Programming Language Theory

Untyped λ-Calculus

陳亮廷 Chen, Liang-Ting

2020 邏輯、語言與計算暑期研習營
Formosan Summer School on Logic, Language, and Computation

Institute of Information Science, Academia Sinica

Deadline 17:00, 20 Aug

Assessment Assignment (10% + 10% + 10%)

Exam (100%)

Email liang.ting.chen.tw(at)gmail(dot)com

Please follow the instructions for assignments below.

1. Write down your name and your student number.
2. Use A4 paper only in physical form or in *non-scanned* PDF.
3. Be brief but comprehensive.
4. Submit assignments in person or by email with

subject [FLOLAC] PL HW%x%

attachment PL-HW%x% - %STDNO% - %NAME%.pdf

body (optional)

Untyped λ-Calculus: Statics

λ-calculus: Terms

Let $V := \{x, y, z, \ldots\}$ be a countably infinite set of *variables*.

Definition 1 (Syntax of λ-calculus)

The formation $M : \mathrm{Term}_\lambda$ is defined by

1. Variable: with the side condition that $x \in V$

$$\frac{}{x : \mathrm{Term}_\lambda} \text{ (var)}$$

2. Function application of M to the argument N:

$$\frac{M : \mathrm{Term}_\lambda \qquad N : \mathrm{Term}_\lambda}{M\,N : \mathrm{Term}_\lambda} \text{ (app)}$$

3. Function abstraction with an argument x and a function body M:

$$\frac{M : \mathrm{Term}_\lambda \qquad x \in V}{\lambda x.\,M : \mathrm{Term}_\lambda} \text{ (abs)}$$

Examples and non-examples

1. $(x \, y) \, z$
2. $x \, (y \, z)$
3. $\lambda x. \, y$
4. $\lambda x. \, x$
5. $\lambda s. \, (\lambda z. \, (s \, z))$
6. $\lambda a. \, (\lambda b. \, (a \, (\lambda c. \, a \, b)))$
7. $(\lambda x. \, x) \, (\lambda y. \, y)$

The following are NOT examples

1. $\lambda(\lambda x. \, x). \, y$
2. $\lambda x.$
3. $\lambda. \, x$
4. . . .

Consecutive abstractions

$$\lambda x_1 x_2 \ldots x_n. M := \lambda x_1. (\lambda x_2. (\ldots (\lambda x_n. M) \ldots))$$

Consecutive applications

$$M_1 M_2 M_3 \ldots M_n := (\ldots ((M_1 M_2) M_3) \ldots) M_n$$

Function body extends as far right as possible

$$\lambda x. M N := \lambda x. (M N)$$

instead of $(\lambda x. M) N$.

For example, $\lambda x_1. (\lambda x_2. x_1) \equiv \lambda x_1 x_2. x_1$ and *x y z* means *(x y) z*.
Warning. We apply these rules in our *meta*-language.

Meta-language and object-language

- *Meta-language* is the language we use to describe the object of study. E.g. English, or naive set theory.
- *Object-language* is the object of study. E.g., arithmetic expressions and λ-terms.

Naming a function is not supported in λ-calculus, so the following

$$\mathtt{id} := \lambda x.\, x$$

happens in the meta-language.

1. $\mathtt{id}$ is a symbol different from '$\lambda x.\, x$' in the meta-language.
2. $\mathtt{id}$ and $\lambda x.\, x$ are *syntactically equivalent* denoted by

$$\mathtt{id} \equiv \lambda x.\, x$$

   in the object language.

Example 2 (Identity function)

$$\mathtt{id} := \lambda x.\, x$$

Example 3 (Projections)

$$\mathtt{fst} := \lambda x.\, \lambda y.\, x \quad \text{and} \quad \mathtt{snd} := \lambda x.\, \lambda y.\, y$$

Example 4 (Church encoding of Natural numbers)

c_0	:=	$\lambda f x.\, x$
c_1	:=	$\lambda f x.\, f x$
c_2	:=	$\lambda f x.\, f(f x)$
c_3	:=	$\lambda f x.\, f(f(f(x)))$
	$\vdots$	

Definition 5

Two terms *M* an *N* are α-equivalent

$$M =_\alpha N$$

if variables *bound* by abstractions can be renamed to derive the same term.

Example 6

1. $\lambda x.\, x$ and $\lambda y.\, y$ are distinct λ-terms but $\lambda x.\, x =_\alpha \lambda y.\, y$.
2. $\lambda x.\, \lambda y.\, y =_\alpha \lambda z.\, \lambda y.\, y$.
3. $\lambda x.\, \lambda y.\, x \neq_\alpha \lambda x.\, \lambda y.\, y$.

α-equivalent terms are *programs of the same structure.*

The evaluation of λ-calculus happens in this form

$$\underbrace{(\lambda x.\, M)\, N}_{\beta\text{-redex}} \longrightarrow \underbrace{M\, [N/x]}_{\text{substitution of } N \text{ for a free variable } x \text{ in } M}$$

For example, $(\lambda x.\, x + 1)\, 3 \to 3 + 1$.

How to evaluate the following terms?

1. $(\lambda x.x)\, z$
2. $(\lambda x\, y.\, x)\, y$
3. $(\lambda y\, y.\, y)\, x$

Definition 7

The set **FV** of free variables of a term M is defined by

$$\mathsf{FV}(x) = \{x\}$$
$$\mathsf{FV}(\lambda x.\, M) = \mathsf{FV}(M) - \{x\}$$
$$\mathsf{FV}(M\, N) = \mathsf{FV}(M) \cup \mathsf{FV}(N)$$

Definition 8

1. A variable y in M is *free* if $y \in \mathsf{FV}(M)$.
2. A λ-term M is *closed* if $\mathsf{FV}(M) = \emptyset$.

Exercise: free variables

$$FV(x \ (\lambda y. \ y) \ z) = FV(x \ (\lambda y. \ y)) \cup FV(z)$$
$$= FV(x) \cup (FV(y) - \{y\}) \cup \{z\}$$
$$= \{x\} \cup (\{y\} - \{y\}) \cup \{z\}$$
$$= \{x, z\}$$

Calculate the set of free variables of following terms:

1. $x \ (y \ z)$
2. $\lambda x. \ y$
3. $\lambda x. \ x$
4. $\lambda s \ z. \ (s \ z))$
5. $(\lambda x. \ x) \ \lambda y. \ y$

Exercise: bound variables

Define

- **Var**(M) the set of variables of a term M by structural recursion on Λ.
- **BV**(M) the set of bound variables.

Untyped λ-Calculus: Substitution

Substitution

A substitution is a process of replacing *free* variables by another terms (on the meta-level).

The name of a variable does not matter but the location does. So, ...

1. bound variables should remain bound after substitution.
2. other free variables should remain free after substitution.

Concretely, we want to avoid ...

1. $(\lambda y.\, y)[x/y] \equiv (\lambda y.\, x)$
2. $(\lambda y.\, x)[y/x] \equiv (\lambda y.\, y)$

Naive substitution I

Definition 9

For $x \in V$ and $L : \mathtt{Term}_\lambda$, the substitution of L for x is defined by

$$
\begin{aligned}
x[L/x] &= L \\
y[L/x] &= y & \text{if } x \neq y \\
(M\ N)[L/x] &= M[L/x]\ N[L/x] \\
(\lambda y.\ M)[L/x] &= \lambda y.\ M[L/x]
\end{aligned}
$$

A bound variable may become free.

$$(\lambda x.\ x)[y/x] = \lambda x.\ y$$

14

Naive substitution II

Definition 10

For $x \in V$ and $L : \mathtt{Term}_\lambda$, the substitution of L for x is defined by

$$
\begin{aligned}
x[L/x] &= L \\
y[L/x] &= y && \text{if } x \neq y \\
(M\,N)[L/x] &= M[L/x]\,N[L/x] \\
(\lambda y.\,M)[L/x] &= \lambda y.\,M[L/x] && \text{if } x \neq y \\
(\lambda y.\,M)[L/x] &= \lambda y.\,M && \text{if } x = y
\end{aligned}
$$

A variable may be captured by an abstraction.

$$(\lambda x.y)[x/y] = \lambda x.\,x$$

Capture-avoiding substitution[1] of L for the free occurrences of x is a *partial* function $(\cdot)[L/x] \colon \text{Term}_\lambda \to \text{Term}_\lambda$ defined by

$$
\begin{aligned}
x[L/x] &= L \\
y[L/x] &= y && \text{if } x \neq y \\
(M\,N)[L/x] &= (M[L/x]\ N[L/x]) \\
(\lambda x.\,M)[L/x] &= \lambda x.\,M \\
(\lambda y.\,M)[L/x] &= \lambda y.\,M[L/x] && \text{if } x \neq y \text{ and } y \notin \text{FV}(L)
\end{aligned}
$$

Definition 11 (Freshness)

A variable y is fresh for L if $y \notin \text{FV}(L)$.

---

[1]Sign, this definition is nevertheless ill-defined.

If a variable *y* is *fresh* for *M*, the bound variable *x* of $\lambda x.\, M$ can be renamed to *y* without changing the meaning.

Definition 12 (α-conversion)

α-conversion is an judgement $M \rightarrow_\alpha N$ between two terms defined by

$$\frac{y \text{ is fresh for } M}{\lambda x.\, M \longrightarrow_\alpha \lambda y.\, M[y/x]}$$

Yet, $M\,(\lambda x.\, x) \longrightarrow_\alpha M\,(\lambda y.\, y)$ does not hold.

α-equivalence

Definition 13

$$\frac{x \text{ is a variable}}{x =_\alpha x} \qquad\qquad \frac{M_1 =_\alpha M_2 \quad N_1 =_\alpha N_2}{M_1 \, N_1 =_\alpha M_2 \, N_2}$$

$$\frac{M_1 \to_\alpha M_2}{M_1 =_\alpha M_2} \qquad\qquad \frac{M_1 =_\alpha M_2}{\lambda x. \, M_1 =_\alpha \lambda x. \, M_2}$$

Lemma 14

α-equivalence is an equivalence, i.e.

reflexivity $M =_\alpha M$ *for any term M;*

symmetry $N =_\alpha M$ *if $M =_\alpha N$;*

transitivity $L =_\alpha N$ *if $L =_\alpha M$ and $M =_\alpha N$.*

Proof of reflexivity

By induction on the formation of M.

1. $M = x$ for some $x \in V$. Then, by definition $x =_\alpha x$ holds.
2. $M = M_1 M_2$. Then, by induction hypothesis, we have derivations D_1 and D_2 for $M_1 =_\alpha M_1$ and $M_2 =_\alpha M_2$ respectively. Therefore, we have the desired derivation

$$
\cfrac{\cfrac{\vdots}{M_1 =_\alpha M_1} D_1 \quad \cfrac{\vdots}{M_1 =_\alpha M_2} D_2}{M_1 M_2 =_\alpha M_1 M_2}
$$

3. $M = \lambda x. M'$. By induction hypothesis, we have a derivation D for $M' =_\alpha M'$. Hence,

$$
\cfrac{\cfrac{\vdots}{M' =_\alpha M'} D}{\lambda x. M' =_\alpha \lambda x. M'}
$$

Proof of symmetry

By induction on the derivation D of $M =_\alpha N$. The only interesting case is that D is derived from an α-conversion, i.e.

$$\lambda x.\, M' \to_\alpha \lambda y.\, M'[y/x]$$

and y is fresh for M'. We know that $x \notin \mathsf{FV}(M'[y/x])$ since the substitution $[y/x]$ replaces[2] the free variable x by y. Therefore, we have $\lambda x.\, M'[y/x][x/y] \equiv \lambda x.\, M'$. It follows that

$$\lambda y.M'[y/x] \to_\alpha \lambda x.M'$$

Hence, $N =_\alpha M$.

---

[2]We actually need to show that $x \notin \mathsf{FV}(M[y/x])$ whenever $\mathsf{FV}(M[y/x])$ is defined.

Proof of transitivity

By induction on the derivations D_1 and D_2 of $L =_\alpha M$ and $M =_\alpha N$ respectively. The interesting case is when D_i's are given by α-conversion

$$\lambda x.\, M' \to_\alpha \lambda y.\, M'[y/x] \to_\alpha \lambda z.\, M'[y/x][z/y].$$

It follows that

$$\lambda x.\, M' \to_\alpha \lambda z.\, M'[z/x]$$

where the freshness condition clearly holds (why?) and also that

$$M'[y/x][z/y] \equiv M'[z/x].$$

Hence, transitivity holds for this case.

Example 15

$$(\lambda y.\, y)\,(\lambda x.\, x) =_\alpha (\lambda x.\, x)\,(\lambda y.\, y)$$

Why? We use the fact that $=_\alpha$ is an equivalence!

Proof.

$$\frac{\dfrac{\lambda x.\, x \to_\alpha \lambda y.\, x[y/x]}{\lambda x.\, x =_\alpha \lambda y.\, y}}{(\lambda y.\, y)\,(\lambda x.\, x) =_\alpha (\lambda y.\, y)\,(\lambda y.\, y)} \qquad \frac{\dfrac{\lambda y.\, y \to_\alpha \lambda x.\, y[x/y]}{\lambda y.\, y =_\alpha \lambda x.\, x}}{(\lambda y.\, y)\,(\lambda y.\, y) =_\alpha (\lambda x.\, x)\,(\lambda y.\, y)}$$
$$(\lambda y.\, y)\,(\lambda x.\, x) =_\alpha (\lambda x.\, x)\,(\lambda y.\, y)$$

$\square$

Which of the following pairs are α-equivalent? Why?

1. x and y
2. $\lambda x\, y.\, y$ and $\lambda z\, y.\, y$
3. $\lambda x\, y.\, x$ and $\lambda y\, x.\, y$
4. $\lambda x\, y.\, x$ and $\lambda x\, y.\, y$

Convention
Terms are equal up to α-equivalence of bound variables.

Feel free to rename any bound variable whenever convenient.

Untyped λ-Calculus: Dynamics

Definition 16 (β-conversion)

β-conversion is defined by

$$\underbrace{(\lambda x.\, M)\, N}_{\beta\text{-redex}} \longrightarrow_\beta M[N/x]$$

$$((\lambda x.\, \lambda y.\, x)\, M) \longrightarrow_\beta (\lambda y.\, x)[M/x]$$
$$\equiv \lambda y.\, x[M/x] \equiv \lambda y.\, M$$

$$((\lambda x\, y.\, x)\, M)\, N \longrightarrow_\beta ?$$

One-step full β-reduction

Definition 17

The one-step full β-reduction is defined by

$$\frac{}{(\lambda x.\, M)\, N \longrightarrow_{\beta 1} M[N/x]}$$

$$\frac{M_1 \longrightarrow_{\beta 1} M_2}{M_1\, N \longrightarrow_{\beta 1} M_2\, N}$$

$$\frac{M_1 \longrightarrow_{\beta 1} M_2}{\lambda x.\, M_1 \longrightarrow_{\beta 1} \lambda x.\, M_2}$$

$$\frac{N_1 \longrightarrow_{\beta 1} N_2}{M\, N_1 \longrightarrow_{\beta 1} M\, N_2}$$

$$((\lambda x\, y.\, x)\, M)\, N \longrightarrow_{\beta 1} (\lambda y.\, M)\, N$$
$$\longrightarrow_{\beta 1} M[N/y]$$

Multi-step full β-reduction

Definition 18

$$\frac{}{M \longrightarrow_{\beta*} M} \text{ (0-step)}$$

$$\frac{L \longrightarrow_{\beta 1} M \qquad M \longrightarrow_{\beta*} N}{L \longrightarrow_{\beta*} N} \text{ (}n+1\text{-steps)}$$

α-conversion during β-conversion

Renaming of bound variables may need to happen during reduction:

$$
\begin{aligned}
(\lambda y.\, y\, y)\, (\lambda z\, x.\, z\, x) \longrightarrow_{\beta 1} \quad & (\lambda z\, x.\, z\, x)\, (\lambda z\, x.\, z\, x) \\
\longrightarrow_{\beta 1} \quad & \lambda x.\, (\lambda z\, x.\, z\, x)\, x \\
=_{\alpha} \quad & \lambda x.\, (\lambda z\, y.\, z\, y)\, x \\
\longrightarrow_{\beta 1} \quad & \lambda x.\, (\lambda y.\, x\, y)
\end{aligned}
$$

Even worse, we actually need infinitely many variables:

$$(\lambda y.\, y\, s\, y)\, (\lambda t\, z\, x.\, z\, (t\, x)\, z)$$

Exercise

Evaluate the above term.

Definition 19

M and N have the same *computational meaning* if $M =_\beta N$ where $=_\beta$ is defined inductively by

$$\frac{M \longrightarrow_{\beta 1} N}{M =_\beta N} \qquad\qquad \frac{M =_\beta N}{N =_\beta M}$$

$$\overline{M =_\beta M} \qquad\qquad \frac{L =_\beta M \qquad M =_\beta N}{L =_\beta N}$$

- $c_2 =_\beta (\lambda x\,y.y)\ c_1\ c_2$
- $\lambda z.\,(\lambda x\,y.x)\ z =_\beta \lambda z\,y.\,z$

28

So far, we have notions of equality and reduction.

- $1 + 1 \neq_\alpha 2$
- $1 + 1 \longrightarrow_{\beta*} 2$ but $2 \not\longrightarrow_{\beta*} 1 + 1$
- $1 + 1 =_\beta 2$

Each of above statements says the following.

- $1 + 1$ is a different expression from 2.
- $1 + 1$ reduces to 2, but 2 does not reduce to $1 + 1$.
- $1 + 1$ and 2 have the same computational meaning.

Programming in λ-Calculus

Church encoding of boolean values

Boolean and conditional can be encoded as combinators.

Boolean

$$\texttt{True} \quad := \quad \lambda x\,y.\,x$$
$$\texttt{False} \quad := \quad \lambda x\,y.\,y$$

Conditional

$$\texttt{if} := \lambda b\,x\,y.\,b\,x\,y$$
$$\texttt{if True } M\,N \longrightarrow_{\beta*} M$$
$$\texttt{if False } M\,N \longrightarrow_{\beta*} N$$

for any two λ-terms M and N.

Natural numbers can be encoded as λ-terms, so can arithmetic operations.

Church numerals

$$
\begin{aligned}
c_0 &:= \lambda f x.\, x \\
c_1 &:= \lambda f x.\, f x \\
c_2 &:= \lambda f x.\, f(f x) \\
c_{n+1} &:= \lambda f x.\, f^{n+1}(x)
\end{aligned}
$$

where $f^1(x) := f x$ and $f^{n+1}(x) := f(f^n(x))$.

Church Encoding of natural numbers ii

Successor

$$\text{succ} \quad := \quad \lambda n.\, \lambda f x.\, f\,(n\, f\, x)$$
$$\text{succ } c_n \quad \longrightarrow_{\beta*} \quad c_{n+1}$$

for any natural number $n \in \mathbb{N}$.

Addition

$$\text{add} \quad := \quad \lambda n\, m.\, \lambda f x.\, n\, f\,(m\, f\, x)$$
$$\text{add } c_n\, c_m \quad \longrightarrow_{\beta*} \quad c_{n+m}$$

Conditional

$$\begin{aligned}
\texttt{ifz} \quad &:= \quad \lambda n\, x\, y.\, n\, (\lambda z.\, y)\, x \\
\texttt{ifz } c_0\, M\, N \quad &\longrightarrow_{\beta*} \quad M \\
\texttt{ifz } c_{n+1}\, M\, N \quad &\longrightarrow_{\beta*} \quad N
\end{aligned}$$

Predecessor

$$\begin{aligned}
\texttt{pred} \quad &:= \quad \lambda n.\, \lambda f x.\, ? \\
\texttt{pred } c_0 \quad &\longrightarrow_{\beta*} \quad c_0 \\
\texttt{pred } c_{n+1} \quad &\longrightarrow_{\beta*} \quad c_n
\end{aligned}$$

Exercise

1. Define the *flip* operation, i.e. a λ-term $\mathtt{flip}$ such that

$$\mathtt{flip} \; M \; N \; P =_\beta M \; P \; N$$

2. Define Boolean operations $\mathtt{not}$, $\mathtt{and}$, and $\mathtt{or}$.
3. Evaluate $\mathtt{succ} \; \mathtt{c}_0$ and $\mathtt{add} \; \mathtt{c}_1 \; \mathtt{c}_2$.
4. Define the multiplication $\mathtt{mult}$ over Church numerals.
5. (Hard) Define $\mathtt{pred}$ so that $\mathtt{pred} \; \mathtt{c}_0 =_\beta \mathtt{c}_0$ and $\mathtt{pred} \; \mathtt{c}_{n+1} =_\beta \mathtt{c}_n$.

General Recursion, informally

The summation $\sum_{i=0}^{n} i$ for $n \in \mathbb{N}$ *can be* defined as

$$\mathsf{sum}(n) = \begin{cases} 0 & \text{if } n = 0 \\ n + \mathsf{sum}(n-1) & \text{otherwise.} \end{cases}$$

We cannot define recursion via self-reference in λ-calculus, can we avoid it? Consider the function $G \colon (\mathbb{N} \to \mathbb{N}) \to (\mathbb{N} \to \mathbb{N})$ defined by

$$(Gf)(n) := \begin{cases} 0 & \text{if } n = 0 \\ n + f(n-1) & \text{otherwise.} \end{cases} \tag{1}$$

If there exists sum' with $G(\mathsf{sum}') = \mathsf{sum}'$, then $\mathsf{sum}' = \mathsf{sum}$.

Proposition 20

Define

$$\mathsf{Y} := \lambda f. (\lambda x. f(xx))(\lambda x. f(xx)).$$

Then,

$$\mathsf{Y}F \longrightarrow_{\beta 1} (\lambda x. F(xx))(\lambda x. F(xx))$$
$$\longrightarrow_{\beta 1} F((\lambda x. F(xx))(\lambda x. F(xx)))$$

for every λ-term F.

Summation, formally

Using the combinators we have known so far, the equation (1) can be defined as λ-terms:

$$G := \lambda f n.\, \mathtt{ifz}\ n\ \mathsf{c}_0\ (\mathtt{add}\ n\ (f\,(\mathtt{pred}\ n)))$$
$$\mathtt{sum} := \mathsf{Y}G$$

For example

$$\begin{aligned}
\mathtt{sum}\ \mathsf{c}_1 &\equiv (\mathsf{Y}G)\ \mathsf{c}_1 \\
&\longrightarrow_{\beta 1} G'\ \mathsf{c}_1 \\
&\longrightarrow_{\beta 1} G\ G'\ \mathsf{c}_1 \\
&\longrightarrow_{\beta 1} (\lambda n.\, \mathtt{ifz}\ n\ \mathsf{c}_0\ (\mathtt{add}\ n\ (G'\,(\mathtt{pred}\ n))))\ \mathsf{c}_1 \\
&\longrightarrow_{\beta 1} \mathtt{ifz}\ \mathsf{c}_1\ \mathsf{c}_0\ (\mathtt{add}\ \mathsf{c}_1\ (G'\,(\mathtt{pred}\ \mathsf{c}_1))) \\
&\longrightarrow_{\beta 1} \cdots
\end{aligned}$$

where $G' := ((\lambda x.\, G\,(x\,x))\,(\lambda x.\, G\,(x\,x)))$.

Turing's fixed-point combinator

Here is a fixed-point operator such that $\Theta F \longrightarrow_{\beta*} F(\Theta F)$.

Proposition 21

Define

$$\Theta := (\lambda x f. f(x \, x \, f)) \, (\lambda x f. f(x \, x \, f))$$

Then,

$$\Theta F \longrightarrow_{\beta*} F(\Theta F)$$

Try Turing's fixed-point combinator with G to define $\sum_{i=0}^{n} i$.

$$G := \lambda f n. \, \mathtt{ifz} \, n \, \mathtt{c_0} \, (\mathtt{add} \, n \, (f(\mathtt{pred} \, n)))$$
$$\mathtt{sum} := \Theta \, G$$

Exercise

1. Evaluate **sum** c_1 to its normal form in detail.
2. Define the factorial $n!$ on Church numerals with Turing's fixed-point combinator.

Properties of λ-Calculus

Example 22

Suppose $M : \mathtt{Term}_\lambda$ and $y \notin \mathsf{FV}(M)$. Then, consider

$$(\lambda y.\, M)\, ((\lambda x.\, x\, x)(\lambda x.\, x\, x))$$
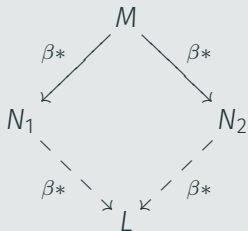
Observations:

- Some evaluation may diverge while some may converge.
- Full β-reduction lacks for determinacy.

Question:

- Does every path give the same evaluation?

Confluence

Theorem 23 (Church-Rosser)

Given N_1 and N_2 with $M \longrightarrow_{\beta*} N_1$ and $M \longrightarrow_{\beta*} N_2$, there is L such that $N_1 \longrightarrow_{\beta*} L$ and $N_2 \longrightarrow_{\beta*} L$.

$$
\begin{array}{ccc}
 & M & \\
{\scriptstyle\beta*}\swarrow & & \searrow{\scriptstyle\beta*} \\
N_1 & & N_2 \\
{\scriptstyle\beta*}\searrow & & \swarrow{\scriptstyle\beta*} \\
 & L &
\end{array}
$$

Normal form

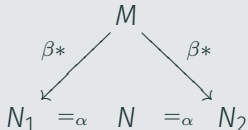We say that M is *in normal form* if $M \not\longrightarrow_{\beta 1}$.

Lemma 24

Suppose that M is in normal form. Then
$M \longrightarrow_{\beta *} N \implies M =_\alpha N.$

Corollary 25 (Uniqueness of normal forms)

Suppose that N_1 and N_2 are in normal form. Then,

$$
\begin{array}{ccccc}
 & & M & & \\
 & \swarrow^{\beta *} & & \searrow^{\beta *} & \\
N_1 & =_\alpha & N & =_\alpha & N_2
\end{array}
$$

Computationally equal terms have a confluent term

Corollary 26

If $M =_\beta N$, then there exists L satisfying

$$
\begin{array}{ccc}
M & =_\beta & N \\
& & \\
\beta* & & \beta* \\
& L &
\end{array}
$$

Proof sketch.

By induction on the derivation of $M =_\beta N$.

For example, if $M \longrightarrow_{\beta 1} N$, then choose L as N. $\square$

An evaluation strategy is a procedure of selecting β-redexes to reduce. It is a subset $\longrightarrow_{\mathrm{ev}}$ of the full β-reduction $\longrightarrow_{\beta 1}$.

Innermost β-redex does not contain any β-redex.

Outermost β-redex is not contained in any other β-redex.

Evaluation strategies ii

the leftmost-outermost (*normal order*) strategy reduces the
leftmost outermost β-redex in a term first. For
example,

$$\frac{(\lambda x.\,(\lambda y.\,y)\,x)}{}\quad \frac{(\lambda x.\,(\lambda y.\,y\,y)\,x)}{}$$

$$\longrightarrow_{\beta 1} \underline{(\lambda y.\,y)}\quad \underline{(\lambda x.\,(\lambda y.\,y\,y)\,x)}$$

$$\longrightarrow_{\beta 1} \lambda x.\,\underline{(\lambda y.\,y\,y)}\quad \underline{x}$$

$$\longrightarrow_{\beta 1} (\lambda x.\,x\,x)$$

$$\not\longrightarrow_{\beta 1}$$

Evaluation strategies iii

the leftmost-innermost strategy reduces the leftmost
        innermost β-redex in a term first. For example,

$$(\lambda x.\,\underline{(\lambda y.\,y)\ x})\,(\lambda x.\,(\lambda y.\,y\,y)\,x)$$
$$\longrightarrow_{\beta 1}(\lambda x.\,x)\,(\lambda x.\,\underline{(\lambda y.\,y\,y)\ x})$$
$$\longrightarrow_{\beta 1}\underline{(\lambda x.\,x)}\ \ \underline{(\lambda x.\,x\,x)}$$
$$\longrightarrow_{\beta 1}(\lambda x.\,x\,x)$$
$$\not\longrightarrow_{\beta 1}$$

the rightmost-innermost/outermost strategy are defined
        similarly where terms are reduced from right to
        left instead.

Call-by-value strategy rightmost-outermost but not under any abstraction

Call-by-name strategy leftmost-outermost but not under any abstraction

Proposition 27 (Determinacy)

Each of evaluation strategies is deterministic, i.e. if
$M \longrightarrow_{\beta 1} N_1$ *and* $M \longrightarrow_{\beta 1} N_2$ *then* $N_1 = N_2$.

Exercise

Define following terms

$$\Omega := (\lambda x.\, x x)\, (\lambda x.\, x x)$$
$$K_1 := \lambda x y.\, x$$

Evaluate

$$K_1\, z\, \Omega$$

using the call-by-value and the call-by-name strategy respectively.

Normalisation

Definition 28

1. *M* is in *normal form* if $M \not\longrightarrow_{\beta 1} N$ for any *N*.
2. *M* is *weakly normalising* if $M \longrightarrow_{\beta *} N$ for some *N* in normal form.

1. Ω is not weakly normalising.
2. K_1 is normal and thus weakly normalising.
3. $K_1 \, z \, \Omega$ is weakly normalising.

Theorem 29

The normal order strategy reduces every weakly normalising term to a normal form.

Homework

1. (25%) Show that $\longrightarrow_{\beta *}$ is transitive, i.e. $L \longrightarrow_{\beta *} N$ whenever $L \longrightarrow_{\beta *} M$ and $M \longrightarrow_{\beta *} N$. **Hint**. By induction on $L \longrightarrow_{\beta *} M$.
2. (25%) Show Lemma 24
3. (25%) Show Corollary 25
4. (25%) Show Corollary 26.