universität innsbruck

Interactive Theorem Proving

Lecture & Exercises Week 3

Cezary Kaliszyk

Summary

Previous Lecture

- LCF style
- HOL provers family
- HOL logic

Today

- HOL Kernel and Exercises
- (untyped) λ-calculus
- λ-calculus vs functional programming

Exercises

- Show symmetry of equality (using the HOL inference rules), namely show $A = B \vdash B = A$.
- How would you implement an LCF system that corresponds to some minimal basic propositional logic? What would be the types? Terms? Are there theorems and what would the rules to construct them be?
- Figure out how to run HOL Light
- Bonus: How would you show the S combinator $(A \to B \to C) \to (A \to B) \to A \to C$ with the basic HOL inference rules? (On paper or in a HOL system).

Guide to reading the HOL Light source

- `hol.ml`: load order
- `lib.ml`: ML standard library for portability
- `fusion.ml`: the kernel
- `drule.ml`: simple derived rules
- `bool.ml`: basic boolean constants
- `tactic.ml`: subgoal package
- `simp.ml`: rewriting

Lambda Calculus: Origin

Goal

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

Lambda Calculus: Origin

Goal

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

Lambda Calculus: Origin

Goal

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

Result

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

Lambda Calculus: 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

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

$\mathcal{T}(\mathcal{V})$ 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

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

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

Syntax

λ-Terms

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

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

Syntax

λ-Terms

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

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

Syntax

λ-Terms

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

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

Conventions

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

Syntax

λ-Terms

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

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

Conventions (omit outermost parentheses)

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

Syntax

λ-Terms

$$t ::= \quad x \quad | \; (\lambda x.t) \; | \; (t \; t)$$

$\mathcal{T}(\mathcal{V})$ 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

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

$\mathcal{T}(\mathcal{V})$ 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

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

Intuition

Example

λ-terms

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

OCaml

Intuition

Example

λ-terms

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

OCaml

- `fun x -> x+1`

Intuition

Example

λ-terms

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

OCaml

- `fun x -> x+1`
- `(fun x -> x+1) 2` $\to^{+}$ `3`

Intuition

Example

λ-terms

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

OCaml

- `fun x -> x+1`
- `(fun x -> x+1) 2` $\to^{+}$ `3`
- `if true then 1 else 0` $\to$ `1`

Intuition

Example

λ-terms

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

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

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

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

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

OCaml

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

Intuition

Example

λ-terms

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

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

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

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

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

Subterms

Definition

$\mathcal{S}\text{ub}(t)$ is set of subterms of t

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

Subterms

Definition

$\mathcal{S}\text{ub}(t)$ is set of subterms of t

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

Example

$$\mathcal{S}\text{ub}(\lambda xy.x)$$

Subterms

Definition

$\mathcal{S}\text{ub}(t)$ is set of subterms of t

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

Example

$$\mathcal{S}\text{ub}(\lambda xy.x) = \{\lambda xy.x\} \cup \mathcal{S}\text{ub}(\lambda y.x)$$

Subterms

Definition

$\mathcal{S}\text{ub}(t)$ is set of subterms of t

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

Example

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

Subterms

Definition

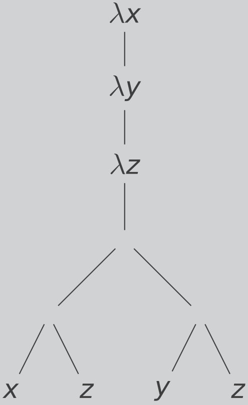
$\mathcal{S}\mathrm{ub}(t)$ is set of subterms of t

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

Example

$$\begin{aligned} \mathcal{S}\mathrm{ub}(\lambda xy.x) &= \{\lambda xy.x\} \cup \mathcal{S}\mathrm{ub}(\lambda y.x) \\ &= \{\lambda xy.x, \lambda y.x\} \cup \mathcal{S}\mathrm{ub}(x) \\ &= \{\lambda xy.x, \lambda y.x, x\} \end{aligned}$$

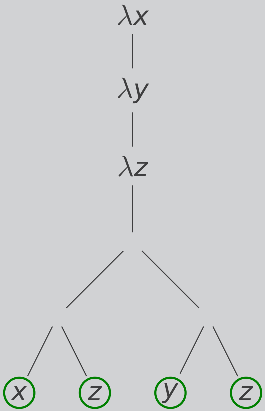
Example



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

$$\begin{aligned}\mathcal{S}\mathrm{ub}(t) = \{&t, \lambda yz.x\ z\ (y\ z),\\ &\lambda z.x\ z\ (y\ z),\\ &x\ z\ (y\ z), x\ z, y\ z,\\ &x, z, y\}\end{aligned}$$

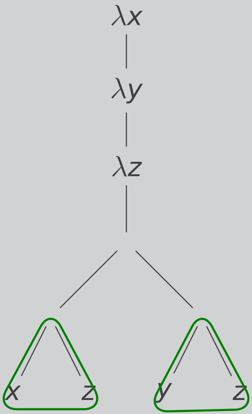
Example



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

$$\begin{aligned}\mathcal{S}\text{ub}(t) = \{&t, \lambda yz.x\ z\ (y\ z),\\ &\lambda z.x\ z\ (y\ z),\\ &x\ z\ (y\ z), x\ z, y\ z,\\ &{\color{red}x}, {\color{red}z}, {\color{red}y}\}\end{aligned}$$

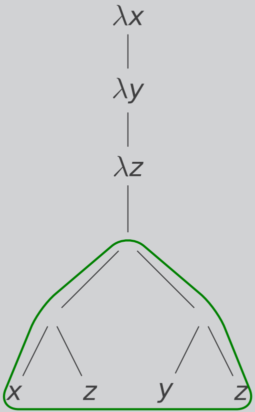
Example



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

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

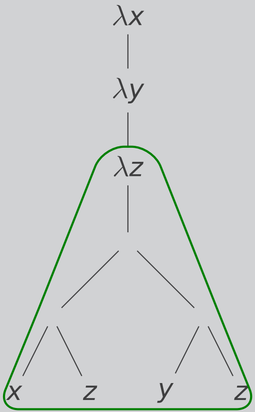
Example



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

$$\begin{aligned}\mathcal{S}\mathrm{ub}(t) = \{&t, \lambda yz.x\ z\ (y\ z),\\ &\lambda z.x\ z\ (y\ z),\\ &\color{red}{x\ z\ (y\ z)}, x\ z, y\ z,\\ &x, z, y\}\end{aligned}$$

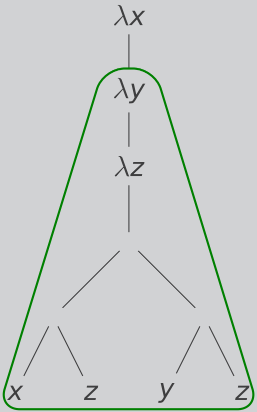
Example



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

$$\begin{aligned}\mathcal{S}\mathrm{ub}(t) = \{&t, \lambda yz.x\ z\ (y\ z),\\ &\lambda z.x\ z\ (y\ z),\\ &x\ z\ (y\ z), x\ z, y\ z,\\ &x, z, y\}\end{aligned}$$

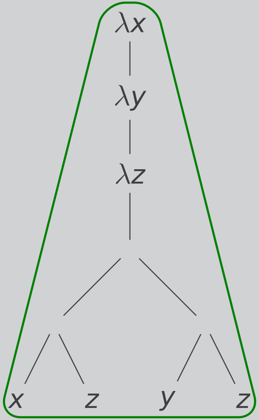
Example



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

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

Example



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

$$\begin{aligned}\mathcal{S}\mathrm{ub}(t) = \{&t, \lambda yz.x\ z\ (y\ z),\\ &\lambda z.x\ z\ (y\ z),\\ &x\ z\ (y\ z), x\ z, y\ z,\\ &x, z, y\}\end{aligned}$$

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

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

Free and Bound Variables

Definition

free variables

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

bound variables

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

Free and Bound Variables

Definition

free variables

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

bound variables

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

A λ-term without free variables is called closed.

t	$\mathcal{V}\text{ar}(t)$	$\mathcal{FV}\text{ar}(t)$	$\mathcal{BV}\text{ar}(t)$	closed
$\lambda x.x$				
$x\ y$				
$(\lambda x.x)\ x$				
$\lambda x.x\ y\ z$				

t	$\mathcal{V}\text{ar}(t)$	$\mathcal{FV}\text{ar}(t)$	$\mathcal{BV}\text{ar}(t)$	closed
$\lambda x.x$	$\{x\}$			
$x\ y$				
$(\lambda x.x)\ x$				
$\lambda x.x\ y\ z$				

t	$\mathcal{V}ar(t)$	$\mathcal{FV}ar(t)$	$\mathcal{BV}ar(t)$	closed
$\lambda x.x$	$\{x\}$	$\varnothing$		
$x\ y$				
$(\lambda x.x)\ x$				
$\lambda x.x\ y\ z$				

t	$\mathcal{V}\mathrm{ar}(t)$	$\mathcal{FV}\mathrm{ar}(t)$	$\mathcal{BV}\mathrm{ar}(t)$	closed
$\lambda x.x$	$\{x\}$	$\varnothing$	$\{x\}$	
$x\ y$				
$(\lambda x.x)\ x$				
$\lambda x.x\ y\ z$				

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

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

t	$\mathcal{V}\mathrm{ar}(t)$	$\mathcal{FV}\mathrm{ar}(t)$	$\mathcal{BV}\mathrm{ar}(t)$	closed
$\lambda x.x$	$\{x\}$	$\varnothing$	$\{x\}$	✓
$x\ y$	$\{x, y\}$	$\{x, y\}$		
$(\lambda x.x)\ x$				
$\lambda x.x\ y\ z$				

t	$\mathcal{V}\mathrm{ar}(t)$	$\mathcal{FV}\mathrm{ar}(t)$	$\mathcal{BV}\mathrm{ar}(t)$	closed
$\lambda x.x$	$\{x\}$	$\varnothing$	$\{x\}$	✓
$x\ y$	$\{x, y\}$	$\{x, y\}$	$\varnothing$	
$(\lambda x.x)\ x$				
$\lambda x.x\ y\ z$				

t	$\mathcal{V}\text{ar}(t)$	$\mathcal{FV}\text{ar}(t)$	$\mathcal{BV}\text{ar}(t)$	closed
$\lambda x.x$	$\{x\}$	$\varnothing$	$\{x\}$	✓
$x\ y$	$\{x, y\}$	$\{x, y\}$	$\varnothing$	X
$(\lambda x.x)\ x$				
$\lambda x.x\ y\ z$				

t	$\mathcal{V}\text{ar}(t)$	$\mathcal{FV}\text{ar}(t)$	$\mathcal{BV}\text{ar}(t)$	closed
$\lambda x.x$	$\{x\}$	$\varnothing$	$\{x\}$	✓
$x\ y$	$\{x, y\}$	$\{x, y\}$	$\varnothing$	X
$(\lambda x.x)\ x$	$\{x\}$			
$\lambda x.x\ y\ z$				

t	$\mathcal{V}\text{ar}(t)$	$\mathcal{FV}\text{ar}(t)$	$\mathcal{BV}\text{ar}(t)$	closed
$\lambda x.x$	$\{x\}$	$\emptyset$	$\{x\}$	✓
$x\ y$	$\{x, y\}$	$\{x, y\}$	$\emptyset$	X
$(\lambda x.x)\ x$	$\{x\}$	$\{x\}$		
$\lambda x.x\ y\ z$				

t	$\mathcal{V}\mathrm{ar}(t)$	$\mathcal{FV}\mathrm{ar}(t)$	$\mathcal{BV}\mathrm{ar}(t)$	closed
$\lambda x.x$	$\{x\}$	$\varnothing$	$\{x\}$	✓
$x\ y$	$\{x, y\}$	$\{x, y\}$	$\varnothing$	X
$(\lambda x.x)\ x$	$\{x\}$	$\{x\}$	$\{x\}$	
$\lambda x.x\ y\ z$				

t	$\mathcal{V}\text{ar}(t)$	$\mathcal{FV}\text{ar}(t)$	$\mathcal{BV}\text{ar}(t)$	closed
$\lambda x.x$	$\{x\}$	$\varnothing$	$\{x\}$	✓
$x\ y$	$\{x, y\}$	$\{x, y\}$	$\varnothing$	X
$(\lambda x.x)\ x$	$\{x\}$	$\{x\}$	$\{x\}$	X
$\lambda x.x\ y\ z$				

t	$\mathcal{V}ar(t)$	$\mathcal{FV}ar(t)$	$\mathcal{BV}ar(t)$	closed
$\lambda x.x$	$\{x\}$	$\varnothing$	$\{x\}$	✓
$x\ y$	$\{x, y\}$	$\{x, y\}$	$\varnothing$	X
$(\lambda x.x)\ x$	$\{x\}$	$\{x\}$	$\{x\}$	X
$\lambda x.x\ y\ z$	$\{x, y, z\}$			

t	$\mathcal{V}ar(t)$	$\mathcal{FV}ar(t)$	$\mathcal{BV}ar(t)$	closed
$\lambda x.x$	$\{x\}$	$\varnothing$	$\{x\}$	✓
$x\ y$	$\{x, y\}$	$\{x, y\}$	$\varnothing$	X
$(\lambda x.x)\ x$	$\{x\}$	$\{x\}$	$\{x\}$	X
$\lambda x.x\ y\ z$	$\{x, y, z\}$	$\{y, z\}$		

t	$\mathcal{V}\text{ar}(t)$	$\mathcal{FV}\text{ar}(t)$	$\mathcal{BV}\text{ar}(t)$	closed
$\lambda x.x$	$\{x\}$	$\varnothing$	$\{x\}$	✓
$x\ y$	$\{x, y\}$	$\{x, y\}$	$\varnothing$	X
$(\lambda x.x)\ x$	$\{x\}$	$\{x\}$	$\{x\}$	X
$\lambda x.x\ y\ z$	$\{x, y, z\}$	$\{y, z\}$	$\{x\}$	

t	$\mathcal{V}\text{ar}(t)$	$\mathcal{FV}\text{ar}(t)$	$\mathcal{BV}\text{ar}(t)$	closed
$\lambda x.x$	$\{x\}$	$\varnothing$	$\{x\}$	✓
$x\ y$	$\{x, y\}$	$\{x, y\}$	$\varnothing$	X
$(\lambda x.x)\ x$	$\{x\}$	$\{x\}$	$\{x\}$	X
$\lambda x.x\ y\ z$	$\{x, y, z\}$	$\{y, z\}$	$\{x\}$	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)

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

application of a function to some input

Computations

Idea

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

The β-rule (informal)

$$(\lambda x.s)\ t \to_\beta \underbrace{s\{x/t\}}_{\text{substitute } x \text{ by } t \text{ in } s}$$

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*"
- $(\lambda xy.x)\ v\ w \leadsto (\lambda y.v)\ w \leadsto v$ ✓
- $(\lambda xy.x)\ y\ z$

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*”
- $(\lambda xy.x)\ v\ w \leadsto (\lambda y.v)\ w \leadsto v$ ✓
- $(\lambda xy.x)\ y\ z \leadsto (\lambda y.y)\ z \leadsto z$

Blindly replacing does not suffice

Example

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

Blindly replacing does not suffice

Example

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

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"*
- $(\lambda xy.x)\ v\ w \leadsto (\lambda y.v)\ w \leadsto v$ ✓
- $(\lambda xy.x)\ y\ z \leadsto (\lambda y.y)\ z \leadsto z$ X
- 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*"
- $(\lambda xy.x)\ v\ w \rightsquigarrow (\lambda y.v)\ w \rightsquigarrow v$ ✓
- $(\lambda xy.x)\ y\ z \rightsquigarrow (\lambda y.y)\ z \rightsquigarrow z$ X
- clearly not intended (Problem: variable capture)
- $(\lambda xy.x)\ y\ z \to_\beta (\lambda y'.y)\ z \to_\beta y$

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*"
- $(\lambda xy.x)\ v\ w \rightsquigarrow (\lambda y.v)\ w \rightsquigarrow v$ ✓
- $(\lambda xy.x)\ y\ z \rightsquigarrow (\lambda y.y)\ z \rightsquigarrow z$ X
- 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

$$\sigma \colon \mathcal{V} \to \mathcal{T}(\mathcal{V})$$

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

$$\sigma\colon \mathcal{V} \to \mathcal{T}(\mathcal{V})$$

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

$$\sigma\colon \mathcal{V} \to \mathcal{T}(\mathcal{V})$$

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

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

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

$$x\sigma = \qquad\qquad y\sigma =$$

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

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

Substitutions (cont'd)

Definition (Application)

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

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

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

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

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

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

Substitutions (cont'd)

Definition (Application)

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

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

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

$$x\sigma = \lambda v.v\;w \qquad\qquad y\sigma = y$$

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

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

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

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

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

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

Substitutions (cont'd)

Definition (Application)

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

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

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

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

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

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

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

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

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

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

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

$$x\sigma = \lambda v.v\;w \qquad y\sigma = y$$

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

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

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

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

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

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

$$\lambda x.x \rightarrow_\beta$$

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

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

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

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

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

$$\lambda x.x \rightarrow_\beta$$

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

$$
\begin{aligned}
(\lambda x.x)\ (\lambda x.x) &\rightarrow_\beta \lambda x.x \\
(\lambda xy.y)\ (\lambda x.x) &\rightarrow_\beta \lambda y.y \\
(\lambda xyz.x\ z\ (y\ z))\ (\lambda x.x) &\rightarrow_\beta \\
(\lambda x.x\ x)\ (\lambda x.x\ x) &\rightarrow_\beta \\
\lambda x.x &\rightarrow_\beta \\
\lambda x.\underline{(\lambda y.y)\ z} &\rightarrow_\beta
\end{aligned}
$$

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

$$(\lambda xy.y)\ (\lambda x.x) \to_\beta \lambda y.y$$

$$(\lambda xyz.x\ z\ (y\ z))\ (\lambda x.x) \to_\beta \lambda yz.(\lambda x.x)\ z\ (y\ z)$$

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

$$\lambda x.x \to_\beta$$

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

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

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

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

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

$$\lambda x.x \rightarrow_\beta$$

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

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

$$(\lambda xy.y)\ (\lambda x.x) \to_\beta \lambda y.y$$

$$(\lambda xyz.x\ z\ (y\ z))\ (\lambda x.x) \to_\beta \lambda yz.(\lambda x.x)\ z\ (y\ z)$$

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

$\lambda x.x \to_\beta$ no β-step possible

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

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

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

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

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

$\lambda x.x \rightarrow_\beta$ no β-step possible

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

β-Reduction

Definition (Context)

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

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

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

- $C[s]$ denotes replacing $\Box$ 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

$$C_1 = \square \qquad C_1[\lambda x.x] =$$
$$C_2 = x\ \square \qquad C_2[\lambda x.x] =$$
$$C_3 = \lambda x.\square\ x \qquad C_3[\lambda x.x] =$$

β-Reduction

Definition (Context)

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

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

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

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

Example

$$\begin{aligned} C_1 &= \Box & \qquad C_1[\lambda x.x] &= \lambda x.x \\ C_2 &= x\ \Box & \qquad C_2[\lambda x.x] &= \\ C_3 &= \lambda x.\Box\ x & \qquad C_3[\lambda x.x] &= \end{aligned}$$

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

$$C_1 = \square \qquad C_1[\lambda x.x] = \lambda x.x$$
$$C_2 = x\ \square \qquad C_2[\lambda x.x] = x\ (\lambda x.x)$$
$$C_3 = \lambda x.\square\ x \qquad C_3[\lambda x.x] =$$

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

- $s \to_\beta^+ t$ denotes sequence $s = t_1 \to_\beta t_2 \to_\beta \cdots \to_\beta t_n = t$ 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

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

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)

Exercises

- Beta-reduce the term $(\lambda x.yy\ x)\ (\lambda x.yx\ (x\ y))\ (\lambda x.yx\ (x\ y))$ as many times as possible
- Implement a minimal λ-calculus together with a beta-reduction step
- Show the soundness of DEDUCT_ANTISYM_RULE from HOL Light
- Bonus exercise from this week (hint: peek into bool.ml and steal implication)