COMP4011/8011
Advanced Topics in
Formal Methods and Programming Languages

– Software Verification with Isabelle/HOL –

Peter Höfner

July 21, 2024

Section 3

λ-Calculus

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 $(a + b) + 5$

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

Remember: $(\lambda x.\ t)\ a$ is evaluated (noted $\longrightarrow_\beta$) to
t where x is replaced by a

Example:

$$(\lambda x\ y.\ Suc\ x\ =\ y)\ 3 \longrightarrow_\beta$$
$$(\lambda x.\ (\lambda y.\ Suc\ x\ =\ y))\ 3 \longrightarrow_\beta$$
$$(\lambda y.\ Suc\ 3\ =\ y)$$

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

Defining Computation

β reduction:

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

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

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:

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

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

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

$$
\begin{array}{lll}
x\,[x \leftarrow t] & = t & \\
y\,[x \leftarrow t] & = y & \text{if } x \neq y \\
c\,[x \leftarrow t] & = c & \\
 & & \\
(s_1\ s_2)\,[x \leftarrow t] & = (s_1[x \leftarrow t]\ s_2[x \leftarrow t]) & \\
 & & \\
(\lambda x.\ s)\,[x \leftarrow t] & = (\lambda x.\ s) & \\
(\lambda y.\ s)\,[x \leftarrow t] & = (\lambda y.\ s[x \leftarrow t]) & \text{if } x \neq y \text{ and } y \notin FV(t) \\
(\lambda y.\ s)\,[x \leftarrow t] & = (\lambda z.\ s[y \leftarrow z][x \leftarrow t]) & \text{if } x \neq y \\
 & & \text{and } z \notin FV(t) \cup FV(s)
\end{array}
$$

Substitution Example

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

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:

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

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

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

α Conversion

Equality in Isabelle is equality modulo α conversion:

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

Examples:

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

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**

Does every λ-term have a normal form?

No!

Example:

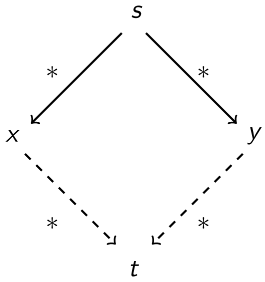
$$
\begin{aligned}
(\lambda x.\ x\ x)\ (\lambda x.\ x\ x) &\longrightarrow_\beta \\
(\lambda x.\ x\ x)\ (\lambda x.\ x\ x) &\longrightarrow_\beta \\
(\lambda x.\ x\ x)\ (\lambda x.\ x\ x) &\longrightarrow_\beta \ldots
\end{aligned}
$$

(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:

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

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

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 ...

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

We will see later why that is possible.

Isabelle Demo

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:

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

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

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

Now, $\texttt{not}$, $\texttt{and}$, $\texttt{or}$, etc is easy:

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

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

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

More Examples

Encoding natural numbers (Church Numerals)

$$
\begin{aligned}
0 &\equiv \lambda f\ x.\ x \\
1 &\equiv \lambda f\ x.\ f\ x \\
2 &\equiv \lambda f\ x.\ f\ (f\ x) \\
3 &\equiv \lambda f\ x.\ f\ (f\ (f\ x)) \\
&\ldots
\end{aligned}
$$

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

$$
\begin{aligned}
\mathtt{iszero} &\equiv \lambda n.\ n\ (\lambda x.\ \mathtt{false})\ \mathtt{true} \\
\mathtt{succ} &\equiv \lambda n\ f\ x.\ f\ (n\ f\ x) \\
\mathtt{add} &\equiv \lambda m\ n.\ \lambda f\ x.\ m\ f\ (n\ f\ x)
\end{aligned}
$$

Fix Points

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

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

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

Nice, but ...

As a mathematical foundation, λ does not work.

It resulted in an inconsistent logic.

- **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

Problem:

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

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 results in an inconsistent logic