



COMP 4161
NICTA Advanced Course

Advanced Topics in Software Verification

Toby Murray, June Andronick, Gerwin Klein



Slide 1



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 like "Suc(Suc x)"?
- What is the name of the theorem for associativity of addition of natural numbers in the library?

Slide 2

Content

- Intro & motivation, getting started [1]
- Foundations & Principles
 - Lambda Calculus, natural deduction [1,2]
 - Higher Order Logic [3^a]
 - Term rewriting [4]
- Proof & Specification Techniques
 - Inductively defined sets, rule induction [5]
 - Datatypes, recursion, induction [6, 7]
 - Hoare logic, proofs about programs, C verification [8^b, 9]
 - (mid-semester break)
 - Writing Automated Proof Methods [10]
 - `Isar`, `codegen`, `typeclasses`, `locales` [11^c, 12]

^aa1 due; ^ba2 due; ^ca3 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

THAT'S IT!



Slide 7

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

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

$(\lambda x y. f (y x)) 5 (\lambda x. x) \rightarrow_{\beta}$

$(\lambda y. f (y 5)) (\lambda x. x) \rightarrow_{\beta}$

$f ((\lambda x. x) 5) \rightarrow_{\beta}$

$f 5$

Slide 12



Defining Computation



β reduction:

$$\begin{aligned} & (\lambda x. s) t \rightarrow_{\beta} s[x \leftarrow t] \\ s \rightarrow_{\beta} s' & \implies (s t) \rightarrow_{\beta} (s' t) \\ t \rightarrow_{\beta} t' & \implies (s t) \rightarrow_{\beta} (s t') \\ s \rightarrow_{\beta} s' & \implies (\lambda x. s) \rightarrow_{\beta} (\lambda x. s') \end{aligned}$$

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:

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

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

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 ".

Slide 15

Substitution



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

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

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

$$(\lambda y. s)[x \leftarrow t] = (\lambda y. s[x \leftarrow t]) \quad \text{if } x \neq y \text{ and } y \notin FV(t)$$

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

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:

$$\begin{aligned}
 s & \rightarrow_{\alpha} s' \iff (\lambda x. t) \rightarrow_{\alpha} (\lambda y. t[x \leftarrow y]) \text{ if } y \notin FV(t) \\
 t & \rightarrow_{\alpha} t' \iff (s t) \rightarrow_{\alpha} (s' t) \\
 s & \rightarrow_{\alpha} s' \iff (\lambda x. s) \rightarrow_{\alpha} (\lambda x. s')
 \end{aligned}$$

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

(\rightarrow_{α}^* = transitive, reflexive closure of \rightarrow_{α} = 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:

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



Slide 19

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**



Slide 20

Does every λ term have a normal form?

No!

Example:

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

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

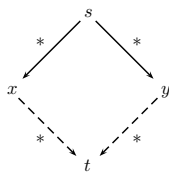
λ calculus is not terminating

Slide 21



β reduction is confluent

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



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

Slide 22



β reduction is confluent

Example:

$$\begin{aligned} (\lambda x y. y) ((\lambda x. x x) a) &\rightarrow_{\beta} (\lambda x y. y) (a a) \rightarrow_{\beta} \lambda y. y \\ (\lambda x y. y) ((\lambda x. x x) a) &\rightarrow_{\beta} \lambda y. y \end{aligned}$$

Slide 23



η Conversion

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

$$\begin{aligned} \text{Definition: } \begin{array}{l} s \rightarrow_{\eta} s' \implies (\lambda x. t x) \rightarrow_{\eta} t \quad \text{if } x \notin FV(t) \\ t \rightarrow_{\eta} t' \implies (s t) \rightarrow_{\eta} (s' t) \\ s \rightarrow_{\eta} s' \implies (\lambda x. s) \rightarrow_{\eta} (\lambda x. s') \end{array} \\ s =_{\eta} t \quad \text{iff } \exists n. s \rightarrow_{\eta}^* n \wedge t \rightarrow_{\eta}^* n \end{aligned}$$

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

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

Slide 24



In fact ...

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?

λ calculus is very expressive, you can encode:

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

Examples:

```

true  ≡ λx y. x           if true x y →β* x
false ≡ λx y. y           if false x y →β* y
if    ≡ λz x y. z x y

```

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

```

not ≡ λx. if x false true
and ≡ λx y. if x y false
or  ≡ λx y. if x true y

```



Slide 26

More Examples

Encoding natural numbers (Church Numerals)

```

0 ≡ λf x. x
1 ≡ λf x. f x
2 ≡ λf x. f (f x)
3 ≡ λf x. f (f (f x))
...

```

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

```

iszero ≡ λn. n (λx. false) true
succ   ≡ λn f x. f (n f x)
add    ≡ λm n. λf x. m f (n f x)

```



Slide 27

Fix Points

```

(λx f. f (x x f)) (λx f. f (x x f)) t →β
(λf. f ((λx f. f (x x f)) (λx f. f (x x f)) f)) t →β
t ((λx f. f (x x f)) (λx f. f (x x f)) t)

```

```

μ = (λx f. f (x x f)) (λx f. f (x x f))
μ t →β t (μ t) →β t (t (μ t)) →β t (t (t (μ t))) →β ...

```

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



Slide 28

Nice, but ...



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

- **Frege** (Predicate Logic, ~ 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 $(R R) = (\lambda x. \text{not } (x x)) R \rightarrow_{\beta} \text{not } (R R)$

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