COMP4011/8011 Advanced Topics in Formal Methods and Programming Languages

– Software Verification with Isabelle/HOL –

Peter Höfner

September 22, 2024

Section 15

Isar (Part 2)

Datatypes in Isar

General Case Distinctions

show *formula*
proof -
have $P_1 \vee P_2 \vee P_3$ <proof>
moreover { **assume** P_1 ... **have** ?thesis <proof> }
moreover { **assume** P_2 ... **have** ?thesis <proof> }
moreover { **assume** P_3 ... **have** ?thesis <proof> }
ultimately show ?thesis **by** blast
qed

{ . . . } is a proof block similar to **proof ... qed**

{ **assume** P_1 . . . **have** P <proof> }
stands for $P_1 \Longrightarrow P$

Datatype case distinction

proof (cases *term*)
   case Constructor_1
   ⋮
next
⋮
next
   case (Constructor_k $\vec{x}$)
   $\cdots$ $\vec{x}$ $\cdots$
qed

case (Constructor_i $\vec{x}$) $\equiv$
fix $\vec{x}$ **assume** Constructor_i : "$\textit{term} = \text{Constructor}_i\ \vec{x}$"

Structural induction for nat

show $P\ n$
proof (induct n)
  case 0 $\equiv$ **let** $?case = P\ 0$
  ...
  show $?case$
next
  case (Suc n) $\equiv$ **fix** n **assume** Suc: $P\ n$
  let $?case = P\ (\text{Suc}\ n)$
  ...
  $\cdots\ n\ \cdots$
  show $?case$
qed

Structural induction: $\Longrightarrow$ and $\bigwedge$

show "$\bigwedge x.\ A\ n \Longrightarrow P\ n$"
proof (induct n)
  case 0 $\equiv$ **fix** x **assume** 0: "$A\ 0$"
    let $?case$ = "$P\ 0$"
  ...
  show $?case$
next
  case (Suc n) $\equiv$ **fix** n and x
    assume Suc: "$\bigwedge x.\ A\ n \Longrightarrow P\ n$"
      "$A\ (\mathrm{Suc}\ n)$"
    let $?case$ = "$P\ (\mathrm{Suc}\ n)$"
  ...
  $\cdots\ n\ \cdots$
  ...
  show $?case$
qed

Demo: Datatypes in Isar

Calculational Reasoning

The Goal

Prove:
$x \cdot x^{-1} = 1$

using:

assoc:	$(x \cdot y) \cdot z = x \cdot (y \cdot z)$
left_inv:	$x^{-1} \cdot x = 1$
left_one:	$1 \cdot x = x$

The Goal

Prove:

$$
\begin{aligned}
x \cdot x^{-1} &= 1 \cdot (x \cdot x^{-1}) \\
\ldots &= 1 \cdot x \cdot x^{-1} \\
\ldots &= (x^{-1})^{-1} \cdot x^{-1} \cdot x \cdot x^{-1} \\
\ldots &= (x^{-1})^{-1} \cdot (x^{-1} \cdot x) \cdot x^{-1} \\
\ldots &= (x^{-1})^{-1} \cdot 1 \cdot x^{-1} \\
\ldots &= (x^{-1})^{-1} \cdot (1 \cdot x^{-1}) \\
\ldots &= (x^{-1})^{-1} \cdot x^{-1} \\
\ldots &= 1
\end{aligned}
$$

assoc:	$(x \cdot y) \cdot z = x \cdot (y \cdot z)$
left_inv:	$x^{-1} \cdot x = 1$
left_one:	$1 \cdot x = x$

Can we do this in Isabelle?

- Simplifier: too eager
- Manual: difficult in apply style
- Isar: with the methods we know, too verbose

Chains of equations

The Problem

$$\begin{array}{ccc} a & = & b \\ \dots & = & c \\ \dots & = & d \end{array}$$

shows $a = d$ by transitivity of $=$

Each step usually nontrivial (requires own subproof)

Solution in Isar:

- Keywords **also** and **finally** to delimit steps
- **. . .**: predefined schematic term variable, refers to right hand side of last expression
- Automatic use of transitivity rules to connect steps

also/finally

have “$t_0 = t_1$” [proof]
also
have “... $= t_2$” [proof]
also
$\vdots$
also
have “$\cdots = t_n$” [proof]
finally
show P
— 'finally' pipes fact “$t_0 = t_n$” into the proof

calculation register
“$t_0 = t_1$”

“$t_0 = t_2$”
$\vdots$
“$t_0 = t_{n-1}$”

$t_0 = t_n$

More about also

- Works for all combinations of $=$, $\leq$ and $<$.
- Uses all rules declared as `[trans]`.
- To view all combinations: `print_trans_rules`

Designing [trans] Rules

have = "$l_1 \odot r_1$" [proof]
also
have "... $\odot r_2$" [proof]
also

Anatomy of a [trans] rule:

- Usual form: plain transitivity $[\![l_1 \odot r_1; r_1 \odot r_2]\!] \Longrightarrow l_1 \odot r_2$
- More general form: $[\![P\ l_1\ r_1; Q\ r_1\ r_2; A]\!] \Longrightarrow C\ l_1\ r_2$

Examples:

- pure transitivity: $[\![a = b; b = c]\!] \Longrightarrow a = c$
- mixed: $[\![a \leq b; b < c]\!] \Longrightarrow a < c$
- substitution: $[\![P\ a; a = b]\!] \Longrightarrow P\ b$
- antisymmetry: $[\![a < b; b < a]\!] \Longrightarrow False$
- monotonicity:
  $[\![a = f\ b; b < c; \bigwedge x\ y.\ x < y \Longrightarrow f\ x < f\ y]\!] \Longrightarrow a < f\ c$

Demo

Finding Theorems

Command **find_theorems** (C-c C-f) finds combinations of:

- pattern: **"_ + _ + _"**
- lhs of simp rules: **simp: "_ * (_ + _)"**
- intro/elim/dest on current goal
- lemma name: **name: assoc**
- exclusions thereof: **-name: "HOL."**

find_theorems dest -"hd" name: "List."

finds all theorems in the current context that

- match the goal as dest rule,
- do not contain the constant "hd"
- are in the List theory (name starts with "List.")

Isar: define and defines

Can define vnameal constant in Isar proof context:

proof

...
define ”f ≡ big term”
have ”g = f x” ...

like definition, not automatically unfolded (f_def)

different to **let** ?f = ”big term”

Also available in lemma statement:

lemma ...:
fixes ...
assumes ...
defines ...
shows ...