Australian National University

COMP3610/6361
Principles of Programming Languages

Peter Höfner

Aug 22, 2023

Section 13

IMP in Isabelle/HOL

Motivation/Disclaimer

- generic proof assistant
- express mathematical formulas in a formal language
- tools for proving those formulas in a logical calculus
- originally developed at the University of Cambridge and Technische Universität München (now numerous contributions, including Australia)

- this is **neither a course about Isabelle nor a proper introduction to Isabelle**

Isabelle/HOL – Introduction

Isabelle/HOL = Functional Programming + Logic

Isabelle HOL has

- datatypes
- recursive functions
- logical operators
- …

Isabelle/HOL is a programming language, too

- Higher-order means that functions are values, too

Isabelle/HOL – Terms (Expressions)

- **Functions**
  - application: $f\ E$
    call of function f with parameter E
  - abstraction: $\lambda x.\ E$
    function with parameter x (of some type) and result E ((**fn** $x : T_? \Rightarrow t$))
  - Convention (as always) $f\ E_1\ E_2\ E_3 \equiv ((f\ E_1)\ E_2)\ E_3$

- **Basic syntax** (Isabelle)

$t ::=$	(t)	
	$\mid a$	identifier (constant or variable)
	$\mid t\ t$	function application
	$\mid \lambda x.\ t$	function abstraction
	$\mid \ldots$	syntactic sugar

- **Substitution** notation: $t[u/x]$

Australian National University

Isabelle/HOL – Types I

- **Basic syntax** (Isabelle)

$\tau ::=$	(τ)	
	$\mid$ `bool` $\mid$ `int` $\mid$ `string` $\mid \ldots$	base types
	$\mid\ 'a \mid 'b \mid \ldots$	type variables
	$\mid\ \tau \Rightarrow \tau$	functions
	$\mid\ \tau \times \tau$	pairs
	$\mid\ \tau$ `list`	lists
	$\mid\ \tau$ `set`	sets
	$\mid\ \ldots$	user-defined types

  Convention: $\tau_1 \Rightarrow \tau_2 \Rightarrow \tau_3 \;\equiv\; \tau_1 \Rightarrow (\tau_2 \Rightarrow \tau_3)$

- **Terms must be well-typed**; in particular

$$\frac{t :: \tau_1 \Rightarrow \tau_2 \qquad u :: \tau_1}{t\ u :: \tau_2}$$

Isabelle/HOL – Types II

Type inference

- automatic
- function overloading possible
  can prevent type inference
- **type annotation** $t :: \tau$ (for example $f\ (x :: \texttt{int})$)

Currying

- curried vs. tupled

$$f \quad \tau_1 \Rightarrow \tau_2 \Rightarrow \tau_3 \qquad \text{vs} \qquad f \quad \tau_1 \times \tau_2 \Rightarrow \tau_3$$

- use curried versions if possible
- advantage: allow *partial function application*

$$f\ a_1 \qquad \text{where } a_1 :: \tau_1$$

Isabelle (Cheatsheet I)

Isabelle module = Theory (File structure)

Syntax: **theory** $MyTh$
imports $Th_1, \ldots, Th_n$
begin
(definitions, lemmas, theorems, proofs, . . .)*
end

$MyTh$: name of theory. Must live in file $MyTh$`.thy`

Th_i: names of imported theories; imports are transitive

Usually: **imports** `Main`

IMP – Syntax (recap)

Booleans	$b \in \mathbb{B} = \{\texttt{true}, \texttt{false}\}$
Integers (Values)	$n \in \mathbb{Z} = \{\ldots, -1, 0, 1, \ldots\}$
Locations	$l \in \mathbb{L} = \{l, l_0, l_1, l_2, \ldots\}$
Operations	$op ::= + \mid \geq$
Expressions	

$$
\begin{aligned}
E ::= {} & n \mid b \mid E \; op \; E \mid \\
& l := E \mid !l \mid \\
& \textbf{if } E \textbf{ then } E \textbf{ else } E \mid \\
& \textbf{skip} \mid E \,;\, E \mid \\
& \textbf{while} E \textbf{ do } E
\end{aligned}
$$

IMP – Syntax (aexp and bexp)

Booleans $b \in \mathbb{B}$

Integers (Values) $n \in \mathbb{Z}$

Locations $l \in \mathbb{L} = \{l, l_0, l_1, l_2, \dots\}$

Operations $aop ::= +$

Expressions

$$\texttt{aexp} ::= n \mid\ !l \mid \texttt{aexp}\ aop\ \texttt{aexp}$$

$$\texttt{bexp} ::= b \mid \texttt{bexp} \wedge \texttt{bexp} \mid \texttt{aexp} \geq \texttt{aexp}$$

$$\texttt{com} ::= \sout{n} \mid \sout{b} \mid \sout{E\ op\ E} \mid$$

$$l ::= \texttt{aexp} \mid \sout{!l} \mid$$

$$\texttt{IF bexp THEN com ELSE com} \mid$$

$$\texttt{SKIP} \mid \texttt{com ;; com} \mid$$

$$\texttt{WHILE bexp DO com}$$

IMP – Syntax (Isabelle)

Booleans	`bool`
Integers (Values)	`int`
Locations	`string`

Expressions

datatype `aexp` ::= `N n` | `V l` | `Plus aexp aexp`

datatype `bexp` ::= `Bc bool` | `Not bexp` |
`And bexp bexp` | `LESS aexp aexp`

datatype `com` ::= `Assign loc aexp` |
`If bexp com com` |
`SKIP` | `Seq com com` |
`WHile bexp com`

IMP – Syntax (Isabelle)

LINK: /src/HOL/IMP

Isabelle (Cheatsheet II)

type_synonym	specify synonym for a type
datatype	define recursive (polymorphic) types
fun	define (simple, recursive) function (tries to prove exhaustiveness, non-overlappedness, and termination)
value	evaluate a term

Small-step semantics

- a configuration $\langle E\,,\,s\rangle$ can perform a step if there is a derivation tree
- vice versa the set of all transitions can be defined inductively
- it is an infinite set

IMP Semantics

(deref) ~~$\langle !l\,,\, s\rangle \longrightarrow \langle n\,,\, s\rangle$ if $l \in \mathsf{dom}(s)$ and $s(l) = n$~~

(assign1) $\langle l := n\,,\, s\rangle \longrightarrow \langle \textbf{skip}\,,\, s + \{l \mapsto n\}\rangle$ if $l \in \mathsf{dom}(s)$

(assign2) ~~$\dfrac{\langle E\,,\, s\rangle \longrightarrow \langle E'\,,\, s'\rangle}{\langle l := E\,,\, s\rangle \longrightarrow \langle l := E'\,,\, s'\rangle}$~~

(seq1) $\langle \textbf{skip}; E_2\,,\, s\rangle \longrightarrow \langle E_2\,,\, s\rangle$

(seq2) $\dfrac{\langle E_1\,,\, s\rangle \longrightarrow \langle E_1'\,,\, s'\rangle}{\langle E_1; E_2\,,\, s\rangle \longrightarrow \langle E_1; E_2\,,\, s\rangle}$

(if1) $\langle \textbf{if}\ \texttt{true}\ \textbf{then}\ E_2\ \textbf{else}\ E_3\,,\, s\rangle \longrightarrow \langle E_2\,,\, s\rangle$

(if2) $\langle \textbf{if}\ \texttt{false}\ \textbf{then}\ E_2\ \textbf{else}\ E_3\,,\, s\rangle \longrightarrow \langle E_3\,,\, s\rangle$

(if3) ~~$\dfrac{\langle E_1\,,\, s\rangle \longrightarrow \langle E_1'\,,\, s'\rangle}{\langle \textbf{if}\ E_1\ \textbf{then}\ E_2\ \textbf{else}\ E_3\,,\, s\rangle \longrightarrow \langle \textbf{if}\ E_1'\ \textbf{then}\ E_2\ \textbf{else}\ E_3\,,\, s'\rangle}$~~

(while) $\langle \textbf{while} E_1\ \textbf{do}\ E_2\,,\, s\rangle \longrightarrow \langle \textbf{if}\ E_1\ \textbf{then}\ (E_2; \textbf{while} E_1\ \textbf{do}\ E_2)\ \textbf{then}\ \textbf{skip}\,,\, s\rangle$

IMP Semantics

(assign1)	$\langle l := n\ ,\ s\rangle \longrightarrow \langle \textbf{skip}\ ,\ s + \{l \mapsto n\}\rangle \qquad \text{if } l \in \text{dom}(s)$
(seq1)	$\langle \textbf{skip}; E_2\ ,\ s\rangle \longrightarrow \langle E_2\ ,\ s\rangle$
(seq2)	$\dfrac{\langle E_1\ ,\ s\rangle \longrightarrow \langle E_1'\ ,\ s'\rangle}{\langle E_1; E_2\ ,\ s\rangle \longrightarrow \langle E_1; E_2\ ,\ s\rangle}$
(if1)	$\langle \textbf{if}\ \texttt{true}\ \textbf{then}\ E_2\ \textbf{else}\ E_3\ ,\ s\rangle \longrightarrow \langle E_2\ ,\ s\rangle$
(if2)	$\langle \textbf{if}\ \texttt{false}\ \textbf{then}\ E_2\ \textbf{else}\ E_3\ ,\ s\rangle \longrightarrow \langle E_3\ ,\ s\rangle$
(while)	$\langle \textbf{while} E_1\ \textbf{do}\ E_2\ ,\ s\rangle \longrightarrow \langle \textbf{if}\ E_1\ \textbf{then}\ (E_2; \textbf{while} E_1\ \textbf{do}\ E_2)\ \textbf{then skip}\ ,\ s\rangle$

IMP Semantics (Isabelle)

LINK: /src/HOL/Small_Step

IMP – Examples

- If $E = (l_2 := 0; \textbf{while}\ !l_1 \geq 1\ \textbf{do}\ (l_2 :=\ !l_2 +\ !l_1 ; l_1 :=\ !l_1 + -1))$
  $s = \{l_1 \mapsto 3, l_2 \mapsto 0\}$
  then $\langle E\ ,\ s \rangle \longrightarrow^* \ ?$
- determinacy
- progress

Isabelle (Cheatsheet III)

inductive	defines (smallest) inductive set
print_theorems	shows generated theorems
find_theorems	searches available theorems by name and/or pattern
apply `(<rule/tactic>)`	applies rule to proof goal (simp, auto, blast, rule <name>)

Big-step semantics
(in Isabelle/HOL)

Another View: Big-step Semantics

- we have seen a **small-step semantics**

$$\langle E\,,\,s\rangle \longrightarrow \langle E'\,,\,s'\rangle$$

- alternatively, we could have looked at a **big-step semantics**

$$\langle E\,,\,s\rangle \Downarrow \langle E'\,,\,s'\rangle$$

For example

$$\frac{}{\langle n\,,\,s\rangle \Downarrow \langle n\,,\,s\rangle} \qquad \frac{\langle E_1\,,\,s\rangle \Downarrow \langle n_1\,,\,s'\rangle \quad \langle E_2\,,\,s'\rangle \Downarrow \langle n_2\,,\,s''\rangle}{\langle E_1 + E_2\,,\,s\rangle \Downarrow \langle n\,,\,s''\rangle}\ (n = n_1 + n_2)$$

- no major difference for sequential programs
- small-step much better for modelling concurrency

Final State

- Isabelle's version of IMP has only one value: `SKIP`
- big-step semantics can be seen as relation

$$\langle E\,,\,s\rangle \Longrightarrow s'$$

Semantics

(Skip)
$$\langle \texttt{SKIP}\, ,\, s\rangle \Longrightarrow s$$

(Assign)
$$\langle l := a\, ,\, s\rangle \Longrightarrow s + \{l \mapsto \textsf{aval}\ a\ s\}$$

(Seq)
$$\frac{\langle E_1\, ,\, s\rangle \Longrightarrow s' \qquad \langle E_2\, ,\, s'\rangle \Longrightarrow s''}{\langle E_1\, ;\, E_2\, ,\, s\rangle \Longrightarrow s''}$$

(IfT)
$$\frac{\textsf{bval}\ b\ s = \texttt{true} \qquad \langle E_1\, ,\, s\rangle \Longrightarrow s'}{\langle \textbf{if}\ b\ \textbf{then}\ E_1\ \textbf{else}\ E_2\, ,\, s\rangle \Longrightarrow s'}$$

(IfF)
$$\frac{\textsf{bval}\ b\ s = \texttt{false} \qquad \langle E_2\, ,\, s\rangle \Longrightarrow s'}{\langle \textbf{if}\ b\ \textbf{then}\ E_1\ \textbf{else}\ E_2\, ,\, s\rangle \Longrightarrow s'}$$

(WhileF)
$$\frac{\textsf{bval}\ b\ s = \texttt{false}}{\langle \textbf{while}\ b\ \textbf{do}\ E\, ,\, s\rangle \Longrightarrow s}$$

(WhileT)
$$\frac{\textsf{bval}\ b\ s = \texttt{true} \qquad \langle E\, ,\, s\rangle \Longrightarrow s' \qquad \langle \textbf{while}\ b\ \textbf{do}\ E\, ,\, s'\rangle \Longrightarrow s''}{\langle \textbf{while}\ b\ \textbf{do}\ E\, ,\, s\rangle \Longrightarrow s''}$$

IMP Semantics (Isabelle)

LINK: /src/HOL/Big_Step

- inversion rules
- induction set up
- see Nipkow/Klein for more details and explanation

Are big and small-step semantics equivalent?

Isabelle (Cheatsheet IV)

Proof Styles/Proof 'Tactics'

apply-style	apply rules (backwards)
ISAR	human readable proofs
slegdehammer	the 'secret' weapon
	incorporating automated theorem provers

From Big to Small

Theorem

If $cs \Rightarrow s'$ *then* $cs \longrightarrow^* \langle \mathit{SKIP}, s' \rangle$.

Proof by rule induction (on $cs \Rightarrow s'$).

In two cases a lemma is needed.

Lemma

If $\langle E, s \rangle \longrightarrow^* \langle E', s' \rangle$ *then* $\langle E ; E_2, s \rangle \longrightarrow^* \langle E' ; E_2, s' \rangle$.

Proof by rule induction.
(generalisation of (seq2))

From Small to Big

Theorem

If $cs \longrightarrow^* \langle \mathit{SKIP}, s' \rangle$ *then* $cs \Rightarrow s'$*.*

Proof by rule induction (on $cs \longrightarrow^* \langle \texttt{SKIP}, s' \rangle$).

The induction step needs the following (interesting) lemma.

Lemma

If $cs \longrightarrow cs'$ *and* $cs' \Rightarrow s$ *then* $cs \Rightarrow s$*.*

Proof by rule induction on $cs \longrightarrow cs'$.

Equivalence

Corollary

$cs \longrightarrow^* \langle \mathit{SKIP}, s' \rangle$ *if and only if* $cs \Rightarrow s'$.

LINK: /src/HOL/Small_Step

But are they really equivalent?

- What about premature termination?
- What about (non) termination?

Lemma

1. *final* $\langle E , s\rangle$ *if and only if* $E = \mathit{SKIP}$.
2. $\exists s.\ cs \Rightarrow s$ *if and only if* $\exists cs'.\ cs \longrightarrow^* cs' \wedge$ *final* cs'.

where final $cs \equiv (\neg\exists cs'.\ cs \rightarrow cs')$

Proof.

1. induction and rule inversion
2. $$\begin{aligned} (\exists s.\ cs \Rightarrow s) &\Leftrightarrow \exists s.\ cs \longrightarrow^* \langle \mathtt{SKIP} , s\rangle && \text{(by big = small)} \\ &\Leftrightarrow \exists cs'.\ cs \longrightarrow^* cs' \wedge \textit{final}\ cs' && \text{(by final = SKIP)} \end{aligned}$$

□

Typing

(almost straight-forward)

LINK: /src/HOL/Types

```
inductive btyping :: "tyenv ⇒ bexp ⇒ bool"(infix "⊢"50)
where
B_ty:"Γ ⊢ Bc v" |
Not_ty:"Γ ⊢ b ⟹ Γ ⊢ Not b" |
And_ty:"Γ ⊢ b1 ⟹ Γ ⊢ b2 ⟹ Γ ⊢ And b1 b2" |
Less_ty:"Γ ⊢ a1 : τ ⟹ Γ ⊢ a2 : τ ⟹ Γ ⊢ Less a1 a2"

inductive ctyping ::"tyenv ⇒ com ⇒ bool"(infix " ⊢ "50)
where
Skip_ty:"Γ ⊢ SKIP" |
Assign_ty:"Γ ⊢ a : Γ(x) ⟹ Γ ⊢ x ::= a" |
Seq_ty:"Γ ⊢ c1 ⟹ Γ ⊢ c2 ⟹ Γ ⊢ c1 ;; c2" |
If_ty:"Γ ⊢ b ⟹ Γ ⊢ c1 ⟹ Γ ⊢ c2 ⟹ Γ ⊢ IF b THEN c1 ELSE c2" |
While_ty:"Γ ⊢ b ⟹ Γ ⊢ c ⟹ Γ ⊢ WHILE b DO c"
```

References

- some slides based on slides by T. Nipkow (TU München)
- Nipkow, T. and Klein, G. (2014) Concrete Semantics. Springer
- Pierce, B. (2004) Types and Programming Languages. MIT Press
- G. Klein. Gerwin's Style Guide for Isabelle/HOL (Part 1 and 2) `https://proofcraft.org/blog/isabelle-style.html`