COMP 4161
NICTA Advanced Course

Advanced Topics in Software Verification

Gerwin Klein, June Andronick, Toby Murray

λ

Slide 1

Exercises for last time

- ➜ Download and install Isabelle from `http://mirror.cse.unsw.edu.au/pub/isabelle/`
- ➜ Switch on X-Symbol in ProofGeneral
- ➜ Step through the demo files from the lecture web page
- ➜ Write your own theory file, look at some theorems in the library, try 'find theorem'

- ➜ How many theorems can help you if you need to prove something like "Suc(Suc x))"?
- ➜ What is the name of the theorem for associativity of addition of natural numbers in the library?

Slide 2

Content

Rough timeline

- ➜ Intro & motivation, getting started [1]
- ➜ Foundations & Principles
  - Lambda Calculus, natural deduction [2,3,4[a]]
  - Higher Order Logic [5,6[b],7]
  - Term rewriting [8,9,10[c]]
- ➜ Proof & Specification Techniques
  - Isar [11,12[d]]
  - Inductively defined sets, rule induction [13[e],15]
  - Datatypes, recursion, induction [16,17[f],18,19]
  - Calculational reasoning, mathematics style proofs [20]
  - Hoare logic, proofs about programs [21[g],22,23]

[a] a1 out; [b] a1 due; [c] a2 out; [d] a2 due; [e] session break; [f] a3 out; [g] a3 due

Slide 3

λ-calculus

Alonzo Church

- ➜ lived 1903–1995
- ➜ supervised people like Alan Turing, Stephen Kleene
- ➜ famous for Church-Turing thesis, lambda calculus, first undecidability results
- ➜ invented λ calculus in 1930's

λ-calculus

- ➜ originally meant as foundation of mathematics
- ➜ important applications in theoretical computer science
- ➜ foundation of computability and functional programming

Slide 4

untyped λ-calculus

- ➜ turing complete model of computation
- ➜ a simple way of writing down functions

Basic intuition:

instead of $f(x) = x + 5$
write $f = \lambda x.\ x + 5$

$\lambda x.\ x + 5$

- ➜ a term
- ➜ a nameless function
- ➜ that adds 5 to its parameter

Slide 5

Function Application

For applying arguments to functions

instead of $f(a)$
write $f\ a$

Example: $(\lambda x.\ x + 5)\ a$

Evaluating: in $(\lambda x.\ t)\ a$ replace x by a in t
(computation!)

Example: $(\lambda x.\ x + 5)\ (a + b)$ evaluates to $(a + b) + 5$

Slide 6

THAT'S IT!

Slide 7

NOW FORMAL

Slide 8

Syntax

Terms: $t \;::=\; v \mid c \mid (t\;t) \mid (\lambda x.\; t)$

$v, x \in V, \quad c \in C, \quad V, C$ sets of names

- v, x variables
- c constants
- $(t\;t)$ application
- $(\lambda x.\; t)$ abstraction

Slide 9

Conventions

- leave out parentheses where possible
- list variables instead of multiple λ

Example: instead of $(\lambda y.\; (\lambda x.\; (x\; y)))$ write $\lambda y\; x.\; x\; y$

Rules:

- list variables: $\lambda x.\; (\lambda y.\; t) \;=\; \lambda x\; y.\; t$
- application binds to the left: $x\; y\; z \;=\; (x\; y)\; z \neq x\; (y\; z)$
- abstraction binds to the right: $\lambda x.\; x\; y \;=\; \lambda x.\; (x\; y) \neq (\lambda x.\; x)\; y$
- leave out outermost parentheses

Slide 10

Getting used to the Syntax

Example:

$\lambda x\; y\; z.\; x\; z\; (y\; z) =$

$\lambda x\; y\; z.\; (x\; z)\; (y\; z) =$

$\lambda x\; y\; z.\; ((x\; z)\; (y\; z)) =$

$\lambda x.\; \lambda y.\; \lambda z.\; ((x\; z)\; (y\; z)) =$

$(\lambda x.\; (\lambda y.\; (\lambda z.\; ((x\; z)\; (y\; z)))))$

Slide 11

Computation

Intuition: replace parameter by argument
this is called β-reduction

Example

$(\lambda x\; y.\; f\; (y\; x))\;\; 5\;\; (\lambda x.\; x) \longrightarrow_\beta$

$(\lambda y.\; f\; (y\; 5))\;\; (\lambda x.\; x) \longrightarrow_\beta$

$f\; ((\lambda x.\; x)\; 5) \longrightarrow_\beta$

$f\; 5$

Slide 12

Defining Computation

β reduction:

$$\begin{array}{lllllll} & & & (\lambda x.\ s)\ t & \longrightarrow_\beta & s[x \leftarrow t] \\ s & \longrightarrow_\beta & s' & \Longrightarrow & (s\ t) & \longrightarrow_\beta & (s'\ t) \\ t & \longrightarrow_\beta & t' & \Longrightarrow & (s\ t) & \longrightarrow_\beta & (s\ t') \\ s & \longrightarrow_\beta & s' & \Longrightarrow & (\lambda x.\ s) & \longrightarrow_\beta & (\lambda x.\ s') \end{array}$$

Still to do: define $s[x \leftarrow t]$

Slide 13

Defining Substitution

Easy concept. Small problem: variable capture.

Example: $(\lambda x.\ x\ z)[z \leftarrow x]$

We do **not** want: $(\lambda x.\ x\ x)$ as result.

What do we want?

In $(\lambda y.\ y\ z)\ [z \leftarrow x] = (\lambda y.\ y\ x)$ there would be no problem.

So, solution is: rename bound variables.

Slide 14

Free Variables

Bound variables: in $(\lambda x.\ t)$, x is a bound variable.

Free variables FV of a term:

$$\begin{array}{ll} FV\ (x) & = \{x\} \\ FV\ (c) & = \{\} \\ FV\ (s\ t) & = FV(s) \cup FV(t) \\ FV\ (\lambda x.\ t) & = FV(t) \setminus \{x\} \end{array}$$

Example: $FV(\ \lambda x.\ (\lambda y.\ (\lambda x.\ x)\ y)\ y\ x\ ) = \{y\}$

Term t is called **closed** if $FV(t) = \{\}$

Our problematic substitution example, $(\lambda x.\ x\ z)[z \leftarrow x]$, is problematic because the bound variable x is a free variable of the replacement term "x".

Slide 15

Substitution

$$\begin{array}{lll} x\ [x \leftarrow t] & = t & \\ y\ [x \leftarrow t] & = y & \text{if } x \neq y \\ c\ [x \leftarrow t] & = c & \end{array}$$

$$(s_1\ s_2)\ [x \leftarrow t] = (s_1[x \leftarrow t]\ s_2[x \leftarrow t])$$

$$\begin{array}{ll} (\lambda x.\ s)\ [x \leftarrow t] = (\lambda x.\ s) & \\ (\lambda y.\ s)\ [x \leftarrow t] = (\lambda y.\ s[x \leftarrow t]) & \text{if } x \neq y \text{ and } y \notin FV(t) \\ (\lambda y.\ s)\ [x \leftarrow t] = (\lambda z.\ s[y \leftarrow z][x \leftarrow t]) & \text{if } x \neq y \\ & \text{and } z \notin FV(t) \cup FV(s) \end{array}$$

Slide 16

Substitution Example

$$
\begin{aligned}
& (x\ \ (\lambda x.\ x)\ \ (\lambda y.\ z\ x))[x \leftarrow y] \\
= \ & (x[x \leftarrow y])\ \ ((\lambda x.\ x)[x \leftarrow y])\ \ ((\lambda y.\ z\ x)[x \leftarrow y]) \\
= \ & y\ (\lambda x.\ x)\ (\lambda y'.\ z\ y)
\end{aligned}
$$

Slide 17

α Conversion

Bound names are irrelevant:
$\lambda x.\ x$ and $\lambda y.\ y$ denote the same function.

α conversion:
$s =_\alpha t$ means $s = t$ up to renaming of bound variables.

Formally:

$$
\begin{array}{llllcll}
 & & & & (\lambda x.\ t) & \longrightarrow_\alpha & (\lambda y.\ t[x \leftarrow y]) \text{ if } y \notin FV(t) \\
s & \longrightarrow_\alpha & s' & \Longrightarrow & (s\ t) & \longrightarrow_\alpha & (s'\ t) \\
t & \longrightarrow_\alpha & t' & \Longrightarrow & (s\ t) & \longrightarrow_\alpha & (s\ t') \\
s & \longrightarrow_\alpha & s' & \Longrightarrow & (\lambda x.\ s) & \longrightarrow_\alpha & (\lambda x.\ s')
\end{array}
$$

$$s =_\alpha t \quad \text{iff} \quad s \longrightarrow_\alpha^* t$$

($\longrightarrow_\alpha^*$ = transitive, reflexive closure of $\longrightarrow_\alpha$ = multiple steps)

Slide 18

α Conversion

Equality in Isabelle is equality modulo α conversion:

if $s =_\alpha t$ then s and t are syntactically equal.

Examples:

$$
\begin{array}{ll}
 & x\ (\lambda x\ y.\ x\ y) \\
=_\alpha & x\ (\lambda y\ x.\ y\ x) \\
=_\alpha & x\ (\lambda z\ y.\ z\ y) \\
\neq_\alpha & z\ (\lambda z\ y.\ z\ y) \\
\neq_\alpha & x\ (\lambda x\ x.\ x\ x)
\end{array}
$$

Slide 19

Back to β

We have defined β reduction: $\longrightarrow_\beta$

Some notation and concepts:

- ➜ β **conversion:** $s =_\beta t$ iff $\exists n.\ s \longrightarrow_\beta^* n \wedge t \longrightarrow_\beta^* n$
- ➜ t is **reducible** if there is an s such that $t \longrightarrow_\beta s$
- ➜ $(\lambda x.\ s)\ t$ is called a **redex** (reducible expression)
- ➜ t is reducible iff it contains a redex
- ➜ if it is not reducible, t is in **normal form**

Slide 20

Does every λ term have a normal form?

No!

Example:

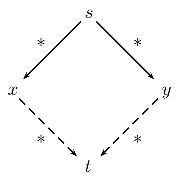
$$
\begin{array}{l}
(\lambda x.\ x\ x)\ (\lambda x.\ x\ x)\ \longrightarrow_\beta \\
(\lambda x.\ x\ x)\ (\lambda x.\ x\ x)\ \longrightarrow_\beta \\
(\lambda x.\ x\ x)\ (\lambda x.\ x\ x)\ \longrightarrow_\beta \dots
\end{array}
$$

(but: $(\lambda x\ y.\ y)\ ((\lambda x.\ x\ x)\ (\lambda x.\ x\ x))\ \longrightarrow_\beta\ \lambda y.\ y$)

λ calculus is not terminating

β reduction is confluent

Confluence: $s \longrightarrow_\beta^* x \wedge s \longrightarrow_\beta^* y \Longrightarrow \exists t.\ x \longrightarrow_\beta^* t \wedge y \longrightarrow_\beta^* t$



Order of reduction does not matter for result
Normal forms in λ calculus are unique

β reduction is confluent

Example:

$(\lambda x\ y.\ y)\ ((\lambda x.\ x\ x)\ a) \longrightarrow_\beta (\lambda x\ y.\ y)\ (a\ a) \longrightarrow_\beta \lambda y.\ y$

$(\lambda x\ y.\ y)\ ((\lambda x.\ x\ x)\ a) \longrightarrow_\beta \lambda y.\ y$

η Conversion

Another case of trivially equal functions: $t = (\lambda x.\ t\ x)$

Definition:
$$
\begin{array}{lllllll}
 & & & (\lambda x.\ t\ x) & \longrightarrow_\eta & t & \text{if } x \notin FV(t) \\
s & \longrightarrow_\eta & s' \Longrightarrow & (s\ t) & \longrightarrow_\eta & (s'\ t) & \\
t & \longrightarrow_\eta & t' \Longrightarrow & (s\ t) & \longrightarrow_\eta & (s\ t') & \\
s & \longrightarrow_\eta & s' \Longrightarrow & (\lambda x.\ s) & \longrightarrow_\eta & (\lambda x.\ s') &
\end{array}
$$

$$s =_\eta t \ \text{ iff } \ \exists n.\ s \longrightarrow_\eta^* n \wedge t \longrightarrow_\eta^* n$$

Example: $(\lambda x.\ f\ x)\ (\lambda y.\ g\ y) \longrightarrow_\eta (\lambda x.\ f\ x)\ g \longrightarrow_\eta f\ g$

- ➜ η reduction is confluent and terminating.
- ➜ $\longrightarrow_{\beta\eta}$ is confluent.
  $\longrightarrow_{\beta\eta}$ means $\longrightarrow_\beta$ and $\longrightarrow_\eta$ steps are both allowed.
- ➜ **Equality in Isabelle is also modulo η conversion.**

In fact ...

NICTA

Equality in Isabelle is modulo α, β, and η conversion.

We will see later why that is possible.

Slide 25

So, what can you do with λ calculus?

NICTA

λ calculus is very expressive, you can encode:

- ➜ logic, set theory
- ➜ turing machines, functional programs, etc.

Examples:

$\texttt{true} \equiv \lambda x\ y.\ x$	$\texttt{if}\ \texttt{true}\ x\ y \longrightarrow_\beta^* x$
$\texttt{false} \equiv \lambda x\ y.\ y$	$\texttt{if}\ \texttt{false}\ x\ y \longrightarrow_\beta^* y$
$\texttt{if} \equiv \lambda z\ x\ y.\ z\ x\ y$	

Now, `not`, `and`, `or`, etc is easy:

$\texttt{not} \equiv \lambda x.\ \texttt{if}\ x\ \texttt{false}\ \texttt{true}$
$\texttt{and} \equiv \lambda x\ y.\ \texttt{if}\ x\ y\ \texttt{false}$
$\texttt{or} \equiv \lambda x\ y.\ \texttt{if}\ x\ \texttt{true}\ y$

Slide 26

More Examples

NICTA

Encoding natural numbers (Church Numerals)

$0 \equiv \lambda f\ x.\ x$
$1 \equiv \lambda f\ x.\ f\ x$
$2 \equiv \lambda f\ x.\ f\ (f\ x)$
$3 \equiv \lambda f\ x.\ f\ (f\ (f\ x))$
$\ldots$

Numeral n takes arguments f and x, applies f n-times to x.

$\texttt{iszero} \equiv \lambda n.\ n\ (\lambda x.\ \texttt{false})\ \texttt{true}$
$\texttt{succ} \equiv \lambda n\ f\ x.\ f\ (n\ f\ x)$
$\texttt{add} \equiv \lambda m\ n.\ \lambda f\ x.\ m\ f\ (n\ f\ x)$

Slide 27

Fix Points

NICTA

$(\lambda x\ f.\ f\ (x\ x\ f))\ \ (\lambda x\ f.\ f\ (x\ x\ f))\ \ t \longrightarrow_\beta$
$(\lambda f.\ f\ ((\lambda x\ f.\ f\ (x\ x\ f))\ \ (\lambda x\ f.\ f\ (x\ x\ f))\ f))\ \ t \longrightarrow_\beta$
$t\ \ ((\lambda x\ f.\ f\ (x\ x\ f))\ (\lambda x\ f.\ f\ (x\ x\ f))\ t)$

$\mu = (\lambda xf.\ f\ (x\ x\ f))\ (\lambda xf.\ f\ (x\ x\ f))$
$\mu\ t \longrightarrow_\beta t\ (\mu\ t) \longrightarrow_\beta t\ (t\ (\mu\ t)) \longrightarrow_\beta t\ (t\ (t\ (\mu\ t))) \longrightarrow_\beta \ldots$

$(\lambda xf.\ f\ (x\ x\ f))\ (\lambda xf.\ f\ (x\ x\ f))$ is Turing's fix point operator

Slide 28

Nice, but ...

As a mathematical foundation, λ does not work. **It is inconsistent.**

- **Frege** (Predicate Logic, $\sim$ 1879):
  allows arbitrary quantification over predicates
- **Russell** (1901): Paradox $R \equiv \{X | X \notin X\}$
- **Whitehead & Russell** (Principia Mathematica, 1910-1913):
  Fix the problem
- **Church** (1930): λ calculus as logic, `true`, `false`, $\wedge$, ... as λ terms

	with	$\{x \mid P\ x\} \equiv \lambda x.\ P\ x \qquad x \in M \equiv M\ x$
Problem:	you can write	$R \equiv \lambda x.\ \texttt{not}\ (x\ x)$
	and get	$(R\ R) =_\beta \texttt{not}\ (R\ R)$

Slide 29

Isabelle Demo

Slide 30

We have learned so far...

- λ calculus syntax
- free variables, substitution
- β reduction
- α and η conversion
- β reduction is confluent
- λ calculus is very expressive (turing complete)
- λ calculus is inconsistent

Slide 31

Exercises

- Reduce $(\lambda x.\ y\ (\lambda v.\ x\ v))\ (\lambda y.\ v\ y)$ to $\beta\eta$ normal form.
- Find an encoding for function fs, sn, and $pair$ such that $fs\ (pair\ a\ b)\ =_\beta\ a$ and $sn\ (pair\ a\ b)\ =_\beta\ b$.
- (harder) Find an encoding of list objects, i.e. for the function $cons$ and nil. Then find an encoding for map (that is, map $f\ [x_1, \ldots, x_n] = [f\ x_1, \ldots, f\ x_n]$), and for $foldl$ (that is, foldl $f\ i\ [x_1, \ldots, x_n] = f\ x_1\ (f\ x_2\ (f\ x_3\ (\ldots (f\ x_n\ i)) \ldots))$)

Slide 32