => V$ build a model for Core-HOL, i.e. they satisfy all core axioms for all assignments of the free variables they contain.

- (Meta) Theorem: Incompleteness

  This model is incomplete for Core-HOL, i.e. there are always true terms for which this fact can not be derived.

How to built theories in a logically safe manner ?

- It is straight-forward to prove for the semantic interpretations $I_{standard}$, I_{henkin} , $I_{construct}$ for HOL types, terms and formulas in ZFC

- (Meta) Theorem: Consistency relative to ZFC

  $I_{Henkin} : \tau \Rightarrow V$ and $I_{Henkin} : T \Rightarrow V$ build a model for Core-HOL, i.e. they satisfy all core axioms for all assignments of the free variables they contain.

- (Meta) Theorem: Incompleteness

  This model is complete for Core-HOL, i.e. there are always true terms for which this fact can not be derived.

How to built theories in a logically safe manner ?

- It is straight-forward to prove for the semantic interpretations $I_{standard}$, I_{henkin} , $I_{construct}$ for HOL types, terms and formulas in ZFC

- (Meta) Theorem: Consistency relative to ZFC

  $I_{Construct} : \tau \Rightarrow V$ and $I_{Construct} : T \Rightarrow V$ build a model for Core-HOL, i.e. they satisfy all core axioms for all assignments of the free variables they contain.

- (Meta) Theorem: Incompleteness

  This model is incomplete for Core-HOL, but there exists an Isomorphism between proofs and (inhabited) types (HoCuSo).

How to built theories in a logically safe manner ?

- Is Isabelle/HOL a correct implementation of HOL?
  - Isabelle as a system clearly contains bugs; but that does not mean that logical inferences produce false results
  - Isabelle has a kernel architecture
    it is a member of the LCF-style systems that protects „theorems", i.e. triples of the form:

    $$\Gamma \vdash_{\Theta} \phi$$

    by a fairly small abstract data-type.
  - Isabelle can generate proof-objects which can be checked outside Isabelle, in principle by any other HOL prover.
  - It is heavily tested and used for a long time.

Conservative Theory Extensions in Isabelle/HOL

How to built theories in a logically safe manner ?

- Are Extensions of HOL, so for example, the library „Main", logically safe ?
  - not necessarily, adding arbitrary axioms command ruins consistency easily.
  - some proof-methods are not based on the kernel (sorry, self-built oracles, the code-generator)
  - However, Isabelle encourages to use conservative specification constructs which are in some cases even formally shown to be logically safe.

Isabelle Specification Constructs

- Constant Definitions:

```
definition f::"<τ>"
  where <name> : "f x_1 ... x_n = <t>"
```

example: definition C::"bool ⇒ bool" where "C x = x"

- Type Definitions:

```
typedef ('α_1..'α_n) κ =
  "<set-expr>" <proof>
```

example: typedef even = "{x::int. x mod 2 = 0}"

Specification Commands

- Simple Definitions (Non-Rec. core variant):

  (Σ, A) "$\in$" Θ

  ↓

  ```
  definition f::"<τ>"
    where <name> : "f x1 ... xn = expr"
  ```

  $(\Sigma \oplus f{::}\tau\ ,\ A \oplus \text{"}f\ x_1\ \ldots\ x_n = expr\text{"})$ "$\in$" Θ'

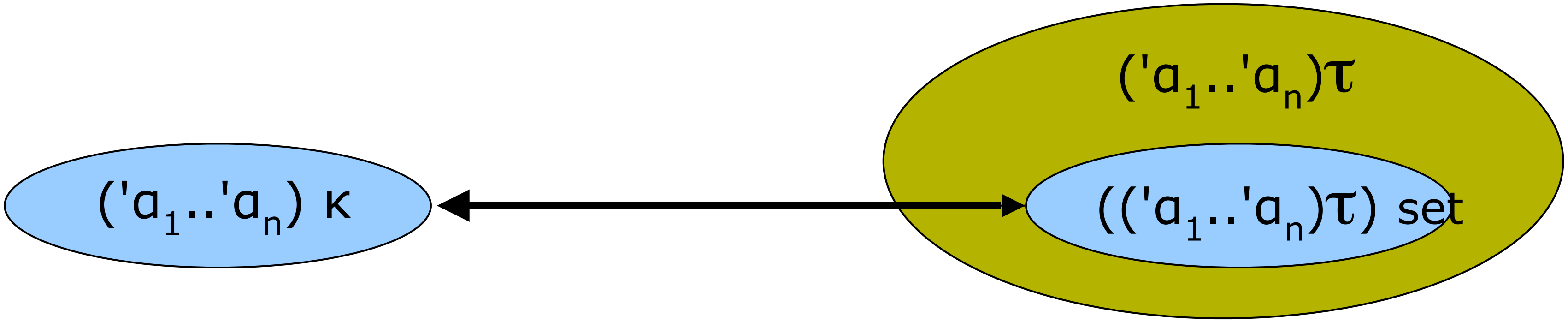
  – Side-Conditions
    - constant symbol f must be fresh
    - f must not be contained in "expr"
    - (all type-variables occurring in expr must occur in τ)

Semantics of a „Type Definition“

- Idea: Similar to constant definitions; we define the new entity (“a type”) by an old one.
- For Type Definitions, we define the new type to be isomorphic to a (non-empty) subset of an old one.
- The Isomorphism is stated by three (conservative) axioms.

Semantics of a „Type Definition"

- Idea: Similar to constant definitions; we define the new entity ("a type") by an old one.



Isabelle Specification Constructs

- Type definition:

(Σ, A) "$\in$" Θ

typedef $('a_1..'a_n)\ \kappa$ =
"<expr :: $(('a_1..'a_n)\tau)$ set>" <proof>

$(\Sigma \oplus ('a_1..'a_n)\ \kappa \oplus Abs\_\kappa::('a_1..'a_n)\tau \Rightarrow ('a_1..'a_n)\kappa$
$\oplus\ Rep\_\kappa::('a_1..'a_n)\kappa \Rightarrow ('a_1..'a_n)\tau$

$A \oplus \{Rep\_\kappa\_inverse \mapsto Abs\_\kappa\ (Rep\_\kappa\ x) = x\}$

$\oplus \{Rep\_\kappa\_inject \mapsto (Rep\_\kappa\ x = Rep\_\kappa\ y) = (x = y)\}$

$\oplus \{Rep\_\kappa \mapsto Rep\_\kappa\ x \in \{x.\ expr\ x\})$ "$\in$" Θ'

- where the type-constructor κ is "fresh" in Θ and expr is closed
- <expr:: $('a_1..'a_n)\tau$ set> is non-empty (to be proven by a witness)

Semantics of a „Type Definition"

- Major example: Typed sets can be built following this scheme. The trick is to identify α set with characteristic functions α ⇒ bool.

- In Isabelle/HOL, α set is introduced via an equivalent axiom scheme; the type-definition uses already implicitly the α set isomorphism to α ⇒ bool.

Isabelle Specification Constructs

- Major example:

  The construction of the cartesian product:

```
definition Pair_Rep :: "'a ⇒ 'b ⇒ 'a ⇒ 'b ⇒ bool"
   where     "Pair_Rep a b = (λx y. x = a ∧ y = b)"

definition "prod = {f. ∃ a b. f = Pair_Rep (a :: 'a) (b :: 'b)}"

typedef ('a, 'b) prod (infixr "*" 20) = "prod :: ('a ⇒ 'b ⇒ bool) set" <proof>

type_notation (xsymbols)  "prod"  ("(_ ×/ _)" [21, 20] 20)
```

Specification Mechanism Commands

- Extended Notation for Cartesian Products: records (as in SML or OCaml; gives a slightly OO-flavor)

```
record    <c> = [<record> + ]
              tag₁ :: "<τ₁>"
                 ...
              tagₙ :: "<τₙ>"
```

- … introduces also semantics and syntax for
  - selectors : $tag_1\ x$
  - constructors : $⦇\ tag_1 = x_1, \ldots, tag_n = x_n\ ⦈$
  - update-functions : $x\ ⦇\ tag_1 := x_n\ ⦈$

Specification Mechanism Commands

- Inductively Defined Sets:

```
inductive_set     <c> :: "  τ ⇒ τ' set" for A::τ
  where  <thmname> : "<φ>"
               | ...
           | <thmname> = <φ>
```

```
example:    inductive_set     Even :: "int set"
              where   null: "0 ∈ Even"
                       | plus:"x ∈ Even ⟹ x+2 ∈ Even"
                       | min :"x ∈ Even ⟹ x-2 ∈ Even"
```

Specification Mechanism Commands

- These are not built-in constructs, rather they are based on a series of definitions and typedefs.

  The machinery behind is based on a fixed-point combinator on sets :

  $$\text{lfp} :: \text{“}('\alpha\ \text{set} \Rightarrow '\alpha\ \text{set}) \Rightarrow '\alpha\ \text{set”}$$

  which can be conservatively defined by

  $$\text{"lfp } f = \bigcap \{u.\ f\ u \subseteq u\}\text{”}$$

  and which enjoys a constrained fixed-point property:

  $$\text{mono } f \Longrightarrow \text{lfp } f = f\ (\text{lfp } f)$$

Specification Mechanism Commands

- Example : Even (see before)
  - the set Even is conservatively defined by:

    Even = lfp (λ X. {0} ∪ (λ x. x + 2) ` X ∪ (λ x. x - 2) ` X)

  - from which the properties:

    null: "0 ∈ Even"
    plus:"x ∈ Even ⟹ x+2 ∈ Even"
    min :"x ∈ Even ⟹ x-2 ∈ Even"

  are derived automatically behind the scenes

Specification Mechanism Commands

- Variante: Inductively Defined Predicates:

```
inductive    <c> [ for <v>:: “<τ>” ]
      where  <thmname> : “<φ>”
                       | ...
             | <thmname> = <φ>
```

```
example: inductive path for rel ::"'a ⇒ 'a ⇒ bool"
           where  base : “path rel x x”
              |   step : “rel x y ⟹ path rel y z ⟹ path rel x z”
```

Specification Mechanism Commands

- Datatype Definitions (similar SML/OCaml/Haskell):
  (Machinery behind : complex series of const and typedefs !)

```
datatype ('a_1..'a_n) T =
 <c> :: "<τ>"  | ... |  <c> :: "<τ>"
```

- Recursive Function Definitions:
  (Machinery behind: Veeery complex series of const and typedefs and automated proofs!)

```
fun <c> ::"<τ>" where
    "<c> <pattern> = <t>"
  | ...
  |  "<c> <pattern> = <t>"
```

Specification Mechanism Commands

- Datatype Definitions (similar SML):
  Examples:

datatype mynat = ZERO | SUC mynat

datatype 'a list = MT | CONS "'a" "'a list"

Some more Automation in Isabelle/HOL

More on Proof-Methods

- Some advanced automated proof-methods use theorem data-bases stored in the global context of a theory
- This holds for:
  - equational reasoning (rewriting : simp, metis)
  - classical reasoning (fast, blast)
  - combined methods (auto, cases, induct)
- Specification Constructs generate theorems and sets up these "background theories" automatically

More on Proof-Methods

- Some composed methods
  (internally based on assumption, erule_tac and
  rule_tac + tactic code that constructs the
  substitutions)

  - simp

    (arbitrary number of left-to-right rewrites,
    assumption or rule refl attepted at the end;
    a global simpset in the background is used.)

  - simp add: <equation> … <equation>
  - simp only: <equation> … <equation>

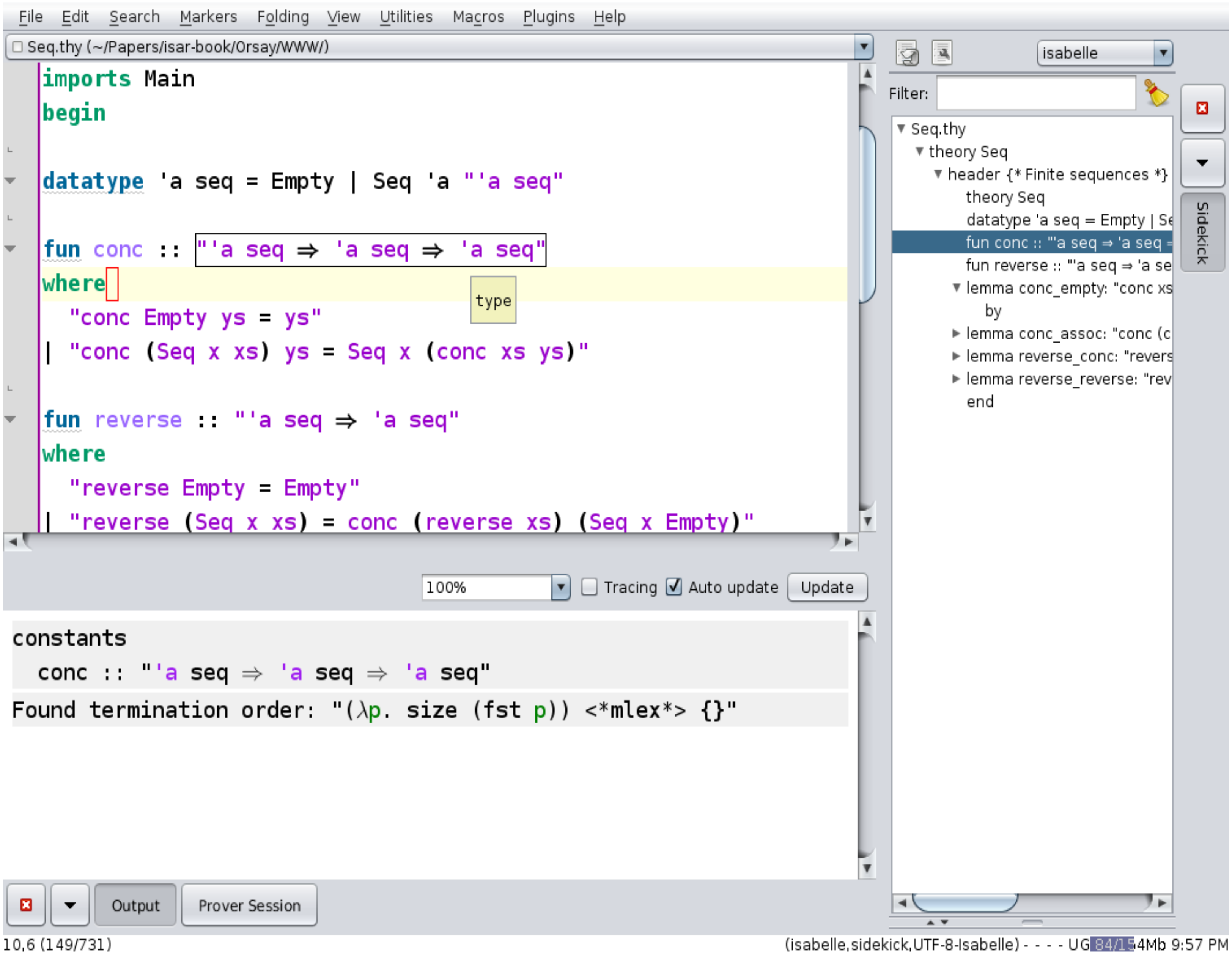
More on Proof-Methods

- Some composed methods
  (internally based on assumption, erule_tac and
  rule_tac + tactic code that constructs the substitutions)
  - auto
    (apply in exhaustive, non-deterministic manner:
    all introduction rules, elimination rules and
  - auto intro: <rule> … <rule>
    elim: <erule> … <erule>
    simp: <equation> … <equation>

More on Proof-Methods

- Some composed methods
 (internally based on assumption, erule_tac and
 rule_tac + tactic code that constructs the
 substitutions)
  - cases „<formula>"
   (split top goal into 2 cases:
   <formula> is true or <formula> is false)
  - cases „<variable>"
   (- precondition : <variable> has type t which is a data-type)
   search for splitting rule and do case-split over this variable.
  - induct_tac „<variable>"
   (- precondition : <variable> has type t which is a data-type)
   search for induction rule and do induction over this variable.

Screenshot with Examples



Conclusion

- HOL has several Models in ZFC, incomplete, complete, and constructivist ones
- Models justify the notion of "conservative theory extensions" (definition, type-definition, ...)
- Isabelle supports a number of "specification constructs" built from conservative theory extensions
- Isabelle/HOL's library is built uniquely from them which guarantees logical consistency by construction
- Isabelle/HOL possesses a kernel-architecture in the tradition of so-called "LCF-style provers"