λ-Calculus

Untyped λ-Calculus

陳亮廷 Chen, Liang-Ting

Formosan Summer School on Logic, Language, and Computation (FLOLAC) 2022

Institute of Information Science, Academia Sinica

Assessment guidelines

Deadline 17:00, 10 Aug

Assessment **Assignment** (15%)
Exam (100%)

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

Please follow the instructions below.

1. Use A4 paper.
2. Write down your name and student id.
3. Be clear and brief.
4. Submit assignments in person or by email as PDF with

   subject `[FLOLAC] PL HW%x%`
   attachment `PL-HW%x% - %STDNO% - %NAME%.pdf`
   body (optional)

Untyped λ-Calculus: Statics

λ-calculus: Term

Definition 1 (Syntax of λ-calculus)

Given a set V of variables, the term formation judgement is defined by

Variable

$$\frac{x \text{ is in } V}{x \quad \mathtt{Term}_V} \text{(var)}$$

Application of M to the argument N

$$\frac{M \quad \mathtt{Term}_V \qquad N \quad \mathtt{Term}_V}{M\ N \quad \mathtt{Term}_V} \text{(app)}$$

Abstraction with an argument x and a function body M

$$\frac{M \quad \mathtt{Term}_V \qquad x \text{ is in } V}{\lambda x.\, M \quad \mathtt{Term}_V} \text{(abs)}$$

An Example

The judgement

$$\boxed{\lambda p.\, \lambda a.\, \lambda b.\, (p\ a)\ b \quad \mathtt{Term}_V}$$

is justified by the following derivation

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{p \text{ is in } V}{p \quad \mathtt{Term}_V} \qquad \dfrac{a \text{ is in } V}{a \quad \mathtt{Term}_V}}{p\ a \quad \mathtt{Term}_V} \qquad \dfrac{b \text{ is in } V}{b \quad \mathtt{Term}_V}}{(p\ a)\ b \quad \mathtt{Term}_V} \qquad b \text{ is in } V}{\lambda b.\, (p\ a)\ b \quad \mathtt{Term}_V} \qquad a \text{ is in } V}{\lambda a.\, \lambda b.\, (p\ a)\ b \quad \mathtt{Term}_V} \qquad p \text{ is in } V}{\lambda p.\, \lambda a.\, \lambda b.\, (p\ a)\ b \quad \mathtt{Term}_V}$$

N.B. brackets '(' and ')' are not parts of terms and they are used only to group a term.

More Example 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. $\ldots$

Conventions

Consecutive abstractions

$$\lambda x_1\, x_2\ \ldots x_n.\, M \equiv \lambda x_1.\, (\lambda x_2.\, (\ldots (\lambda x_n.\, M) \ldots))$$

Consecutive applications

$$M_1\ M_2\ M_3\ \ldots\ M_n \equiv (\ldots ((M_1\ M_2)\ M_3) \ldots)\ M_n$$

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 $(x\ y)\ z$.

Examples

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

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

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

happens in the meta-language.

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

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

Example 2 (Identity function)

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

Example 3 (Projections)

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

Remember that there are only three constructs in λ-calculus. For convenience, we normally use a surface language to generate terms in the object-language.

α-equivalence, informally

Definition 4

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 5

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 modulo the name of bound variables.*

Evaluation, informally

The evaluation of λ-calculus is of this form

$$\boxed{\cdots \underbrace{(\lambda x.\, M)\, N}_{\beta\text{-redex}} \cdots} \longrightarrow_{\beta 1} \boxed{\cdots \qquad \underbrace{M\, [N/x]}_{\text{substitution of } N \text{ for } x \text{ in } M} \qquad \cdots}$$

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

Structural recursion: Free variables

Definition 6

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

$$\begin{aligned} \mathbf{FV}\colon \Lambda_V &\to \mathcal{P}(V) \\ \mathbf{FV}(x) &= \{x\} \\ \mathbf{FV}(\lambda x.\, M) &= \mathbf{FV}(M) - \{x\} \\ \mathbf{FV}(M\ N) &= \mathbf{FV}(M) \cup \mathbf{FV}(N) \end{aligned}$$

Definition 7

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

Exercise

The set of free variables of a term is calculated by definition readily, e.g.,

$$\begin{aligned}\mathsf{FV}(x\ (\lambda y.\, y)\ z) &= \mathsf{FV}(x\ (\lambda y.\, y)) \cup \mathsf{FV}(z) \\ &= \mathsf{FV}(x) \cup (\mathsf{FV}(y) - \{y\}) \cup \{z\} \\ &= \{x\} \cup (\{y\} - \{y\}) \cup \{z\} \\ &= \{x, z\}\end{aligned}$$

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: Height

The height of a term is given informally as follows:

1. the height of a variable is zero;
2. the height of an application is the maximum of the heights of its subterms plus 1;
3. the height of an abstraction is the height of its body plus 1.

Define the height function $h \colon \mathtt{Term}_V \to \mathbb{N}$ inductively.

Untyped λ-Calculus: Substitution

Substitution

A substitution is a process of replacing *free* variables by another terms on the meta-level. Hence, a substitution of N for a free variable x is a function

$$\_[N/x] : \texttt{Term}_V \to \texttt{Term}_V$$

The name of a variable does not matter but its location does.

1. bound variables should remain bound after substitution.
2. free variables which are not x 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

For $x \in V$ and $L : \texttt{Term}_V$, 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 after substitution, e.g.,

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

so this is not the one we want.

Naive substitution II

For $x \in V$ and $L : \mathtt{Term}_V$, 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 after substitution, e.g.,

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

so again it is not the desired definition.

Capture-avoiding substitution

Definition 8

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

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

[1]Sign, this definition is still not rigorous.

Renaming of bound variables

Definition 9 (Freshness)

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

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 10 (α-conversion)

α-conversion is an judgement $M \to_\alpha N$ between 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 follow by definition, so we introduce a new judgement to allow α-conversion in any subterm of a term.

α-equivalence

Definition 11

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

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

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 $M =_\alpha N$.

All of these can be proved by induction on the derivation of $M =_\alpha M$.

Example 12

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

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

Proof.

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

□

Exercise

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

α-equivalent terms are identified.

In the following development, we do not distinguish M and N if $M =_\alpha N$ at all. Feel free to rename any bound variable whenever convenient.

Untyped λ-Calculus: Dynamics

β-conversion

Definition 13 (β-conversion)

β-conversion is a judgement $M \longrightarrow_\beta N$ defined by

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

for any x, M and N.

By definition, we can conclude that

$$\begin{aligned}(\lambda x.\, \lambda y.\, x)\ M &\longrightarrow_\beta (\lambda y.\, x)[M/x] \\ &\equiv \lambda y.\, x[M/x] \equiv \lambda y.\, M\end{aligned}$$

but not $((\lambda x\, y.\, x)\ M)\ N \longrightarrow_\beta (\lambda y.\, M)\ N$, since the above judgement is defined only for β-redexes.

One-step β-reduction

One-step β-reduction extends β-conversion to any subterm of a term.

Definition 14

The *one-step (full) β-reduction* is defined inductively by

$$\frac{M[N/x] \text{ is defined}}{(\lambda x.\, M)\ N \longrightarrow_{\beta 1} M[N/x]} \qquad \frac{M_1 \longrightarrow_{\beta 1} M_2}{M_1\ N \longrightarrow_{\beta 1} M_2\ N}$$

$$\frac{M_1 \longrightarrow_{\beta 1} M_2}{\lambda x.\, M_1 \longrightarrow_{\beta 1} \lambda x.\, M_2} \qquad \frac{N_1 \longrightarrow_{\beta 1} N_2}{M\ N_1 \longrightarrow_{\beta 1} M\ N_2}$$

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

Multi-step full β-reduction

It is convenient to represents a sequence of β-reductions

$$M \longrightarrow_{\beta 1} M_1 \longrightarrow_{\beta 1} \ldots \longrightarrow_{\beta 1} N$$

by a single judgement $M \longrightarrow_{\beta *} N$.

Definition 15

The *multi-step (full) β-reduction* is defined inductively by

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

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

$M \longrightarrow_{\beta *} N$ is transitive

Lemma 16

For every derivations of $L \longrightarrow_{\beta *} M$ and $M \longrightarrow_{\beta *} N$, there is a derivation of $L \longrightarrow_{\beta *} N$.

We often omit the term "derivation" and say "if $L \longrightarrow_{\beta *} M$ and $M \longrightarrow_{\beta *} N$ then $L \longrightarrow_{\beta *} N$" instead.

Proof.

By induction on the derivation d of $L \longrightarrow_{\beta *} M$.

1. If d is given by (0-step), then $L =_\alpha M$ (by convention).
2. If d is given by (n+1-step), i.e. there exists M' such that $L \longrightarrow_{\beta 1} M'$ and $M' \longrightarrow_{\beta *} M$. By induction hypothesis, every derivation $M \longrightarrow_{\beta *} N$ gives rise to a derivation of $M' \longrightarrow_{\beta *} N$. Hence, by (n+1-step), we have a derivation of $L \longrightarrow_{\beta *} N$.

□

α-conversion during β-reduction

Renaming of bound variables may need to happen during reduction:

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

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.

Computational meaning

Two terms M and N may not have the same structure or even not reducible from one to the other, but they may have the same meaning with respect to computation.

Definition 17

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

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

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

Summary

SUMMARISE HERE ALL THE RELATIONS JUST INTRODUCED.

Programming in λ-Calculus

Church encoding of boolean values

Boolean and conditional can be encoded as combinators.

Boolean

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

Conditional

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

for any two λ-terms M and N.

Church Encoding of natural numbers i

Natural numbers as well as arithmetic operations can be encoded in untyped lambda calculus.

Church numerals

$$\begin{aligned} \mathsf{c}_0 &:= \lambda f x.\, x \\ \mathsf{c}_1 &:= \lambda f x.\, f x \\ \mathsf{c}_2 &:= \lambda f x.\, f\,(f x) \\ \mathsf{c}_{n+1} &:= \lambda f x.\, f^{n+1}\,(x) \end{aligned}$$

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

Church Encoding of natural numbers ii

Successor

$$\begin{aligned} \texttt{succ} &:= \lambda n.\, \lambda f x.\, f\,(n\,f\,x) \\ \texttt{succ}\ \mathsf{c}_n &\longrightarrow_{\beta *} \mathsf{c}_{n+1} \end{aligned}$$

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

Addition

$$\begin{aligned} \texttt{add} &:= \lambda n\, m.\, \lambda f x.\; n\, f\, (m\, f\, x) \\ \texttt{add}\ \mathsf{c}_n\ \mathsf{c}_m &\longrightarrow_{\beta *} \mathsf{c}_{n+m} \end{aligned}$$

Conditional

$$\begin{aligned} \texttt{ifz} &:= \lambda n\, x\, y.\; n\, (\lambda z.\, y)\, x \\ \texttt{ifz}\ \mathsf{c}_0\ M\ N &\longrightarrow_{\beta *} M \\ \texttt{ifz}\ \mathsf{c}_{n+1}\ M\ N &\longrightarrow_{\beta *} N \end{aligned}$$

Exercise

1. Define Boolean operations `not`, `and`, and `or`.
2. Evaluate `succc` c_0 and `add` c_1 c_2.
3. Define the multiplication `mult` over Church numerals.

General Recursion via self-reference

The summation $\sum_{i=0}^{n} i$ for $n \in \mathbb{N}$ is usually described by self-reference in mathematics as follows.

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

This cannot be done in λ-calculus directly. (Why?)

Observation

If *sum* is unfolded as many times as it requires, then

$$sum(n) = \begin{cases} 0 & \text{if } n = 0 \\ 1 + sum(0) & n = 1 \\ 2 + sum(1) & n = 2 \\ \dots & \\ n + sum(n-1) & \text{otherwise.} \end{cases}$$

Curry's paradoxical combinator

The *Y combinator* is defined as a term

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

Proposition 18

Y *is a fixed-point operator, i.e.*

$$\begin{aligned}\mathsf{Y}F &\longrightarrow_{\beta 1} (\lambda x.\, F\,(x\,x))\,(\lambda x.\, F\,(x\,x)) \\ &\longrightarrow_{\beta 1} F\,((\lambda x.\, F\,(x\,x))\,(\lambda x.\, F\,(x\,x)))\end{aligned}$$

for every λ-term F. In particular, $\mathsf{Y}F =_\beta F(\mathsf{Y}F)$.

Intuitively, $\mathsf{Y}F$ defines recursion where F describes each iteration.

Summation via Y

We encode the following recursion

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

by generalising each iteration G with an additional function f

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

so that $\texttt{sum} := \mathsf{Y}G$. For example,

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

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

Turing's fixed-point combinator

Recall that $\mathsf{Y}G =_\beta G(\mathsf{Y}\ G)$ but $\mathsf{Y}G \longrightarrow_{\beta*} G(\mathsf{Y}\ G)$ does not hold. Here is a fixed-point operator such that $\Theta F \longrightarrow_{\beta*} F(\Theta F)$.

Proposition 19

Define

$$\Theta := (\lambda xf.f(xxf))\ (\lambda xf.f(xxf))$$

Then,

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

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

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

Exercise

1. Evaluate $\texttt{sum}\ \mathsf{c}_1$ to its normal form in detail.
2. Define the factorial $n!$ with Church numerals.

Properties of λ-Calculus

Example 20

Suppose M $\texttt{Term}_\lambda$ and $y \notin \texttt{FV}(M)$. 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.

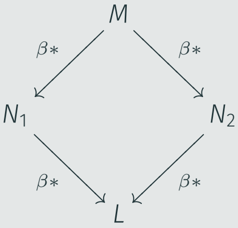
Question:

- Does every path give the same evaluation?

Confluence

Theorem 21 (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 $N_2 \longrightarrow_{\beta*} L$.



No matter which way we choose we can always find a confluent term.

Normal form

Definition 22

M is *in normal form* if there is no N such that $M \longrightarrow_{\beta 1} N$, abbreviated to $M \not\longrightarrow_{\beta 1}$.

Lemma 23

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

Proof.

By induction on the derivation d of $M \longrightarrow_{\beta *} N$.

1. If d is given by (0-step), then $M \longrightarrow_{\beta *} N$ where $M =_\alpha N$ by definition.
2. If d is given by (n+1-step), then $M \longrightarrow_{\beta 1} M'$ and $M' \longrightarrow_{\beta *} N$ are derivable for some M'. By assumption $M \longrightarrow_{\beta 1} N$ is not derivable for any N, so by contradiction the statement follows.

□

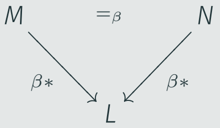
Corollaries of confluence

Corollary 24 (Uniqueness of normal forms)

Let M be a term with $M \longrightarrow_{\beta *} N_1$ and $M \longrightarrow_{\beta *} N_2$ where N_i's are in normal form. Then, $N_1 =_\alpha N_2$.

Corollary 25 (Computationally equal terms have a confluent term)

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



Proof sketch.

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

Homework

1. (2.5%) Show Corollary 24
2. (2.5%) Show Corollary 25.

Appendix: Evaluation strategy

Evaluation strategies i

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,

$$
\begin{aligned}
& \underline{(\lambda x.\,(\lambda y.\,y)\,x)} \quad \underline{(\lambda x.\,(\lambda y.\,y\,y)\,x)} \\
\longrightarrow_{\beta 1} & \underline{(\lambda y.\,y)} \quad \underline{(\lambda x.\,(\lambda y.\,y\,y)\,x)} \\
\longrightarrow_{\beta 1} & \lambda x.\,\underline{(\lambda y.\,y\,y)} \quad \underline{x} \\
\longrightarrow_{\beta 1} & (\lambda x.\,x\,x) \\
\not\longrightarrow_{\beta 1} &
\end{aligned}
$$

Evaluation strategies iii

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

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

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

CBV versus CBN

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

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

Proposition 26 (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

$$\Omega := (\lambda x.\, x\, x)\ (\lambda x.\, x\, x)$$
$$\mathsf{K}_1 := \lambda x\, y.\ x$$

Evaluate

$$\mathsf{K}_1\ z\ \Omega$$

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

Normalisation

Definition 27

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. $\mathsf{K}_1\ z\ \Omega$ is weakly normalising.

Theorem 28

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