NICTA Advanced Course

Theorem Proving
Principles, Techniques, Applications

λ

Content

- ➜ Intro & motivation, getting started with Isabelle
- ➜ **Foundations & Principles**
  - **Lambda Calculus**
  - Higher Order Logic, natural deduction
  - Term rewriting
- ➜ Proof & Specification Techniques
  - Datatypes, recursion, induction
  - Inductively defined sets, rule induction
  - Calculational reasoning, mathematics style proofs
  - Hoare logic, proofs about programs

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

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

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

Basic intuition:

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

- ➜ 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

- ➜ 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

- ➜ 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	$f(x)$
write	$f\ x$

FUNCTION APPLICATION

For applying arguments to functions

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

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

Function Application

For applying arguments to functions

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

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

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

(computation!)

Function Application

For applying arguments to functions

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

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

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

SYNTAX

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

$v, x \in V, \quad c \in C, \quad V, C$ 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 $\lambda y\ x.\ x\ y$

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

Example:

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

Example:

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

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

GETTING USED TO THE SYNTAX

Example:

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

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

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

Example:

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

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

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

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

Example:

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

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

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

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

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

Intuition: replace parameter by argument

this is called β-reduction

Example

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

Intuition: replace parameter by argument

this is called β-reduction

Example

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

Intuition: replace parameter by argument

this is called β-reduction

Example

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

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

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

Intuition: replace parameter by argument

this is called β-reduction

Example

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

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

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

$f\ 5$

Defining Computation

β reduction:

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

β reduction:

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

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

Easy concept. Small problem: variable capture.

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

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?

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.

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: $FV(\quad \lambda x.\ (\lambda y.\ (\lambda x.\ x)\ y)\ y\ x \quad)$

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

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

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

Substitution

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

$$y\ [x \leftarrow t] = y \qquad \text{if } x \neq y$$

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

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

Substitution

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

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

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

Substitution

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

$$y\ [x \leftarrow t] = y \qquad \text{if } x \neq y$$

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

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

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

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

SUBSTITUTION

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

$$y\ [x \leftarrow t] = y \qquad \text{if } x \neq y$$

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

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

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

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

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

Substitution

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

$$y\,[x \leftarrow t] = y \qquad \text{if } x \neq y$$

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

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

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

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

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

SUBSTITUTION EXAMPLE

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

Substitution Example

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

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.

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:

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

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:

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

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

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

Equality in Isabelle is equality modulo α conversion:

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

Examples:

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

Equality in Isabelle is equality modulo α conversion:

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

Examples:

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

Equality in Isabelle is equality modulo α conversion:

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

Examples:

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

Equality in Isabelle is equality modulo α conversion:

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

Examples:

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

Equality in Isabelle is equality modulo α conversion:

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

Examples:

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

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

Some notation and concepts:

➜ β **conversion:** $s =_\beta t$ iff $\exists n.\ s \longrightarrow_\beta^* n \wedge t \longrightarrow_\beta^* n$

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 $t \longrightarrow_\beta s$

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)

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

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**

We have defi ned 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**
- ➜ t **has a normal form** if there is an irreducible s such that $t \longrightarrow^*_\beta s$

Example:

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

Example:

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

Does every λ term have a normal form?

No!

Example:

$$(\lambda x.\, x\, x)\, (\lambda x.\, x\, x) \longrightarrow_\beta$$
$$(\lambda x.\, x\, x)\, (\lambda x.\, x\, x) \longrightarrow_\beta$$
$$(\lambda x.\, x\, x)\, (\lambda x.\, x\, x) \longrightarrow_\beta \ldots$$

No!

Example:

$$(\lambda x.\ x\ x)\ (\lambda x.\ x\ x)\ \longrightarrow_\beta$$
$$(\lambda x.\ x\ x)\ (\lambda x.\ x\ x)\ \longrightarrow_\beta$$
$$(\lambda x.\ x\ x)\ (\lambda x.\ x\ x)\ \longrightarrow_\beta \ldots$$

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

No!

Example:

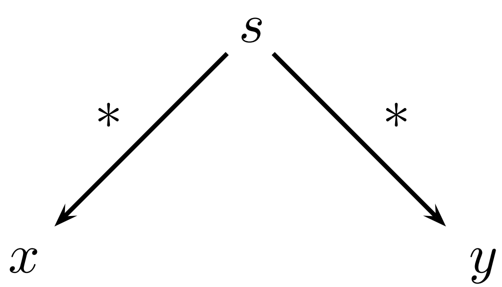
$$(\lambda x.\ x\ x)\ (\lambda x.\ x\ x)\ \longrightarrow_\beta$$
$$(\lambda x.\ x\ x)\ (\lambda x.\ x\ x)\ \longrightarrow_\beta$$
$$(\lambda x.\ x\ x)\ (\lambda x.\ x\ x)\ \longrightarrow_\beta \ldots$$

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

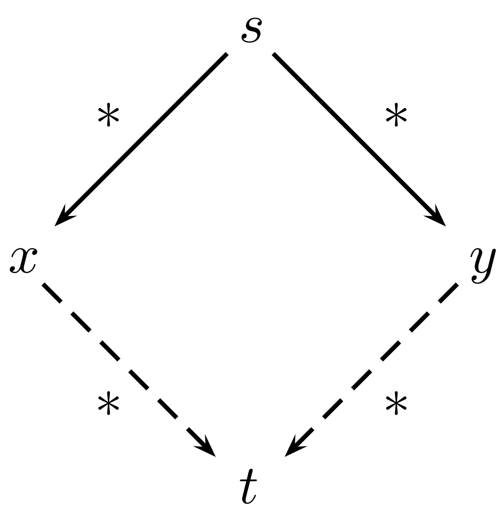
λ calculus is not terminating

β REDUCTION IS CONFLUENT

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



Confluence: $s \longrightarrow_{\beta}^{*} x \land s \longrightarrow_{\beta}^{*} y \Longrightarrow \exists t.\ x \longrightarrow_{\beta}^{*} t \land y \longrightarrow_{\beta}^{*} t$



Order of reduction does not matter for result

Normal forms in λ calculus are unique

Example:

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

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

Example:

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

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

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: $t = (\lambda x.\ t\ x)$

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

Defi nition:

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

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

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

η Conversion

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

Definition:

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

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

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

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

Defi nition:

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

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

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

η CONVERSION

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

Definition:

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

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

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.**

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

We will see next lecture 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:

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:

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

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

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

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:

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

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

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

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:

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

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

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:

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Fix Points

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

Fix Points

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

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

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

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

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

Fix Points

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

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

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

$$\mu = (\lambda xf.\ f\ (x\ x\ f))\ (\lambda xf.\ f\ (x\ x\ f))$$

$$\mu\ t \longrightarrow_\beta t\ (\mu\ t) \longrightarrow_\beta t\ (t\ (\mu\ t)) \longrightarrow_\beta t\ (t\ (t\ (\mu\ t))) \longrightarrow_\beta \ldots$$

Fix Points

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

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

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

$\mu = (\lambda xf.\ f\ (x\ x\ f))\ (\lambda xf.\ f\ (x\ x\ f))$

$\mu\ t \longrightarrow_\beta t\ (\mu\ t) \longrightarrow_\beta t\ (t\ (\mu\ t)) \longrightarrow_\beta t\ (t\ (t\ (\mu\ t))) \longrightarrow_\beta \ldots$

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

NICE, BUT ...

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

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

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

Problem:

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

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

Problem:

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

NICE, BUT ...

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

➜ **Frege** (Predicate Logic, $\sim$ 1879):
allows arbitrary quantification over predicates

➜ **Russel** (1901): Paradox $R \equiv \{X | X \notin X\}$

➜ **Whitehead & Russel** (Principia Mathematica, 1910-1913):
Fix the problem

➜ **Church** (1930): λ calculus as logic, `true`, `false`, $\wedge$, ... as λ terms

Problem:

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

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

Nice, but ...

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

- ➜ **Frege** (Predicate Logic, $\sim$ 1879):
  allows arbitrary quantification over predicates
- ➜ **Russel** (1901): Paradox $R \equiv \{X | X \notin X\}$
- ➜ **Whitehead & Russel** (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	$R \equiv \lambda x.$ `not` $(x\ x)$
and get	$(R\ R) =_\beta$ `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 is inconsistent

ISABELLE DEMO

EXERCISES

- ➜ Play around with the syntax. Enter a number of λ terms into Isabelle.
- ➜ Not all λ terms are accepted by Isabelle. Which are not? Why?
- ➜ Evaluate the substitution $(y\ (\lambda v.\ x\ v))[x \leftarrow (\lambda y.\ v\ y)]$ on paper.
- ➜ Reduce $(\lambda n.\ \lambda f\ x.\ f\ (n\ f\ x))\ ((\lambda n.\ \lambda f\ x.\ f\ (n\ f\ x))\ (\lambda f\ x.\ x))$ to its β normal form on paper and in Isabelle.
- ➜ Pairs in λ calculus: define functions fs, sn, and $pair$ such that $fs\ (pair\ a\ b)\ \longrightarrow_\beta^*\ a$ and $sn\ (pair\ a\ b)\ \longrightarrow_\beta^*\ b$
- ➜ What can be done to fix the inconsistency in λ calculus?