Isabelle Tutorial:

System, HOL and Proofs

Burkhart Wolff

Université Paris-Sud

What we will talk about

What we will talk about Isabelle with:

- its System Framework
- the Logical Framework
- the Isabelle/HOL Environment
- Proof Contexts and Structured Proof
- Tactic Proofs ("apply style")

The Isabelle
Logical Framework (I)

Overview

- A Universal Notion of Terms & Types:
  Curry-Style Typed λ Calculus with Type-Classes
- A Universal Notion of Rule: Isabelle/Pure
- A Gentle Introduction to HOL
- Forward Proofs
- Backward Proofs
- ML-Level Proofs
- System Architecture
- Conclusion

Isabelle Kernel: Types and Terms

- A Typed Lambda-Calculus without frills.
- Types:
  - type classes Ξ (* eg. order, lattice *)
  - type constructors κ (* eg. bool, list,_x_*)
  - type variables [and actually schematic type variables $?\alpha$]
  - types τ ::= prop | $\tau \Rightarrow \tau$ | $(\tau,\ldots,\tau)\kappa$ | $\alpha::\{\Xi, \ldots, \Xi\}$
- Terms:
  - variable symbols: $V = \{x_1, x_2, \ldots\}$ [and actually ?V's too]
  - constant symbols: $C = \{c_1, c_2, \ldots\}$
  - term ::= $V::\tau$ | $C::\tau$ | term term | $\lambda\, V::\tau$. term
  - Isabelle offers powerful pretty-printing: (_ + _) t t' == t + t' !!

Isabelle Kernel: Typed Terms

- Well-typed terms (cterm's):
  the usual type inference system.

- Congruences on cterm's:
  - equality on cterm is $\alpha\beta\eta$ congruence
  - α : $\lambda x.\ t \equiv \lambda y.\ t[x \mapsto y]$
  - β : $(\lambda x.\ t)\ t' \equiv t[x \mapsto t']$
  - η : $(\lambda x.\ t) \equiv t$ (provided x not occuring in t)
  - equality for well-typed terms decidable.

Isabelle Kernel: Global Contexts

- Global Contexts Θ, i.e. Theories,
  i.e. inductively defined sets of pairs pair of:
  - Signature Σ (types, constants,syntax)
    where $\Sigma \equiv C \mapsto \tau$
    (a partial map from constant symbols to types τ)
  - Axioms A (a partial map of names to "thm"s))
    where thm's are triples:

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

- with a set Γ *of* assumptions, i.e. cterm's of type prop
- with the conclusion ϕ, i.e. a ctem of type prop
- with Θ the context in which this thm is valid.

Isabelle Kernel: Commands as global context transactions

- Theory Extensions are:
  - Signature Σ (types, constants,syntax)
  - Axioms A (set of formulas)

(Σ, A) "$\in$" Θ

command denoting global context transition

(Σ', A') "$\in$" Θ'

Isabelle Kernel: Commands as global context transactions

- Theory Extensions are:
  - Signature Σ (types, constants,syntax)
  - Axioms A (set of formulas)

(Σ, A) "$\in$" Θ

$\Downarrow$

$(\Sigma + \{c \mapsto \tau\}, A)$ "$\in$" Θ'

consts <c> :: "<τ>"

Isabelle Kernel: Commands as global context transactions

- Theory Extensions are:
  - Signature Σ (types, constants,syntax)
  - Axioms A (set of formulas)

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

axiomatization <c>
where <name>:"<ϕ>"

$$(\Sigma, A + \{\text{name} \mapsto \phi + \ldots\}) \text{ "}{\in}\text{" } \Theta'$$

Isabelle Kernel: Commands as global context transactions

- Theory Extensions (roughly speaking) are:
  - Signature Σ (types, constants,syntax)
  - Axioms A (set of formulas)

(Σ, A) "$\in$" Θ

$\Downarrow$

$(\Sigma, A + \{name \mapsto \phi + \ldots\})$ "$\in$" Θ'

axiomatization <c>
where <name>:"<ϕ>"

Don't use this ever directly !!!

Isabelle Kernel: The initial global context „Pure“

- Pure is a logical meta-language, i.e. the built-in language in which logical rules as such can be represented.
- It consists of typed λ-terms with constants:
  - foundational types “prop” and “_ => _” (“_ ⇒_”)
  - the Pure (universal) quantifier

    all :: “(α ⇒ Prop) ⇒ Prop”

    (“⋀x. P x”,“\<And> x. P x”   “!!x. P x”)
  - the Pure implication “A ==> B” (“_ ⟹ _”)
  - the Pure equality   “A == B”   “A ≡ B”

Isabelle Kernel: The initial global context „Pure“

- Pure is the meta-language, i.e. the built-in formula language (“inner syntax”).
- Equivalent notations for natural deduction rules:

$A_1 \Longrightarrow (\dots \Longrightarrow (A_n \Longrightarrow A_{n+1})\dots),$

$[\![A_1; \dots; A_n]\!] \Longrightarrow A_{n+1},$

$$\frac{A_1 \quad \dots \quad A_n}{A_{n+1}}$$

theorem
assumes A_1
and …
and A_n
shows A_{n+1}

Isabelle Kernel: The initial global context „Pure“

- Pure is the meta-language, i.e. the built-in formula language (“inner syntax”).
- Equivalent notations for natural deduction rules:

$(P \Longrightarrow Q) \Longrightarrow R$:

$$\dfrac{\begin{matrix}[P] \\ \vdots \\ Q\end{matrix}}{R}$$

theorem
assumes "P ⟹ Q"
shows "R"

Isabelle Kernel: The initial global context „Pure“

- Pure provides a built-in formula-language, a is the meta-language.
- Equivalent notations for natural deduction rules:

$(\bigwedge a.\ P\ a \Longrightarrow Q\ a) \Longrightarrow R$:

$$\dfrac{\begin{array}{c}[P]_a \\ \vdots \\ Q\end{array}}{R}$$

```
theorem
  fixes a
   assumes "P a ⟹ Q a"
  shows   R
```

Isabelle Specification Constructs

- Methodology to use only logically safe („conservative") Theory Extensions.
  These are:
  - constant definition
  - type definition
  - constant specification
  - type specification

Advanced Isabelle Specification Constructs

- Methodology to use only logically safe („conservative") Theory Extensions. These are:
  - datatype definition
  - inductive definition
  - primrec , fun definitions
  - type specification

Isabelle Specification Constructs

- Constant definition:

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

```
definition <c> :: “<τ>”
where  <name>:"<c ≡ φ>"
```

$$(\Sigma + \{c :: \tau\}, A + \{name \mapsto c \equiv \phi\}) \text{ ”}\in\text{” } \Theta'$$

  - where c is “fresh” in Θ
  - ϕ is closed
  - ϕ is type variable closed

Some Commands for Inspection

- Some Isabelle document commands serve to inspect the document content.
  - checking a type expression:

    prop "<τ>"

    example: typ "prop ⇒ prop"

  - checking a term expression:

    term "<t>"

    example: term "λx. x"

Some Statements (for Inspection)

- We can state (not yet prove) lemmas and theorems:
  - a lemma:

```
lemma <name>: "<ϕ>"
  <proof>
```

example: lemma nix: "A ⇒ A" sorry

  - a theorem:

```
theorem <name>:
  fixes V ...
  assumes "<ϕ>"
  shows "<ϕ>" <proof>
```

example: term "λx. x"

Exercise demo2.thy

- Build a theory in “Main”
  (which is actually the brand-name for
  “Higher-order Logic“ (HOL) to be discussed next)
- Check some types
- Check some propositions
- State lemmas (proof by „sorry“ or „oops“)
- State a theorem in structured syntax
- State an Axiom and a Definition