

Functional Programming

WS 2015/16

Cezary Kaliszyk (VO+PS)

Yann Savoye (PS)

Łukasz Czajka (PS)

Computational Logic
Institute of Computer Science
University of Innsbruck

week 05



Overview

- Week 5 - λ -Calculus
 - Summary of Week 4
 - λ -Calculus - Introduction
 - λ -Calculus - Formalities
 - The λ Interpreter `lips`



Overview

- Week 5 - λ -Calculus
 - Summary of Week 4
 - λ -Calculus - Introduction
 - λ -Calculus - Formalities
 - The λ Interpreter `lips`



Summary of Week 4

Binary Trees

- at most 2 children per node
- applications
 - search trees
 - Huffman coding

Huffman Coding

- Idea: use shortest codewords for most frequent symbols
- Application: (lossless) data compression

Overview

- Week 5 - λ -Calculus
 - Summary of Week 4
 - λ -Calculus - Introduction
 - λ -Calculus - Formalities
 - The λ Interpreter `lips`



This Week

Practice I

OCaml introduction, lists, strings, trees

Theory I

lambda-calculus, evaluation strategies, induction,
reasoning about functional programs

Practice II

efficiency, tail-recursion, combinator-parsing,

Theory II

type checking, type inference

Advanced Topics

lazy evaluation, infinite data structures, dependent types, monads

Overview

- Week 5 - λ -Calculus
 - Summary of Week 4
 - λ -Calculus - Introduction
 - λ -Calculus - Formalities
 - The λ Interpreter `lips`



Origin

Goal

- find a framework in which every algorithm can be defined
- universal language

Origin

Goal

- find a framework in which **every** algorithm can be defined
- universal language

Origin

Goal

- find a framework in which every algorithm can be defined
- universal language

Result

- Turing machines (Turing, 1930s)
- λ -Calculus (Church, 1930s)
- ...

Origin

Goal

- find a framework in which every algorithm can be defined
- universal language

Result

- Turing machines (Turing, 1930s)
- λ -Calculus (Church, 1930s)
- ...

Syntax

λ -Terms

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

$\mathcal{T}(\mathcal{V})$ set of all λ -terms over set of variables \mathcal{V}

Syntax

λ -Terms

$$t ::= \overbrace{x}^{\text{Variable}} \mid (\lambda x.t) \mid (t t)$$

$\mathcal{T}(\mathcal{V})$ set of all λ -terms over set of variables \mathcal{V}

Syntax

λ -Terms

$$t ::= x \mid \underbrace{(\lambda x. t)}_{\text{Abstraction}} \mid (t t)$$

$\mathcal{T}(\mathcal{V})$ set of all λ -terms over set of variables \mathcal{V}

Syntax

λ -Terms

Application

$$t ::= x \mid (\lambda x.t) \mid \overbrace{(t t)}$$

$\mathcal{T}(\mathcal{V})$ set of all λ -terms over set of variables \mathcal{V}

Syntax

λ -Terms

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

$\mathcal{T}(\mathcal{V})$ set of **all** λ -terms over set of variables \mathcal{V}

Syntax

λ -Terms

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

$\mathcal{T}(\mathcal{V})$ set of all λ -terms over set of variables \mathcal{V}

Conventions

$$\begin{aligned} & (\lambda x.x) \\ & (\lambda x.(\lambda y.x)) \\ & (\lambda x.(\lambda y.(\lambda z.((x z) (y z)))))) \\ & (\lambda x.((\lambda y.(\lambda z.(z y))) x)) \end{aligned}$$

Syntax

λ -Terms

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

$\mathcal{T}(\mathcal{V})$ set of all λ -terms over set of variables \mathcal{V}

Conventions (omit outermost parentheses)

$$\begin{aligned} &\lambda x.x \\ &\lambda x.(\lambda y.x) \\ &\lambda x.(\lambda y.(\lambda z.((x z) (y z)))) \\ &\lambda x.((\lambda y.(\lambda z.(z y))) x) \end{aligned}$$

Syntax

λ -Terms

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

$\mathcal{T}(\mathcal{V})$ set of all λ -terms over set of variables \mathcal{V}

Conventions (combine nested lambdas)

$$\lambda x.x$$
$$\lambda xy.x$$
$$\lambda xyz.((x z) (y z))$$
$$\lambda x.((\lambda yz.(z y)) x)$$

Syntax

λ -Terms

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

$\mathcal{T}(\mathcal{V})$ set of all λ -terms over set of variables \mathcal{V}

Conventions (application is left-associative and binds strongest)

$$\lambda x.x$$

$$\lambda xy.x$$

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

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

Intuition

Example

λ -terms

- $\lambda x. \text{add } x \ \bar{1}$
- $(\lambda x. \text{add } x \ \bar{1}) \ \bar{2}$
- $\text{if true } \bar{1} \ \bar{0}$
- $\text{pair } \bar{2} \ \bar{4}$
- $\text{fst}(\text{pair } \bar{2} \ \bar{4})$
- $\lambda xy. \text{add } x \ y$
- $\lambda x. (\lambda y. \text{add } x \ y)$

Intuition

Example

λ -terms

OCaml

- $\lambda x. \text{add } x \ \bar{1}$
- $(\lambda x. \text{add } x \ \bar{1}) \ \bar{2}$
- $\text{if true } \bar{1} \ \bar{0}$
- $\text{pair } \bar{2} \ \bar{4}$
- $\text{fst}(\text{pair } \bar{2} \ \bar{4})$
- $\lambda xy. \text{add } x \ y$
- $\lambda x. (\lambda y. \text{add } x \ y)$

Intuition

Example

λ -terms

- $\lambda x. \text{add } x \ \bar{1}$
- $(\lambda x. \text{add } x \ \bar{1}) \ \bar{2}$
- $\text{if true } \bar{1} \ \bar{0}$
- $\text{pair } \bar{2} \ \bar{4}$
- $\text{fst}(\text{pair } \bar{2} \ \bar{4})$
- $\lambda xy. \text{add } x \ y$
- $\lambda x. (\lambda y. \text{add } x \ y)$

OCaml

- `fun x -> x+1`

Intuition

Example

λ -terms

- $\lambda x. \text{add } x \ \bar{1}$
- $(\lambda x. \text{add } x \ \bar{1}) \ \bar{2}$
- $\text{if true } \bar{1} \ \bar{0}$
- $\text{pair } \bar{2} \ \bar{4}$
- $\text{fst}(\text{pair } \bar{2} \ \bar{4})$
- $\lambda xy. \text{add } x \ y$
- $\lambda x. (\lambda y. \text{add } x \ y)$

OCaml

- `fun x -> x+1`
- `(fun x -> x+1) 2 \rightarrow^+ 3`

Intuition

Example

λ -terms

- $\lambda x. \text{add } x \ \bar{1}$
- $(\lambda x. \text{add } x \ \bar{1}) \ \bar{2}$
- $\text{if true } \bar{1} \ \bar{0}$
- $\text{pair } \bar{2} \ \bar{4}$
- $\text{fst}(\text{pair } \bar{2} \ \bar{4})$
- $\lambda xy. \text{add } x \ y$
- $\lambda x. (\lambda y. \text{add } x \ y)$

OCaml

- `fun x -> x+1`
- `(fun x -> x+1) 2 \rightarrow^+ 3`
- `if true then 1 else 0 \rightarrow 1`

Intuition

Example

λ -terms

- $\lambda x. \text{add } x \ \bar{1}$
- $(\lambda x. \text{add } x \ \bar{1}) \ \bar{2}$
- $\text{if true } \bar{1} \ \bar{0}$
- $\text{pair } \bar{2} \ \bar{4}$
- $\text{fst}(\text{pair } \bar{2} \ \bar{4})$
- $\lambda xy. \text{add } x \ y$
- $\lambda x. (\lambda y. \text{add } x \ y)$

OCaml

- `fun x -> x+1`
- `(fun x -> x+1) 2 \rightarrow^+ 3`
- `if true then 1 else 0 \rightarrow 1`
- `(2,4)`

Intuition

Example

λ -terms

- $\lambda x. \text{add } x \bar{1}$
- $(\lambda x. \text{add } x \bar{1}) \bar{2}$
- $\text{if true } \bar{1} \bar{0}$
- $\text{pair } \bar{2} \bar{4}$
- $\text{fst}(\text{pair } \bar{2} \bar{4})$
- $\lambda xy. \text{add } x y$
- $\lambda x. (\lambda y. \text{add } x y)$

OCaml

- `fun x -> x+1`
- `(fun x -> x+1) 2 \rightarrow^+ 3`
- `if true then 1 else 0 \rightarrow 1`
- `(2,4)`
- `fst(2,4) \rightarrow 2`

Intuition

Example

λ -terms

- $\lambda x. \text{add } x \ \bar{1}$
- $(\lambda x. \text{add } x \ \bar{1}) \ \bar{2}$
- $\text{if true } \bar{1} \ \bar{0}$
- $\text{pair } \bar{2} \ \bar{4}$
- $\text{fst}(\text{pair } \bar{2} \ \bar{4})$
- $\lambda xy. \text{add } x \ y$
- $\lambda x. (\lambda y. \text{add } x \ y)$

OCaml

- `fun x -> x+1`
- `(fun x -> x+1) 2 \rightarrow^+ 3`
- `if true then 1 else 0 \rightarrow 1`
- `(2,4)`
- `fst(2,4) \rightarrow 2`
- `fun x y -> x + y`

Intuition

Example

λ -terms

- $\lambda x. \text{add } x \ \bar{1}$
- $(\lambda x. \text{add } x \ \bar{1}) \ \bar{2}$
- $\text{if true } \bar{1} \ \bar{0}$
- $\text{pair } \bar{2} \ \bar{4}$
- $\text{fst}(\text{pair } \bar{2} \ \bar{4})$
- $\lambda xy. \text{add } x \ y$
- $\lambda x. (\lambda y. \text{add } x \ y)$

OCaml

- `fun x -> x+1`
- `(fun x -> x+1) 2 \rightarrow^+ 3`
- `if true then 1 else 0 \rightarrow 1`
- `(2,4)`
- `fst(2,4) \rightarrow 2`
- `fun x y -> x + y`
- `fun x -> fun y -> x + y`

Intuition

Example

λ -terms

- $\lambda x. \text{add } x \bar{1}$
- $(\lambda x. \text{add } x \bar{1}) \bar{2}$
- $\text{if true } \bar{1} \bar{0}$
- $\text{pair } \bar{2} \bar{4}$
- $\text{fst}(\text{pair } \bar{2} \bar{4})$
- $\lambda xy. \text{add } x y$
- $\lambda x. (\lambda y. \text{add } x y)$

OCaml

- `fun x -> x+1`
- `(fun x -> x+1) 2 \rightarrow^+ 3`
- `if true then 1 else 0 \rightarrow 1`
- `(2,4)`
- `fst(2,4) \rightarrow 2`
- `fun x y -> x + y`
- `fun x -> fun y -> x + y`

Remark

' $\bar{0}$ ', ' $\bar{1}$ ', ' $\bar{2}$ ', ' $\bar{3}$ ', ' $\bar{4}$ ', 'add', 'fst', 'if', 'pair', and 'true' are just abbreviations for more complex λ -terms

Overview

- Week 5 - λ -Calculus
 - Summary of Week 4
 - λ -Calculus - Introduction
 - λ -Calculus - Formalities
 - The λ Interpreter `lips`



Subterms

Definition

$\text{Sub}(t)$ is set of subterms of t

$$\text{Sub}(t) \stackrel{\text{def}}{=} \begin{cases} \{t\} & t = x \\ \{t\} \cup \text{Sub}(u) & t = \lambda x.u \\ \{t\} \cup \text{Sub}(u) \cup \text{Sub}(v) & t = u v \end{cases}$$

Subterms

Definition

$\text{Sub}(t)$ is set of subterms of t

$$\text{Sub}(t) \stackrel{\text{def}}{=} \begin{cases} \{t\} & t = x \\ \{t\} \cup \text{Sub}(u) & t = \lambda x.u \\ \{t\} \cup \text{Sub}(u) \cup \text{Sub}(v) & t = u v \end{cases}$$

Example

$$\text{Sub}(\lambda xy.x)$$

Subterms

Definition

$\text{Sub}(t)$ is set of subterms of t

$$\text{Sub}(t) \stackrel{\text{def}}{=} \begin{cases} \{t\} & t = x \\ \{t\} \cup \text{Sub}(u) & t = \lambda x.u \\ \{t\} \cup \text{Sub}(u) \cup \text{Sub}(v) & t = u v \end{cases}$$

Example

$$\text{Sub}(\lambda xy.x) = \{\lambda xy.x\} \cup \text{Sub}(\lambda y.x)$$

Subterms

Definition

$\text{Sub}(t)$ is set of subterms of t

$$\text{Sub}(t) \stackrel{\text{def}}{=} \begin{cases} \{t\} & t = x \\ \{t\} \cup \text{Sub}(u) & t = \lambda x.u \\ \{t\} \cup \text{Sub}(u) \cup \text{Sub}(v) & t = u v \end{cases}$$

Example

$$\begin{aligned} \text{Sub}(\lambda xy.x) &= \{\lambda xy.x\} \cup \text{Sub}(\lambda y.x) \\ &= \{\lambda xy.x, \lambda y.x\} \cup \text{Sub}(x) \end{aligned}$$

Subterms

Definition

$\text{Sub}(t)$ is set of subterms of t

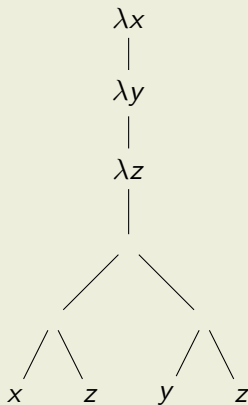
$$\text{Sub}(t) \stackrel{\text{def}}{=} \begin{cases} \{t\} & t = x \\ \{t\} \cup \text{Sub}(u) & t = \lambda x.u \\ \{t\} \cup \text{Sub}(u) \cup \text{Sub}(v) & t = u v \end{cases}$$

Example

$$\begin{aligned} \text{Sub}(\lambda xy.x) &= \{\lambda xy.x\} \cup \text{Sub}(\lambda y.x) \\ &= \{\lambda xy.x, \lambda y.x\} \cup \text{Sub}(x) \\ &= \{\lambda xy.x, \lambda y.x, x\} \end{aligned}$$

Syntax Trees

Example

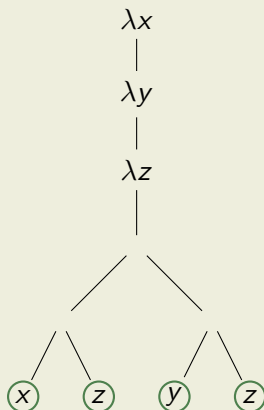


$$t = \lambda xyz.x z (y z)$$

$$\mathcal{Sub}(t) = \{t, \lambda yz.x z (y z), \\ \lambda z.x z (y z), \\ x z (y z), x z, y z, \\ x, z, y\}$$

Syntax Trees

Example

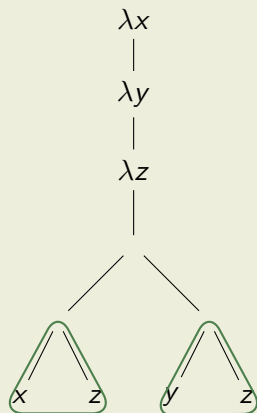


$$t = \lambda xyz.x z (y z)$$

$$\mathcal{Sub}(t) = \{t, \lambda yz.x z (y z), \lambda z.x z (y z), x z (y z), x z, y z, x, z, y\}$$

Syntax Trees

Example

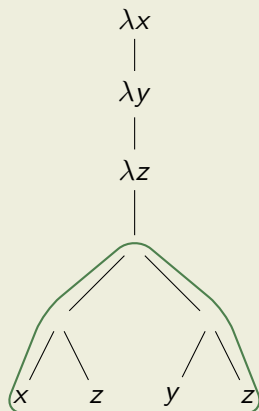


$$t = \lambda xyz.x z (y z)$$

$$\text{Sub}(t) = \{t, \lambda yz.x z (y z), \\ \lambda z.x z (y z), \\ x z (y z), x z, y z, \\ x, z, y\}$$

Syntax Trees

Example

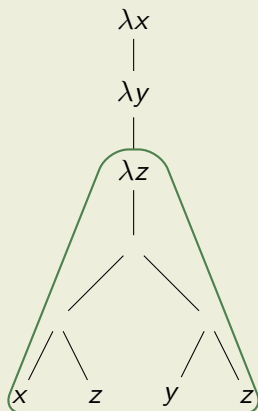


$$t = \lambda xyz.x z (y z)$$

$$\mathcal{S}ub(t) = \{t, \lambda yz.x z (y z), \lambda z.x z (y z), x z (y z), x z, y z, x, z, y\}$$

Syntax Trees

Example

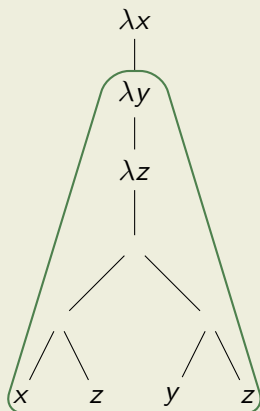


$$t = \lambda xyz.x z (y z)$$

$$\mathcal{S}ub(t) = \{t, \lambda yz.x z (y z), \\ \lambda z.x z (y z), \\ x z (y z), x z, y z, \\ x, z, y\}$$

Syntax Trees

Example

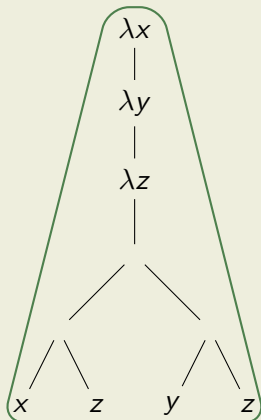


$$t = \lambda xyz.x z (y z)$$

$$\text{Sub}(t) = \{t, \lambda yz.x z (y z), \\ \lambda z.x z (y z), \\ x z (y z), x z, y z, \\ x, z, y\}$$

Syntax Trees

Example



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

$$\text{Sub}(t) = \{t, \lambda yz.x z (y z), \lambda z.x z (y z), x z (y z), x z, y z, x, z, y\}$$

Variables

Definition

variables

$$\mathcal{V}\text{ar}(t) \stackrel{\text{def}}{=} \begin{cases} \{t\} & t = x \\ \{x\} \cup \mathcal{V}\text{ar}(u) & t = \lambda x.u \\ \mathcal{V}\text{ar}(u) \cup \mathcal{V}\text{ar}(v) & t = u v \end{cases}$$

Free and Bound Variables

Definition

free variables

$$\mathcal{FVar}(t) \stackrel{\text{def}}{=} \begin{cases} \{t\} & t = x \\ \mathcal{FVar}(u) \setminus \{x\} & t = \lambda x.u \\ \mathcal{FVar}(u) \cup \mathcal{FVar}(v) & t = u v \end{cases}$$

Free and Bound Variables

Definition

free variables

$$\mathcal{FVar}(t) \stackrel{\text{def}}{=} \begin{cases} \{t\} & t = x \\ \mathcal{FVar}(u) \setminus \{x\} & t = \lambda x.u \\ \mathcal{FVar}(u) \cup \mathcal{FVar}(v) & t = u v \end{cases}$$

bound variables

$$\mathcal{BVar}(t) \stackrel{\text{def}}{=} \begin{cases} \emptyset & t = x \\ \{x\} \cup \mathcal{BVar}(u) & t = \lambda x.u \\ \mathcal{BVar}(u) \cup \mathcal{BVar}(v) & t = u v \end{cases}$$

Free and Bound Variables

Definition

free variables

$$\mathcal{FVar}(t) \stackrel{\text{def}}{=} \begin{cases} \{t\} & t = x \\ \mathcal{FVar}(u) \setminus \{x\} & t = \lambda x.u \\ \mathcal{FVar}(u) \cup \mathcal{FVar}(v) & t = u v \end{cases}$$

bound variables

$$\mathcal{BVar}(t) \stackrel{\text{def}}{=} \begin{cases} \emptyset & t = x \\ \{x\} \cup \mathcal{BVar}(u) & t = \lambda x.u \\ \mathcal{BVar}(u) \cup \mathcal{BVar}(v) & t = u v \end{cases}$$

A λ -term without free variables is called **closed**.

Examples

| t | $\mathcal{V}\text{ar}(t)$ | $\mathcal{F}\mathcal{V}\text{ar}(t)$ | $\mathcal{B}\mathcal{V}\text{ar}(t)$ | closed |
|-------------------|---------------------------|--------------------------------------|--------------------------------------|--------|
| $\lambda x.x$ | | | | |
| $x y$ | | | | |
| $(\lambda x.x) x$ | | | | |
| $\lambda x.x y z$ | | | | |

Examples

| t | $\mathcal{V}\text{ar}(t)$ | $\mathcal{F}\mathcal{V}\text{ar}(t)$ | $\mathcal{B}\mathcal{V}\text{ar}(t)$ | closed |
|-------------------|---------------------------|--------------------------------------|--------------------------------------|--------|
| $\lambda x.x$ | $\{x\}$ | | | |
| $x y$ | | | | |
| $(\lambda x.x) x$ | | | | |
| $\lambda x.x y z$ | | | | |

Examples

| t | $\mathcal{V}\text{ar}(t)$ | $\mathcal{F}\mathcal{V}\text{ar}(t)$ | $\mathcal{B}\mathcal{V}\text{ar}(t)$ | closed |
|-------------------|---------------------------|--------------------------------------|--------------------------------------|--------|
| $\lambda x.x$ | $\{x\}$ | \emptyset | | |
| $x y$ | | | | |
| $(\lambda x.x) x$ | | | | |
| $\lambda x.x y z$ | | | | |

Examples

| t | $\mathcal{V}\text{ar}(t)$ | $\mathcal{F}\mathcal{V}\text{ar}(t)$ | $\mathcal{B}\mathcal{V}\text{ar}(t)$ | closed |
|-------------------|---------------------------|--------------------------------------|--------------------------------------|--------|
| $\lambda x.x$ | $\{x\}$ | \emptyset | $\{x\}$ | |
| $x y$ | | | | |
| $(\lambda x.x) x$ | | | | |
| $\lambda x.x y z$ | | | | |

Examples

| t | $\mathcal{V}\text{ar}(t)$ | $\mathcal{F}\mathcal{V}\text{ar}(t)$ | $\mathcal{B}\mathcal{V}\text{ar}(t)$ | closed |
|-------------------|---------------------------|--------------------------------------|--------------------------------------|--------|
| $\lambda x.x$ | $\{x\}$ | \emptyset | $\{x\}$ | ✓ |
| $x y$ | | | | |
| $(\lambda x.x) x$ | | | | |
| $\lambda x.x y z$ | | | | |

Examples

| t | $\mathcal{V}\text{ar}(t)$ | $\mathcal{F}\mathcal{V}\text{ar}(t)$ | $\mathcal{B}\mathcal{V}\text{ar}(t)$ | closed |
|-------------------|---------------------------|--------------------------------------|--------------------------------------|--------|
| $\lambda x.x$ | $\{x\}$ | \emptyset | $\{x\}$ | ✓ |
| $x y$ | $\{x, y\}$ | | | |
| $(\lambda x.x) x$ | | | | |
| $\lambda x.x y z$ | | | | |

Examples

| t | $\mathcal{V}\text{ar}(t)$ | $\mathcal{F}\mathcal{V}\text{ar}(t)$ | $\mathcal{B}\mathcal{V}\text{ar}(t)$ | closed |
|-------------------|---------------------------|--------------------------------------|--------------------------------------|--------|
| $\lambda x.x$ | $\{x\}$ | \emptyset | $\{x\}$ | ✓ |
| $x y$ | $\{x, y\}$ | $\{x, y\}$ | | |
| $(\lambda x.x) x$ | | | | |
| $\lambda x.x y z$ | | | | |

Examples

| t | $\mathcal{V}\text{ar}(t)$ | $\mathcal{F}\mathcal{V}\text{ar}(t)$ | $\mathcal{B}\mathcal{V}\text{ar}(t)$ | closed |
|-------------------|---------------------------|--------------------------------------|--------------------------------------|--------|
| $\lambda x.x$ | $\{x\}$ | \emptyset | $\{x\}$ | ✓ |
| $x y$ | $\{x, y\}$ | $\{x, y\}$ | \emptyset | |
| $(\lambda x.x) x$ | | | | |
| $\lambda x.x y z$ | | | | |

Examples

| t | $\mathcal{V}\text{ar}(t)$ | $\mathcal{F}\mathcal{V}\text{ar}(t)$ | $\mathcal{B}\mathcal{V}\text{ar}(t)$ | closed |
|-------------------|---------------------------|--------------------------------------|--------------------------------------|--------|
| $\lambda x.x$ | $\{x\}$ | \emptyset | $\{x\}$ | ✓ |
| $x y$ | $\{x, y\}$ | $\{x, y\}$ | \emptyset | ✗ |
| $(\lambda x.x) x$ | | | | |
| $\lambda x.x y z$ | | | | |

Examples

| t | $\mathcal{V}\text{ar}(t)$ | $\mathcal{F}\mathcal{V}\text{ar}(t)$ | $\mathcal{B}\mathcal{V}\text{ar}(t)$ | closed |
|-------------------|---------------------------|--------------------------------------|--------------------------------------|--------|
| $\lambda x.x$ | $\{x\}$ | \emptyset | $\{x\}$ | ✓ |
| $x y$ | $\{x, y\}$ | $\{x, y\}$ | \emptyset | ✗ |
| $(\lambda x.x) x$ | $\{x\}$ | | | |
| $\lambda x.x y z$ | | | | |

Examples

| t | $\mathcal{V}\text{ar}(t)$ | $\mathcal{F}\mathcal{V}\text{ar}(t)$ | $\mathcal{B}\mathcal{V}\text{ar}(t)$ | closed |
|-------------------|---------------------------|--------------------------------------|--------------------------------------|--------|
| $\lambda x.x$ | $\{x\}$ | \emptyset | $\{x\}$ | ✓ |
| $x y$ | $\{x, y\}$ | $\{x, y\}$ | \emptyset | ✗ |
| $(\lambda x.x) x$ | $\{x\}$ | $\{x\}$ | | |
| $\lambda x.x y z$ | | | | |

Examples

| t | $\mathcal{V}\text{ar}(t)$ | $\mathcal{F}\mathcal{V}\text{ar}(t)$ | $\mathcal{B}\mathcal{V}\text{ar}(t)$ | closed |
|-------------------|---------------------------|--------------------------------------|--------------------------------------|--------|
| $\lambda x.x$ | $\{x\}$ | \emptyset | $\{x\}$ | ✓ |
| $x y$ | $\{x, y\}$ | $\{x, y\}$ | \emptyset | ✗ |
| $(\lambda x.x) x$ | $\{x\}$ | $\{x\}$ | $\{x\}$ | |
| $\lambda x.x y z$ | | | | |

Examples

| t | $\mathcal{V}\text{ar}(t)$ | $\mathcal{F}\mathcal{V}\text{ar}(t)$ | $\mathcal{B}\mathcal{V}\text{ar}(t)$ | closed |
|-------------------|---------------------------|--------------------------------------|--------------------------------------|--------|
| $\lambda x.x$ | $\{x\}$ | \emptyset | $\{x\}$ | ✓ |
| $x y$ | $\{x, y\}$ | $\{x, y\}$ | \emptyset | ✗ |
| $(\lambda x.x) x$ | $\{x\}$ | $\{x\}$ | $\{x\}$ | ✗ |
| $\lambda x.x y z$ | | | | |

Examples

| t | $\mathcal{V}\text{ar}(t)$ | $\mathcal{F}\mathcal{V}\text{ar}(t)$ | $\mathcal{B}\mathcal{V}\text{ar}(t)$ | closed |
|-------------------|---------------------------|--------------------------------------|--------------------------------------|--------|
| $\lambda x.x$ | $\{x\}$ | \emptyset | $\{x\}$ | ✓ |
| $x y$ | $\{x, y\}$ | $\{x, y\}$ | \emptyset | ✗ |
| $(\lambda x.x) x$ | $\{x\}$ | $\{x\}$ | $\{x\}$ | ✗ |
| $\lambda x.x y z$ | $\{x, y, z\}$ | | | |

Examples

| t | $\mathcal{V}\text{ar}(t)$ | $\mathcal{F}\mathcal{V}\text{ar}(t)$ | $\mathcal{B}\mathcal{V}\text{ar}(t)$ | closed |
|-------------------|---------------------------|--------------------------------------|--------------------------------------|--------|
| $\lambda x.x$ | $\{x\}$ | \emptyset | $\{x\}$ | ✓ |
| $x y$ | $\{x, y\}$ | $\{x, y\}$ | \emptyset | ✗ |
| $(\lambda x.x) x$ | $\{x\}$ | $\{x\}$ | $\{x\}$ | ✗ |
| $\lambda x.x y z$ | $\{x, y, z\}$ | $\{y, z\}$ | | |

Examples

| t | $\mathcal{V}\text{ar}(t)$ | $\mathcal{F}\mathcal{V}\text{ar}(t)$ | $\mathcal{B}\mathcal{V}\text{ar}(t)$ | closed |
|-------------------|---------------------------|--------------------------------------|--------------------------------------|--------|
| $\lambda x.x$ | $\{x\}$ | \emptyset | $\{x\}$ | ✓ |
| $x y$ | $\{x, y\}$ | $\{x, y\}$ | \emptyset | ✗ |
| $(\lambda x.x) x$ | $\{x\}$ | $\{x\}$ | $\{x\}$ | ✗ |
| $\lambda x.x y z$ | $\{x, y, z\}$ | $\{y, z\}$ | $\{x\}$ | |

Examples

| t | $\mathcal{V}\text{ar}(t)$ | $\mathcal{F}\mathcal{V}\text{ar}(t)$ | $\mathcal{B}\mathcal{V}\text{ar}(t)$ | closed |
|-------------------|---------------------------|--------------------------------------|--------------------------------------|--------|
| $\lambda x.x$ | $\{x\}$ | \emptyset | $\{x\}$ | ✓ |
| $x y$ | $\{x, y\}$ | $\{x, y\}$ | \emptyset | ✗ |
| $(\lambda x.x) x$ | $\{x\}$ | $\{x\}$ | $\{x\}$ | ✗ |
| $\lambda x.x y z$ | $\{x, y, z\}$ | $\{y, z\}$ | $\{x\}$ | ✗ |

Computations

Idea

- rules to manipulate λ -terms
- a single rule is enough

Computations

Idea

- rules to manipulate λ -terms
- a single rule is enough

The β -rule (informal)

$$(\lambda x.s) t \rightarrow_{\beta} s\{x/t\}$$

application of a function to some input

Computations

Idea

- rules to manipulate λ -terms
- a single rule is enough

The β -rule (informal)

$$(\lambda x.s) t \rightarrow_{\beta} \underbrace{s\{x/t\}}$$

substitute x by t in s

application of a function to some input

Blindly replacing does not suffice

Example

Blindly replacing does not suffice

Example

- consider $\lambda xy.x$ (**fun** **x** **y** **->** **x** in OCaml)

Blindly replacing does not suffice

Example

- consider $\lambda xy.x$ (**fun** **x** **y** **->** **x** in OCaml)
- behavior: *“take 2 arguments, ignore second, return first”*

Blindly replacing does not suffice

Example

- consider $\lambda xy.x$ (**fun** **x** **y** **->** **x** in OCaml)
- behavior: *“take 2 arguments, ignore second, return first”*
- $(\lambda xy.x) v w$

Blindly replacing does not suffice

Example

- consider $\lambda xy.x$ (**fun** **x** **y** **->** **x** in OCaml)
- behavior: *“take 2 arguments, ignore second, return first”*
- $(\lambda xy.x) v w \rightsquigarrow (\lambda y.v) w$

Blindly replacing does not suffice

Example

- consider $\lambda xy.x$ (`fun x y -> x` in OCaml)
- behavior: “take 2 arguments, ignore second, return first”
- $(\lambda xy.x) v w \rightsquigarrow (\lambda y.v) w \rightsquigarrow v$

Blindly replacing does not suffice

Example

- consider $\lambda xy.x$ (**fun** **x** **y** **->** **x** in OCaml)
- behavior: *“take 2 arguments, ignore second, return first”*
- $(\lambda xy.x) v w \rightsquigarrow (\lambda y.v) w \rightsquigarrow v$ ✓

Blindly replacing does not suffice

Example

- consider $\lambda xy.x$ (`fun x y -> x` in OCaml)
- behavior: “take 2 arguments, ignore second, return first”
- $(\lambda xy.x) v w \rightsquigarrow (\lambda y.v) w \rightsquigarrow v$ ✓
- $(\lambda xy.x) y z$

Blindly replacing does not suffice

Example

- consider $\lambda xy.x$ (`fun x y -> x` in OCaml)
- behavior: “take 2 arguments, ignore second, return first”
- $(\lambda xy.x) v w \rightsquigarrow (\lambda y.v) w \rightsquigarrow v$ ✓
- $(\lambda xy.x) y z \rightsquigarrow (\lambda y.y) z$

Blindly replacing does not suffice

Example

- consider $\lambda xy.x$ (`fun x y -> x` in OCaml)
- behavior: “take 2 arguments, ignore second, return first”
- $(\lambda xy.x) v w \rightsquigarrow (\lambda y.v) w \rightsquigarrow v$ ✓
- $(\lambda xy.x) y z \rightsquigarrow (\lambda y.y) z \rightsquigarrow z$

Blindly replacing does not suffice

Example

- consider $\lambda xy.x$ (`fun x y -> x` in OCaml)
- behavior: “take 2 arguments, ignore second, return first”
- $(\lambda xy.x) v w \rightsquigarrow (\lambda y.v) w \rightsquigarrow v$ ✓
- $(\lambda xy.x) y z \rightsquigarrow (\lambda y.y) z \rightsquigarrow z$ ✗

Blindly replacing does not suffice

Example

- consider $\lambda xy.x$ (`fun x y -> x` in OCaml)
- behavior: “take 2 arguments, ignore second, return first”
- $(\lambda xy.x) v w \rightsquigarrow (\lambda y.v) w \rightsquigarrow v$ ✓
- $(\lambda xy.x) y z \rightsquigarrow (\lambda y.y) z \rightsquigarrow z$ ✗
- clearly not intended

Blindly replacing does not suffice

Example

- consider $\lambda xy.x$ (**fun** **x** **y** **->** **x** in OCaml)
- behavior: “take 2 arguments, ignore second, return first”
- $(\lambda xy.x) v w \rightsquigarrow (\lambda y.v) w \rightsquigarrow v$ ✓
- $(\lambda xy.x) y z \rightsquigarrow (\lambda y.y) z \rightsquigarrow z$ ✗
- clearly not intended (Problem: **variable capture**)

Blindly replacing does not suffice

Example

- consider $\lambda xy.x$ (**fun** **x** **y** **->** **x** in OCaml)
- behavior: “take 2 arguments, ignore second, return first”
- $(\lambda xy.x) v w \rightsquigarrow (\lambda y.v) w \rightsquigarrow v$ ✓
- $(\lambda xy.x) y z \rightsquigarrow (\lambda y.y) z \rightsquigarrow z$ ✗
- clearly not intended (Problem: **variable capture**)
- $(\lambda xy.x) y z$

Blindly replacing does not suffice

Example

- consider $\lambda xy.x$ (**fun** **x** **y** **->** **x** in OCaml)
- behavior: “take 2 arguments, ignore second, return first”
- $(\lambda xy.x) v w \rightsquigarrow (\lambda y.v) w \rightsquigarrow v$ ✓
- $(\lambda xy.x) y z \rightsquigarrow (\lambda y.y) z \rightsquigarrow z$ ✗
- clearly not intended (Problem: **variable capture**)
- $(\lambda xy.x) y z \rightarrow_{\beta} (\lambda y'.y) z$

Solution

rename bound variables where necessary

Blindly replacing does not suffice

Example

- consider $\lambda xy.x$ (**fun** **x** **y** **->** **x** in OCaml)
- behavior: “take 2 arguments, ignore second, return first”
- $(\lambda xy.x) v w \rightsquigarrow (\lambda y.v) w \rightsquigarrow v$ ✓
- $(\lambda xy.x) y z \rightsquigarrow (\lambda y.y) z \rightsquigarrow z$ ✗
- clearly not intended (Problem: **variable capture**)
- $(\lambda xy.x) y z \rightarrow_{\beta} (\lambda y'.y) z \rightarrow_{\beta} y$

Solution

rename bound variables where necessary

Blindly replacing does not suffice

Example

- consider $\lambda xy.x$ (`fun x y -> x` in OCaml)
- behavior: “take 2 arguments, ignore second, return first”
- $(\lambda xy.x) v w \rightsquigarrow (\lambda y.v) w \rightsquigarrow v$ ✓
- $(\lambda xy.x) y z \rightsquigarrow (\lambda y.y) z \rightsquigarrow z$ ✗
- clearly not intended (Problem: **variable capture**)
- $(\lambda xy.x) y z \rightarrow_{\beta} (\lambda y'.y) z \rightarrow_{\beta} y$

Solution

rename bound variables where necessary

OCaml

```
let y = 3 and z = 2;;
(fun x -> (fun y -> x)) y z;;
```

Blindly replacing does not suffice

Example

- consider $\lambda xy.x$ (`fun x y -> x` in OCaml)
- behavior: “take 2 arguments, ignore second, return first”
- $(\lambda xy.x) v w \rightsquigarrow (\lambda y.v) w \rightsquigarrow v$ ✓
- $(\lambda xy.x) y z \rightsquigarrow (\lambda y.y) z \rightsquigarrow z$ ✗
- clearly not intended (Problem: **variable capture**)
- $(\lambda xy.x) y z \rightarrow_{\beta} (\lambda y'.y) z \rightarrow_{\beta} y$

Solution

rename bound variables where necessary

OCaml

```
let y = 3 and z = 2;;
(fun u -> (fun v -> u)) y z;;
```

Substitutions

Definition

function from variables to terms

$$\sigma: \mathcal{V} \rightarrow \mathcal{T}(\mathcal{V})$$

in our case we only need substitutions replacing a single variable, i.e., only for one $x \in \mathcal{V}$, $\sigma(x) \neq x$

Substitutions

Definition

function from variables to terms

$$\sigma: \mathcal{V} \rightarrow \mathcal{T}(\mathcal{V})$$

in our case we only need substitutions replacing a single variable, i.e., only for one $x \in \mathcal{V}$, $\sigma(x) \neq x$

Notation

binding for x such that $\sigma(x) \neq x$

$$\sigma = \{x/t\}$$

Substitutions

Definition

function from variables to terms

$$\sigma: \mathcal{V} \rightarrow \mathcal{T}(\mathcal{V})$$

in our case we only need substitutions replacing a single variable, i.e., only for one $x \in \mathcal{V}$, $\sigma(x) \neq x$

Notation

binding for x such that $\sigma(x) \neq x$

$$\sigma = \{x/t\}$$

Example

$$\sigma = \{x/\lambda x.x\} \text{ hence } \sigma(x) = \lambda x.x \text{ and } \sigma(y) = y$$

Substitutions (cont'd)

Definition (Application)

apply substitution $\sigma = \{x/s\}$ to term t

$$t\sigma \stackrel{\text{def}}{=} \begin{cases} s & t = x \\ y & t = y, x \neq y \\ (u\sigma) (v\sigma) & t = u v \\ \lambda x.u & t = \lambda x.u \\ \lambda y.(u\sigma) & t = \lambda y.u, x \neq y, y \notin \mathcal{FVar}(s) \\ \lambda y'.((u\{y/y'\})\sigma) & t = \lambda y.u, x \neq y, y \in \mathcal{FVar}(s), y' \text{ fresh} \end{cases}$$

Substitutions (cont'd)

Definition (Application)

apply substitution $\sigma = \{x/s\}$ to term t

$$t\sigma \stackrel{\text{def}}{=} \begin{cases} s & t = x \\ y & t = y, x \neq y \\ (u\sigma) (v\sigma) & t = u v \\ \lambda x. u & t = \lambda x. u \\ \lambda y. (u\sigma) & t = \lambda y. u, x \neq y, y \notin \mathcal{FV}\text{ar}(s) \\ \lambda y'. ((u\{y/y'\})\sigma) & t = \lambda y. u, x \neq y, y \in \mathcal{FV}\text{ar}(s), y' \text{ fresh} \end{cases}$$

Example ($\sigma = \{x/\lambda v. v w\}$)

$$x\sigma =$$

$$y\sigma =$$

$$(x y)\sigma =$$

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

$$(\lambda v. x w)\sigma =$$

$$(\lambda w. x w)\sigma =$$

Substitutions (cont'd)

Definition (Application)

apply substitution $\sigma = \{x/s\}$ to term t

$$t\sigma \stackrel{\text{def}}{=} \begin{cases} s & t = x \\ y & t = y, x \neq y \\ (u\sigma) (v\sigma) & t = u v \\ \lambda x.u & t = \lambda x.u \\ \lambda y.(u\sigma) & t = \lambda y.u, x \neq y, y \notin \mathcal{FV}\text{ar}(s) \\ \lambda y'.((u\{y/y'\})\sigma) & t = \lambda y.u, x \neq y, y \in \mathcal{FV}\text{ar}(s), y' \text{ fresh} \end{cases}$$

Example ($\sigma = \{x/\lambda v.v w\}$)

$$x\sigma = \lambda v.v w$$

$$y\sigma =$$

$$(x y)\sigma =$$

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

$$(\lambda v.x w)\sigma =$$

$$(\lambda w.x w)\sigma =$$

Substitutions (cont'd)

Definition (Application)

apply substitution $\sigma = \{x/s\}$ to term t

$$t\sigma \stackrel{\text{def}}{=} \begin{cases} s & t = x \\ y & t = y, x \neq y \\ (u\sigma) (v\sigma) & t = u v \\ \lambda x. u & t = \lambda x. u \\ \lambda y. (u\sigma) & t = \lambda y. u, x \neq y, y \notin \mathcal{FV}\text{ar}(s) \\ \lambda y'. ((u\{y/y'\})\sigma) & t = \lambda y. u, x \neq y, y \in \mathcal{FV}\text{ar}(s), y' \text{ fresh} \end{cases}$$

Example ($\sigma = \{x/\lambda v. v w\}$)

$$x\sigma = \lambda v. v w$$

$$y\sigma = y$$

$$(x y)\sigma =$$

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

$$(\lambda v. x w)\sigma =$$

$$(\lambda w. x w)\sigma =$$

Substitutions (cont'd)

Definition (Application)

apply substitution $\sigma = \{x/s\}$ to term t

$$t\sigma \stackrel{\text{def}}{=} \begin{cases} s & t = x \\ y & t = y, x \neq y \\ (u\sigma) (v\sigma) & t = u v \\ \lambda x.u & t = \lambda x.u \\ \lambda y.(u\sigma) & t = \lambda y.u, x \neq y, y \notin \mathcal{FV}\text{ar}(s) \\ \lambda y'.((u\{y/y'\})\sigma) & t = \lambda y.u, x \neq y, y \in \mathcal{FV}\text{ar}(s), y' \text{ fresh} \end{cases}$$

Example ($\sigma = \{x/\lambda v.v w\}$)

$$x\sigma = \lambda v.v w$$

$$y\sigma = y$$

$$(x y)\sigma = (\lambda v.v w) y$$

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

$$(\lambda v.x w)\sigma =$$

$$(\lambda w.x w)\sigma =$$

Substitutions (cont'd)

Definition (Application)

apply substitution $\sigma = \{x/s\}$ to term t

$$t\sigma \stackrel{\text{def}}{=} \begin{cases} s & t = x \\ y & t = y, x \neq y \\ (u\sigma) (v\sigma) & t = u v \\ \lambda x.u & t = \lambda x.u \\ \lambda y.(u\sigma) & t = \lambda y.u, x \neq y, y \notin \mathcal{FVar}(s) \\ \lambda y'.((u\{y/y'\})\sigma) & t = \lambda y.u, x \neq y, y \in \mathcal{FVar}(s), y' \text{ fresh} \end{cases}$$

Example ($\sigma = \{x/\lambda v.v w\}$)

$$x\sigma = \lambda v.v w$$

$$y\sigma = y$$

$$(x y)\sigma = (\lambda v.v w) y$$

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

$$(\lambda v.x w)\sigma =$$

$$(\lambda w.x w)\sigma =$$

Substitutions (cont'd)

Definition (Application)

apply substitution $\sigma = \{x/s\}$ to term t

$$t\sigma \stackrel{\text{def}}{=} \begin{cases} s & t = x \\ y & t = y, x \neq y \\ (u\sigma) (v\sigma) & t = u v \\ \lambda x.u & t = \lambda x.u \\ \lambda y.(u\sigma) & t = \lambda y.u, x \neq y, y \notin \mathcal{FVar}(s) \\ \lambda y'.((u\{y/y'\})\sigma) & t = \lambda y.u, x \neq y, y \in \mathcal{FVar}(s), y' \text{ fresh} \end{cases}$$

Example ($\sigma = \{x/\lambda v.v w\}$)

$$x\sigma = \lambda v.v w$$

$$y\sigma = y$$

$$(x y)\sigma = (\lambda v.v w) y$$

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

$$(\lambda v.x w)\sigma = \lambda v.(\lambda v.v w) w$$

$$(\lambda w.x w)\sigma =$$

Substitutions (cont'd)

Definition (Application)

apply substitution $\sigma = \{x/s\}$ to term t

$$t\sigma \stackrel{\text{def}}{=} \begin{cases} s & t = x \\ y & t = y, x \neq y \\ (u\sigma) (v\sigma) & t = u v \\ \lambda x.u & t = \lambda x.u \\ \lambda y.(u\sigma) & t = \lambda y.u, x \neq y, y \notin \mathcal{FVar}(s) \\ \lambda y'.((u\{y/y'\})\sigma) & t = \lambda y.u, x \neq y, y \in \mathcal{FVar}(s), y' \text{ fresh} \end{cases}$$

Example ($\sigma = \{x/\lambda v.v w\}$)

$$x\sigma = \lambda v.v w$$

$$y\sigma = y$$

$$(x y)\sigma = (\lambda v.v w) y$$

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

$$(\lambda v.x w)\sigma = \lambda v.(\lambda v.v w) w$$

$$(\lambda w.x w)\sigma = \lambda w'.(\lambda v.v w) w'$$

Examples

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

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

$$(\lambda xyz.x z (y z)) (\lambda x.x) \rightarrow_{\beta}$$

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

$$\lambda x.x \rightarrow_{\beta}$$

$$\lambda x.\underline{(\lambda y.y)} z \rightarrow_{\beta}$$

Examples

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

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

$$(\lambda xyz.x z (y z)) (\lambda x.x) \rightarrow_{\beta}$$

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

$$\lambda x.x \rightarrow_{\beta}$$

$$\lambda x.\underline{(\lambda y.y)} z \rightarrow_{\beta}$$

Examples

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

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

$$(\lambda xyz.x z (y z)) (\lambda x.x) \rightarrow_{\beta}$$

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

$$\lambda x.x \rightarrow_{\beta}$$

$$\lambda x.\underline{(\lambda y.y)} z \rightarrow_{\beta}$$

Examples

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

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

$$(\lambda xyz.x z (y z)) (\lambda x.x) \rightarrow_{\beta} \lambda yz.(\lambda x.x) z (y z)$$

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

$$\lambda x.x \rightarrow_{\beta}$$

$$\lambda x.\underline{(\lambda y.y) z} \rightarrow_{\beta}$$

Examples

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

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

$$(\lambda xyz.x z (y z)) (\lambda x.x) \rightarrow_{\beta} \lambda yz.(\lambda x.x) z (y z)$$

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

$$\lambda x.x \rightarrow_{\beta}$$

$$\lambda x.\underline{(\lambda y.y) z} \rightarrow_{\beta}$$

Examples

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

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

$$(\lambda xyz.x z (y z)) (\lambda x.x) \rightarrow_{\beta} \lambda yz.(\lambda x.x) z (y z)$$

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

$$\lambda x.x \rightarrow_{\beta} \text{no } \beta\text{-step possible}$$

$$\lambda x.\underline{(\lambda y.y)} z \rightarrow_{\beta}$$

Examples

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

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

$$(\lambda xyz.x z (y z)) (\lambda x.x) \rightarrow_{\beta} \lambda yz.(\lambda x.x) z (y z)$$

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

$$\lambda x.x \rightarrow_{\beta} \text{no } \beta\text{-step possible}$$

$$\lambda x. \underline{(\lambda y.y) z} \rightarrow_{\beta} \lambda x. \underline{z}$$

β -Reduction

Definition (Context)

context $C \in \mathcal{C}(\mathcal{V})$

$$C ::= \square \mid \lambda x.C \mid C t \mid t C$$

with $\square \notin \mathcal{V}$, $x \in \mathcal{V}$ and $t \in \mathcal{T}(\mathcal{V})$

- $C[s]$ denotes replacing \square by term s in context C

β -Reduction

Definition (Context)

context $C \in \mathcal{C}(\mathcal{V})$

$$C ::= \square \mid \lambda x. C \mid C t \mid t C$$

with $\square \notin \mathcal{V}$, $x \in \mathcal{V}$ and $t \in \mathcal{T}(\mathcal{V})$

- $C[s]$ denotes replacing \square by term s in context C

Example

$$C_1 = \square$$

$$C_2 = x \square$$

$$C_3 = \lambda x. \square x$$

$$C_1[\lambda x. x] =$$

$$C_2[\lambda x. x] =$$

$$C_3[\lambda x. x] =$$

β -Reduction

Definition (Context)

context $C \in \mathcal{C}(\mathcal{V})$

$$C ::= \square \mid \lambda x. C \mid C t \mid t C$$

with $\square \notin \mathcal{V}$, $x \in \mathcal{V}$ and $t \in \mathcal{T}(\mathcal{V})$

- $C[s]$ denotes replacing \square by term s in context C

Example

$$C_1 = \square$$

$$C_1[\lambda x. x] = \lambda x. x$$

$$C_2 = x \square$$

$$C_2[\lambda x. x] =$$

$$C_3 = \lambda x. \square x$$

$$C_3[\lambda x. x] =$$

β -Reduction

Definition (Context)

context $C \in \mathcal{C}(\mathcal{V})$

$$C ::= \square \mid \lambda x.C \mid C t \mid t C$$

with $\square \notin \mathcal{V}$, $x \in \mathcal{V}$ and $t \in \mathcal{T}(\mathcal{V})$

- $C[s]$ denotes replacing \square by term s in context C

Example

$$C_1 = \square$$

$$C_1[\lambda x.x] = \lambda x.x$$

$$C_2 = x \square$$

$$C_2[\lambda x.x] = x (\lambda x.x)$$

$$C_3 = \lambda x.\square x$$

$$C_3[\lambda x.x] =$$

β -Reduction

Definition (Context)

context $C \in \mathcal{C}(\mathcal{V})$

$$C ::= \square \mid \lambda x.C \mid C t \mid t C$$

with $\square \notin \mathcal{V}$, $x \in \mathcal{V}$ and $t \in \mathcal{T}(\mathcal{V})$

- $C[s]$ denotes replacing \square by term s in context C

Example

$$C_1 = \square$$

$$C_1[\lambda x.x] = \lambda x.x$$

$$C_2 = x \square$$

$$C_2[\lambda x.x] = x (\lambda x.x)$$

$$C_3 = \lambda x.\square x$$

$$C_3[\lambda x.x] = \lambda x.(\lambda x.x) x$$

β -Reduction (cont'd)

Definition (β -step)

if exist context C and terms s , u , and v such that

$$s = C[(\lambda x.u) v]$$

then

$$s \rightarrow_{\beta} C[u\{x/v\}]$$

is a β -step with redex $(\lambda x.u) v$ and contractum $u\{x/v\}$

β -Reduction (cont'd)

Definition (β -step)

if exist context C and terms s , u , and v such that

$$s = C[(\lambda x.u) v]$$

then

$$s \rightarrow_{\beta} C[u\{x/v\}]$$

is a β -step with redex $(\lambda x.u) v$ and contractum $u\{x/v\}$

β -Reduction (cont'd)

Definition (β -step)

if exist context C and terms s , u , and v such that

$$s = C[(\lambda x.u) v]$$

then

$$s \rightarrow_{\beta} C[u\{x/v\}]$$

is a β -step with **redex** $(\lambda x.u) v$ and **contractum** $u\{x/v\}$

β -Reduction (cont'd)

Definition (β -step)

if exist context C and terms s , u , and v such that

$$s = C[(\lambda x.u) v]$$

then

$$s \rightarrow_{\beta} C[u\{x/v\}]$$

is a β -step with redex $(\lambda x.u) v$ and **contractum** $u\{x/v\}$

β -Reduction (cont'd)

Definition (β -step)

if exist context C and terms s , u , and v such that

$$s = C[(\lambda x.u) v]$$

then

$$s \rightarrow_{\beta} C[u\{x/v\}]$$

is a β -step with redex $(\lambda x.u) v$ and contractum $u\{x/v\}$

- $s \rightarrow_{\beta}^+ t$ denotes sequence $s = t_1 \rightarrow_{\beta} t_2 \rightarrow_{\beta} \cdots \rightarrow_{\beta} t_n = t$ with $n > 0$

β -Reduction (cont'd)

Definition (β -step)

if exist context C and terms s , u , and v such that

$$s = C[(\lambda x.u) v]$$

then

$$s \rightarrow_{\beta} C[u\{x/v\}]$$

is a β -step with redex $(\lambda x.u) v$ and contractum $u\{x/v\}$

- $s \rightarrow_{\beta}^+ t$ denotes sequence $s = t_1 \rightarrow_{\beta} t_2 \rightarrow_{\beta} \cdots \rightarrow_{\beta} t_n = t$ with $n > 0$
- $s \rightarrow_{\beta}^* t$ is sequence with $n \geq 0$ (s β -reduces to t)

β -Reduction

Example

$$\Omega = (\lambda x.x x) (\lambda x.x x)$$

$$K_* = \lambda xy.y$$

$$I_2 = \lambda xy.x y$$

$$K_* \Omega$$

$$K_* \Omega$$

$$I_2 I_2$$

β -Reduction

Example

$$\Omega = (\lambda x.x x) (\lambda x.x x)$$

$$K_* = \lambda xy.y$$

$$I_2 = \lambda xy.x y$$

$$K_* \Omega \rightarrow_{\beta} K_* \Omega$$

$$K_* \Omega$$

$$I_2 I_2$$

β -Reduction

Example

$$\Omega = (\lambda x.x x) (\lambda x.x x)$$

$$K_* = \lambda xy.y$$

$$I_2 = \lambda xy.x y$$

$$K_* \Omega \rightarrow_{\beta} K_* \Omega \rightarrow_{\beta} \dots$$

$$K_* \Omega$$

$$I_2 I_2$$

β -Reduction

Example

$$\Omega = (\lambda x.x x) (\lambda x.x x)$$

$$K_* = \lambda xy.y$$

$$I_2 = \lambda xy.x y$$

$$K_* \Omega \rightarrow_{\beta} K_* \Omega \rightarrow_{\beta} \dots$$

$$K_* \Omega \rightarrow_{\beta} \lambda y.y$$

$$I_2 I_2$$

β -Reduction

Example

$$\Omega = (\lambda x.x x) (\lambda x.x x)$$

$$K_* = \lambda xy.y$$

$$I_2 = \lambda xy.x y$$

$$K_* \Omega \rightarrow_{\beta} K_* \Omega \rightarrow_{\beta} \dots$$

$$K_* \Omega \rightarrow_{\beta} \lambda y.y$$

$$I_2 I_2 = (\lambda xy.x y) (\lambda xy.x y)$$

β -Reduction

Example

$$\Omega = (\lambda x.x x) (\lambda x.x x)$$

$$K_* = \lambda xy.y$$

$$I_2 = \lambda xy.x y$$

$$K_* \Omega \rightarrow_{\beta} K_* \Omega \rightarrow_{\beta} \dots$$

$$K_* \Omega \rightarrow_{\beta} \lambda y.y$$

$$I_2 I_2 = (\lambda xy.x y) (\lambda xy.x y) \rightarrow_{\beta} \lambda y.(\lambda xy.x y) y$$

β -Reduction

Example

$$\Omega = (\lambda x.x x) (\lambda x.x x)$$

$$K_* = \lambda xy.y$$

$$I_2 = \lambda xy.x y$$

$$K_* \Omega \rightarrow_{\beta} K_* \Omega \rightarrow_{\beta} \dots$$

$$K_* \Omega \rightarrow_{\beta} \lambda y.y$$

$$I_2 I_2 = (\lambda xy.x y) (\lambda xy.x y) \rightarrow_{\beta} \lambda y.(\lambda xy.x y) y \equiv \lambda y.(\lambda xy'.x y') y$$

β -Reduction

Example

$$\Omega = (\lambda x.x x) (\lambda x.x x)$$

$$K_* = \lambda xy.y$$

$$I_2 = \lambda xy.x y$$

$$K_* \Omega \rightarrow_{\beta} K_* \Omega \rightarrow_{\beta} \dots$$

$$K_* \Omega \rightarrow_{\beta} \lambda y.y$$

$$\begin{aligned} I_2 I_2 &= (\lambda xy.x y) (\lambda xy.x y) \rightarrow_{\beta} \lambda y.(\lambda xy.x y) y \equiv \lambda y.(\lambda xy'.x y') y \\ &\rightarrow_{\beta} \lambda y.(\lambda y'.y y') \end{aligned}$$

β -Reduction

Example

$$\Omega = (\lambda x.x x) (\lambda x.x x)$$

$$K_* = \lambda xy.y$$

$$I_2 = \lambda xy.x y$$

$$K_* \Omega \rightarrow_{\beta} K_* \Omega \rightarrow_{\beta} \dots$$

$$K_* \Omega \rightarrow_{\beta} \lambda y.y$$

$$\begin{aligned} I_2 I_2 &= (\lambda xy.x y) (\lambda xy.x y) \rightarrow_{\beta} \lambda y.(\lambda xy.x y) y \equiv \lambda y.(\lambda xy'.x y') y \\ &\rightarrow_{\beta} \lambda y.(\lambda y'.y y') = \lambda yy'.y y' \end{aligned}$$

β -Reduction

Example

$$\Omega = (\lambda x.x x) (\lambda x.x x)$$

$$K_* = \lambda xy.y$$

$$I_2 = \lambda xy.x y$$

$$K_* \Omega \rightarrow_{\beta} K_* \Omega \rightarrow_{\beta} \dots$$

$$K_* \Omega \rightarrow_{\beta} \lambda y.y$$

$$\begin{aligned} I_2 I_2 &= (\lambda xy.x y) (\lambda xy.x y) \rightarrow_{\beta} \lambda y.(\lambda xy.x y) y \equiv \lambda y.(\lambda xy'.x y') y \\ &\rightarrow_{\beta} \lambda y.(\lambda y'.y y') = \lambda yy'.y y' \equiv I_2 \end{aligned}$$

What Are the Results of Computations?

Idea

- only **terms** in λ -calculus
- express functions and values through λ -terms

What Are the Results of Computations?

Idea

- only terms in λ -calculus
- express functions **and** values through λ -terms

What Are the Results of Computations?

Idea

- only terms in λ -calculus
- express functions and values through λ -terms

Definition (Normal form)

$t \in \mathcal{T}(\mathcal{V})$ is in normal form (NF) if no β -step possible

What Are the Results of Computations?

Idea

- only terms in λ -calculus
- express functions and values through λ -terms

Definition (Normal form)

$t \in \mathcal{T}(\mathcal{V})$ is in **normal form** (NF) if no β -step possible

What Are the Results of Computations?

Idea

- only terms in λ -calculus
- express functions and values through λ -terms

Definition (Normal form)

$t \in \mathcal{T}(\mathcal{V})$ is in normal form (NF) if no β -step possible

Example

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

What Are the Results of Computations?

Idea

- only terms in λ -calculus
- express functions and values through λ -terms

Definition (Normal form)

$t \in \mathcal{T}(\mathcal{V})$ is in normal form (NF) if no β -step possible

Example

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

What Are the Results of Computations?

Idea

- only terms in λ -calculus
- express functions and values through λ -terms

Definition (Normal form)

$t \in \mathcal{T}(\mathcal{V})$ is in normal form (NF) if no β -step possible

Example

| | |
|-------------------|--------|
| $\lambda x.x$ | NF |
| $(\lambda x.x) y$ | not NF |

Overview

- Week 5 - λ -Calculus
 - Summary of Week 4
 - λ -Calculus - Introduction
 - λ -Calculus - Formalities
 - The λ Interpreter `lips`



Lambda Interpreter for Pure Students

developed by Michael Brunner ([bachelor thesis](#))

Lambda Interpreter for Pure Students

developed by Michael Brunner ([bachelor thesis](#))

λ -Terms

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

Lambda Interpreter for Pure Students

developed by Michael Brunner ([bachelor thesis](#))

λ -Terms

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

Conventions

- interpreter command `!pretty` toggles use of conventions for printing
- nested abstractions use spaces to separate variable names, e.g.,

$$\begin{array}{ll} \lambda xy.x & \lambda x y.x \\ \lambda x_1.y & \lambda x1.y \end{array}$$

Result

Normal Forms

- result of input is corresponding NF
 - $> (\lambda x.x) (\lambda x.x)$
NF: $(\lambda x.x)$

Result

Normal Forms

- result of input is corresponding NF
 - $> (\lambda x.x) (\lambda x.x)$
NF: $(\lambda x.x)$

Evaluation Strategy

- `!by_value` activates call-by-value evaluation (next lecture)
- `!by_name` activates call-by-name evaluation (next lecture)
- `!trace` toggles tracing

Abbreviations & Initialisation

Interpreter Command

```
!def  $\langle name \rangle = t$ 
```

Abbreviations & Initialisation

Interpreter Command

```
!def <name> = t
```

Example

```
> !def I = \x.x
> !def K = \x y.x
> !def S = \x y z.x z (y z)
> S K I
NF: \z.z
```

Abbreviations & Initialisation

Interpreter Command

```
!def <name> = t
```

Example

```
> !def I = \x.x
> !def K = \x y.x
> !def S = \x y z.x z (y z)
> S K I
NF: \z.z
```

`.lambdainit`

content of file `.lambdainit` is loaded on start-up of `lips`