

COMP4161: Advanced Topics in Software Verification

λ

June Andronick, Christine Rizkallah, Miki Tanaka, Johannes Åman Pohjola
T3/2019

data61.csiro.au

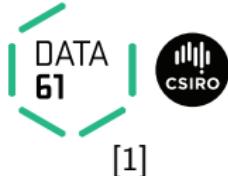


Exercises from last time



- Download and install Isabelle from
<http://mirror.cse.unsw.edu.au/pub/isabelle/>
- Step through the demo files from the lecture web page
- Write your own theory file, look at some theorems in the library, try 'find_theorems'
- How many theorems can help you if you need to prove something containing the term "Suc(Suc x)"?
- What is the name of the theorem for associativity of addition of natural numbers in the library?

Content



→ Intro & motivation, getting started

[1]

→ Foundations & Principles

- Lambda Calculus, natural deduction
- Higher Order Logic, Isar (part 1)
- Term rewriting

[1,2]

[3^a]

[4]

→ Proof & Specification Techniques

- Inductively defined sets, rule induction
- Datatypes, recursion, induction, Isar (part 2)
- Hoare logic, proofs about programs, invariants
- C verification
- Practice, questions, exam prep

[5]

[6, 7^b]

[8]

[9]

[10^c]

^aa1 due; ^ba2 due; ^ca3 due

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

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

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 $(a + b) + 5$

That's it!

Now Formal

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

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

Getting used to the Syntax



Example:

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

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

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

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

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

Computation



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

Example

$$\begin{aligned} & (\lambda x. y. f(yx)) \ 5 \ (\lambda x. x) \xrightarrow{\beta} \\ & (\lambda y. f(y5)) \ (\lambda x. x) \xrightarrow{\beta} \\ & f((\lambda x. x) 5) \xrightarrow{\beta} \\ & f \ 5 \end{aligned}$$

Defining Computation



β reduction:

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

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

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.

Free Variables



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

Free variables FV of a term:

$$FV(x) = \{x\}$$

$$FV(c) = \{\}$$

$$FV(s \ t) = FV(s) \cup FV(t)$$

$$FV(\lambda x. t) = FV(t) \setminus \{x\}$$

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

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

The 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 ”.

Substitution



$$x [x \leftarrow t] = t$$

$$y [x \leftarrow t] = y$$

$$c [x \leftarrow t] = c$$

if $x \neq y$

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

$$(\lambda x. s) [x \leftarrow t] = (\lambda x. s)$$

$$(\lambda y. s) [x \leftarrow t] = (\lambda y. s[x \leftarrow t])$$

$$(\lambda y. s) [x \leftarrow t] = (\lambda z. s[y \leftarrow z][x \leftarrow t])$$

if $x \neq y$ and $y \notin FV(t)$

if $x \neq y$

and $z \notin FV(t) \cup FV(s)$

Substitution Example



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

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:

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

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

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

α Conversion



Equality in Isabelle is equality modulo α conversion:

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

Examples:

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

Back to β



We have defined β reduction: \rightarrow_{β}

Some notation and concepts:

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

Does every λ term have a normal form?



No!

Example:

$$\begin{aligned} (\lambda x. x\ x) \ (\lambda x. x\ x) &\longrightarrow_{\beta} \\ (\lambda x. x\ x) \ (\lambda x. x\ x) &\longrightarrow_{\beta} \\ (\lambda x. x\ x) \ (\lambda x. x\ x) &\longrightarrow_{\beta} \dots \end{aligned}$$

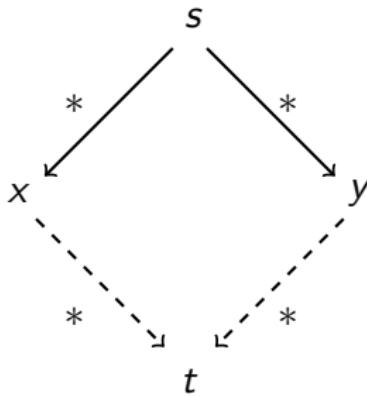
$$(\text{but: } (\lambda x\ y. y) ((\lambda x. x\ x) \ (\lambda x. x\ x))) \longrightarrow_{\beta} \lambda y. y)$$

λ calculus is not terminating

β reduction is confluent



Confluence: $s \xrightarrow{\beta}^* x \wedge s \xrightarrow{\beta}^* y \implies \exists t. x \xrightarrow{\beta}^* t \wedge y \xrightarrow{\beta}^* t$



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

β reduction is confluent



Example:

$$(\lambda x. y. y) ((\lambda x. x x) a) \xrightarrow{\beta} (\lambda x. y. y) (a a) \xrightarrow{\beta} \lambda y. y$$

$$(\lambda x. y. y) ((\lambda x. x x) a) \xrightarrow{\beta} \lambda y. y$$

η Conversion



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

Definition:

$$\begin{array}{lll} & (\lambda x. t x) \xrightarrow{\eta} t & \text{if } x \notin FV(t) \\ s \xrightarrow{\eta} s' & \implies & (s t) \xrightarrow{\eta} (s' t) \\ t \xrightarrow{\eta} t' & \implies & (s t) \xrightarrow{\eta} (s t') \\ s \xrightarrow{\eta} s' & \implies & (\lambda x. s) \xrightarrow{\eta} (\lambda x. s') \\ \\ s =_{\eta} t & \text{iff} & \exists n. s \xrightarrow{\eta}^* n \wedge t \xrightarrow{\eta}^* n \end{array}$$

Example: $(\lambda x. f x) (\lambda y. g y) \xrightarrow{\eta} (\lambda x. f x) g \xrightarrow{\eta} f g$

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

In fact ...



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

We will see later why that is possible.

So, what can you do with λ calculus?



λ calculus is very expressive, you can encode:

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

Examples:

$$\text{true} \equiv \lambda x. y. x$$

$$\text{if true } x \ y \xrightarrow{\beta}^* x$$

$$\text{false} \equiv \lambda x. y. y$$

$$\text{if false } x \ y \xrightarrow{\beta}^* y$$

$$\text{if } \equiv \lambda z. x \ y. z \ x \ y$$

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

$$\text{not} \equiv \lambda x. \text{if } x \text{ false true}$$

$$\text{and} \equiv \lambda x. y. \text{if } x \ y \text{ false}$$

$$\text{or} \equiv \lambda x. y. \text{if } x \text{ true } y$$

More Examples



Encoding natural numbers (Church Numerals)

$$0 \equiv \lambda f x. x$$

$$1 \equiv \lambda f x. f x$$

$$2 \equiv \lambda f x. f (f x)$$

$$3 \equiv \lambda f x. f (f (f x))$$

...

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

$$\text{iszero} \equiv \lambda n. n (\lambda x. \text{false}) \text{ true}$$

$$\text{succ} \equiv \lambda n f x. f (n f x)$$

$$\text{add} \equiv \lambda m n. \lambda f x. m f (n f x)$$

Fix Points



$$\begin{aligned} & (\lambda x f. f (x x f)) \ (\lambda x f. f (x x f)) \ t \longrightarrow_{\beta} \\ & (\lambda f. f ((\lambda x f. f (x x f)) \ (\lambda x f. f (x x f)) \ f)) \ t \longrightarrow_{\beta} \\ & t \ ((\lambda x f. f (x x f)) \ (\lambda x f. f (x x f)) \ t) \end{aligned}$$

$$\begin{aligned} \mu &= (\lambda x f. f (x x f)) \ (\lambda x f. f (x x f)) \\ \mu \ t &\longrightarrow_{\beta} t \ (\mu \ t) \longrightarrow_{\beta} t \ (t \ (\mu \ t)) \longrightarrow_{\beta} t \ (t \ (t \ (\mu \ t))) \longrightarrow_{\beta} \dots \end{aligned}$$

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

Nice, but ...



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

- **Frege** (Predicate Logic, ~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

Problem:

with $\{x | P x\} \equiv \lambda x. P x \quad x \in M \equiv M x$

you can write $R \equiv \lambda x. \text{not } (x x)$

and get $(R R) =_{\beta} \text{not } (R R)$

because $(R R) = (\lambda x. \text{not } (x x)) R \longrightarrow_{\beta} \text{not } (R R)$

Isabelle Demo

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