

λ -calculus

Valeran MAYTIE

Contents

1	Presentation	2
1.1	Definitions	3
2	Computing with the λ-calculus	3
2.1	Inductive reasoning	4
2.2	Variables and substitutions	4
2.2.1	Free and bound variables	4
2.2.2	Substitution	5
2.2.3	α -conversion	5
2.3	β -reduction	7
2.3.1	Position	8
2.3.2	Inductive reasoning on reduction	8
2.3.3	Reduction sequences	9
2.3.4	Context	10
3	Reduction strategies	11
3.1	Normalization	12
3.2	Reduction strategies	13
3.3	Confluence	14
3.3.1	Parallel reduction	16

1 Presentation

- 1935 (a theory of computable functions)
Alonzo Church, attempt at formalizing computation

Functions:

- maths : $f : A \rightarrow B$ is a set of pairs
- programming : instruction to compute an output

1.1 Definitions

We can define the set of λ -terms (Λ) with a grammar:

$\Lambda := x, y, z \dots$	(variable)
$\mid \lambda. \Lambda$	(functions)
$\mid \Lambda \Lambda$	(application)

The application is left associative: $(l_1 l_2) l_3$.

Notations We can define some notations to simplify the syntax :

Real λ -term	notations
$\lambda x_1. (\dots (\lambda x_n. t) \dots)$	$\lambda x_1 \dots \lambda x_n. t$
$(\dots (t u_1) \dots)$	$t u_1 \dots u_n$
$t u_1 \dots; u_n$	$t \vec{u}$ with $\vec{u} = u_1 \dots u_n$

Example – We can define this λ -term:

- Identity : $I = \lambda x. x$
- Constant generator: $C_c = \lambda x. c$
- Distribution : $\lambda x y z. (x z) (y z)$
- What ? : $\delta = \lambda x. x x$

Curryfication Functions are curryfied (Haskell Curry)

They are no cartesian product in the λ -calculus. So we can define :

- A function: $(x, y) \mapsto t \quad \lambda x y. t$
- An application $f(x, y) \quad f x y$

2 Computing with the λ -calculus

Example, we want to compute $(\lambda x y z. x z (y z)) (\lambda a b. a) t u$

$$\begin{aligned}
 & (\lambda x y z. x z (y z)) (\lambda a b. a) t u \\
 &= (\lambda y z. (\lambda a b. a) z (y z)) t u \\
 &= (\lambda z. (\lambda a b. a) z (t z)) u \\
 &= (\lambda a b. a) u (t u) \\
 &= (\lambda b. u) (t u) \\
 &= u
 \end{aligned}$$

Here are some examples of slightly more subtle calculations:

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

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

We will define the reduction rewrite rule called β -reduction later.

2.1 Inductive reasoning

We can also define Λ with the smallest set such that :

- $\forall x \in \text{Var}, x \in \Lambda$
- $\forall x \in \text{Var}, \forall t \in \Lambda, \lambda x. t \in \Lambda$
- $\forall t_1 t_2, t_1 t_2 \in \Lambda$

We define Λ by induction, so we can write induction function.

For example, we can write f_v the function who compute the number of variable in term t and $f_{@}$ the function who compute the number of application

$$\begin{cases} f_v(x) &= 1 \\ f_v(\lambda x. t) &= f_v(t) \\ f_v(t_1 t_2) &= f_v(t_1) + f_v(t_2) \end{cases} \quad \begin{cases} f_{@}(x) &= 0 \\ f_{@}(\lambda x. t) &= f_{@}(t) \\ f_{@}(t_1 t_2) &= 1 + f_{@}(t_1) + f_{@}(t_2) \end{cases}$$

How to prove that some property $P(t)$ is valid for all λ -terms t ?

1. Prove that $\forall x \in \text{Var}, P(x)$ is valid
2. Prove that $\forall x \in \text{Var}, \forall t, P(t) \Rightarrow P(\lambda x. t)$ is valid
3. Prove that $\forall t_1, t_2, P(t_1) \wedge P(t_2) \Rightarrow P(t_1 t_2)$ is valid

Example – We want to prove $H : \forall t, f_v(t) = 1 + f_{@}(t)$

Proof – We proof H by induction on the term t :

- $t = x$, $f_v(x) = x$ and $f_{@}(x) = 0$, so we have $f_v(x) = 1 + f_{@}(x)$
- $t = \lambda x. t$, we assume that $f_v(t) = 1 + f_{@}(t)$. We calculate $f_v(\lambda x. t) = f_v(t) = 1 + f_{@}(t) = 1 + f_{@}(\lambda x. t)$
- $t = t_1 t_2$, we assume that $f_v(t_1) = 1 + f_{@}(t_1)$ and $f_v(t_2) = 1 + f_{@}(t_2)$. By the calculation $f_v(t_1 t_2) = f_v(t_1) + f_v(t_2) = 1 + f_{@}(t_1) + 1 + f_{@}(t_2) = 1 + f_{@}(t_1 t_2)$

□

2.2 Variables and substitutions

2.2.1 Free and bound variables

To define more calculation operations, we define free variables and bound variables.

Informally, free variables are variables used, but linked to no lambda abstraction. While linked variables are those used and linked to a lambda abstraction.

Definition :

$$\begin{cases} f_v(x) &= \{x\} \\ f_v(\lambda x. t) &= f_v(t) \setminus \{x\} \\ f_v(u v) &= f_v(u) \cup f_v(v) \end{cases} \quad \begin{cases} b_v(x) &= \emptyset \\ b_v(\lambda x. t) &= \{x\} \cup b_v(t) \\ b_v(u v) &= b_v(u) \cup b_v(v) \end{cases}$$

2.2.2 Substitution

The substitution is an operation on λ -term. The aim is to replace the free occurrences of a variable x in term t with another λ -term u . It is noted : $t\{x \leftarrow u\}$. We can define this operation by induction on a λ -term :

$$\begin{aligned} y\{x \leftarrow u\} &= \begin{cases} u & \text{if } x = y \\ y & \text{if } x \neq y \end{cases} \\ (t_1 \ t_2)\{y \leftarrow u\} &= t_1\{y \leftarrow u\} \ t_2\{y \leftarrow u\} \\ (\lambda y. t)\{x \leftarrow u\} &= \begin{cases} \lambda y. t & \text{if } x = y \\ \lambda y. t\{x \leftarrow u\} & \text{if } x \neq y \text{ and } y \notin fv(u) \\ \lambda z. t\{y \leftarrow z\}\{x \leftarrow u\} & \text{if } x \neq y \text{ and } y \in fv(u) \end{cases} \quad z \text{ fresh} \end{aligned}$$

Barendregt's convention The definition of substitution above is not very easy to handle. So we are going to use a convention to greatly simplify the substitution :

no variable name appears both free and bound in any given subterm

Good	Not Good
$\lambda x. x \ (\lambda x. x)$	$\lambda x. (x \ (\lambda y. y))$

The substitution definition become :

$$\begin{aligned} y\{x \leftarrow u\} &= \begin{cases} u & \text{if } x = y \\ y & \text{if } x \neq y \end{cases} \\ (t_1 \ t_2)\{y \leftarrow u\} &= t_1\{y \leftarrow u\} \ t_2\{y \leftarrow u\} \\ (\lambda y. t)\{x \leftarrow u\} &= \lambda y. t\{x \leftarrow u\} \end{aligned}$$

(Un)stability of Barendregt's convention Sometimes during the computation we need to change variables name to preserve the convention :

$$\begin{aligned} &(\lambda x. x \ x) \ (\lambda yz. y \ z) \\ &\rightarrow (\lambda yz. y \ z) \ (\lambda yz. y \ z) \\ &\rightarrow (\lambda yz. (\lambda yz. y \ z)) \end{aligned} \quad \text{Wrong}$$

2.2.3 α -conversion

Two term can be structurally different, but with the same meaning ($\lambda x. x$ and $\lambda y. y$). We can therefore rename linked variables under certain conditions without changing the meaning of a lambda term. We call this operation α -conversion or α -renaming.

α -conversion definition :

$$\lambda x. t =_{\alpha} \lambda y. x \leftarrow y \quad \text{with } x, y \notin bd(t) \text{ and } y \notin fv(t)$$

The α -conversion is a congruence :

$$\begin{aligned} t =_{\alpha} t' &\Rightarrow \lambda x. t =_{\alpha} t' \\ t_1 =_{\alpha} t'_1 &\Rightarrow t_1 \ t_2 =_{\alpha} t'_1 \ t_2 \\ t_2 =_{\alpha} t'_2 &\Rightarrow t_1 \ t_2 =_{\alpha} t_1 \ t'_2 \end{aligned}$$

From now on we assume that any term we work with satisfies Barendregt's convention.

Exercise 2.1 – Make them nice

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

Answer :

- $\lambda x. (\lambda x. x y)(\lambda y. x y) =_{\alpha} \lambda x. (\lambda z. z y)(\lambda w. x w)$
- $\lambda xy. x(\lambda y. (\lambda y. y) y z) =_{\alpha} \lambda xy. x(\lambda a. (\lambda t. t) a z)$

Exercise 2.2 – Compute $(\lambda f. f f) (\lambda a b. b a b)$

Answer :

$$\begin{aligned}
 (\lambda f. f f) (\lambda a b. b a b) &\rightarrow_{\beta} (\lambda a b. b a b) (\lambda a b. b a b) \\
 &\rightarrow_{\beta} \lambda b. b (\lambda a b. b a b) b \\
 &=_{\alpha} \lambda b. b (\lambda x y. y x y) b \\
 &\rightarrow_{\beta} \lambda b. b (\lambda y. y b y)
 \end{aligned}$$

Exercise 2.3 – Prove that $fv(t[x \leftarrow u]) \subseteq (fv(t) \setminus \{x\}) \cup fv(u)$

Answer : Proof by induction on the structure of t

- Case where t is a variable
 - case x : $fv(x[x \leftarrow u]) = fv(x) \subseteq (fv(t) \setminus \{x\}) \cup fv(u)$
 - case $y \neq x$: $fv(y[x \leftarrow u])fv(y) = \{y\}$ and $\{y\}$ is indeed a subset of $(fv(y) \setminus \{x\}) \cup fv(u) = \{y\} \cup fv(u)$
- case where t is an application $t_1 t_2$. Assume $fv(t_1[x \leftarrow u]) \subseteq (fv(t_1) \setminus \{x\}) \cup fv(u)$ and $fv(t_2[x \leftarrow u]) \subseteq (fv(t_2) \setminus \{x\}) \cup fv(u)$. Then

$$\begin{aligned}
 &fv((t_1 t_2)[x \leftarrow u]) \\
 &= fv(t_1[x \leftarrow u] t_2[x \leftarrow u]) && \text{by definition of the substitution} \\
 &= fv(t_1[x \leftarrow u]) \cup fv(t_2[x \leftarrow u]) && \text{by definition of } fv \\
 &\subseteq fv(t_1 \setminus \{x\}) \cup fv(u) \cup fv(t_2 \setminus \{x\}) \cup fv(u) && \text{by induction hypothesis} \\
 &= fv(t_1 \setminus \{x\}) \cup fv(t_2) \setminus \{x\} \cup fv(u) \\
 &= (fv(t_1 \cup fv(t_2)) \setminus \{x\}) \cup fv(u) \\
 &= (fv(t_1 t_2) \setminus \{x\}) \cup fv(u)
 \end{aligned}$$

- Case where t is λ -abstraction $\lambda y. t_0$. Assume $fv(t_0[x \leftarrow u]) \subseteq (fv(t_0) \setminus \{x\}) \cup fv(u)$. Then

$$\begin{aligned}
 &fv(\lambda y. t_0)[x \leftarrow u] \\
 &= fv(\lambda y. t_0[x \leftarrow u]) \\
 &= fv(t_0[x \leftarrow u]) \setminus \{y\} \\
 &\subseteq ((fv(t_0) \setminus \{x\}) \cup fv(u)) \setminus \{y\} && \text{induction hypothesis} \\
 &= (fv(t_0) \setminus \{x\} \setminus \{y\}) \cup fv(u) \setminus \{y\} \\
 &= (fv(t_0) \setminus \{x\} \setminus \{y\}) \cup fv(u) \\
 &= (fv(t_0) \setminus \{y\} \setminus \{x\}) \cup fv(u) \\
 &= (fv(\lambda y. t_0) \setminus \{x\}) \cup fv(u)
 \end{aligned}$$

□

Exercise 2.4 – Prove when $x \notin fv(v)$ and $x \neq y$ then $t\{x \leftarrow u\}\{y \leftarrow v\} = t\{y \leftarrow v\}\{x \leftarrow u\{y \leftarrow v\}\}$
Answer Proof by induction on t

- Case where t is a variable z :

– $z = x$:

$$\begin{aligned} x\{x \leftarrow u\}\{y \leftarrow v\} &= x\{y \leftarrow v\}\{x \leftarrow u\{y \leftarrow v\}\} \\ &= u\{y \leftarrow v\} \end{aligned}$$

– $z = y$:

$$\begin{aligned} y\{x \leftarrow u\}\{y \leftarrow v\} &= y\{y \leftarrow v\}\{x \leftarrow u\{y \leftarrow v\}\} \\ &= v \\ &= v \end{aligned}$$

x is not free in v

– $z \neq y$ and $z \neq x$: $z\{x \leftarrow u\}\{y \leftarrow v\} = z$ and $z\{y \leftarrow v\}\{x \leftarrow u\{y \leftarrow v\}\} = z$

- $t = t_1 t_2$:

$$\begin{aligned} (t_1 t_2)\{x \leftarrow u\}\{y \leftarrow v\} &= t_1\{x \leftarrow u\}\{y \leftarrow v\} t_2\{x \leftarrow u\}\{y \leftarrow v\} \\ &= t_1\{y \leftarrow v\}\{x \leftarrow u\{y \leftarrow v\}\} t_2\{y \leftarrow v\}\{x \leftarrow u\{y \leftarrow v\}\} \quad \text{induction hypothesis} \\ &= (t_1 t_2)\{y \leftarrow v\}\{x \leftarrow u\{y \leftarrow v\}\} \end{aligned}$$

- $t = \lambda z. t_0$

$$\begin{aligned} (\lambda z. t_0)\{x \leftarrow u\}\{y \leftarrow v\} &= \lambda z. t_0\{x \leftarrow u\}\{y \leftarrow v\} \\ &= \lambda z. t_0\{y \leftarrow v\}\{x \leftarrow u\{y \leftarrow v\}\} \quad \text{induction hypothesis} \\ &= (\lambda z. t_0)\{y \leftarrow v\}\{x \leftarrow u\{y \leftarrow v\}\} \end{aligned}$$

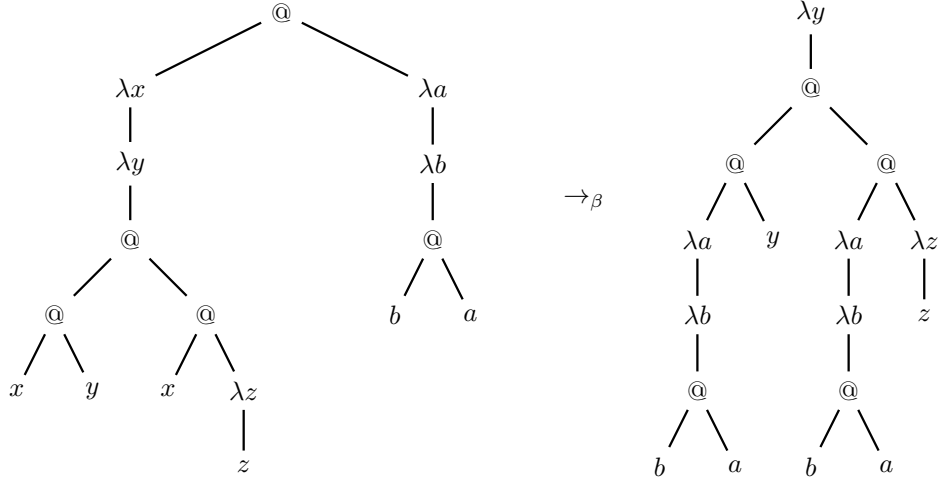
□

2.3 β -reduction

The β -reduction is a rewrite rule who apply an argument to a function. We need to have a λ -term on the form $(\lambda x. t) u$. This form is called a (β -redex). The computation rule is :

$$(\lambda x. t) u \rightarrow_{\beta} t\{x \leftarrow u\}$$

We can draw :



A β -reduction can be done anywhere in a term. We must therefore manage cases where a reduction is made after a lambda abstraction or in the left or right branch of an application. So we're going to describe our reduction using inference rule :

$$\begin{array}{c}
 \overline{(\lambda x. t) u \rightarrow_{\beta} t\{x \leftarrow u\}} \\
 \\
 \frac{t \rightarrow_{\beta} t'}{t u \rightarrow_{\beta} t' u} \qquad \frac{u \rightarrow_{\beta} u'}{t u \rightarrow_{\beta} t u'} \\
 \\
 \frac{t \rightarrow_{\beta} u'}{\lambda x. t \rightarrow_{\beta} \lambda x. t'}
 \end{array}$$

2.3.1 Position

We can locate the beta reduction by encoding the position of the reduction operation, we can rewrite the resets like this :

$$\begin{array}{c}
 \overline{(\lambda x. t) u \xrightarrow{\epsilon}_{\beta} t\{x \leftarrow u\}} \\
 \\
 \frac{t \xrightarrow{p}_{\beta} t'}{t u \xrightarrow{1 \cdot p}_{\beta} t' u} \qquad \frac{u \xrightarrow{p}_{\beta} u'}{t u \xrightarrow{2 \cdot p}_{\beta} t u'} \\
 \\
 \frac{t \xrightarrow{p}_{\beta} u'}{\lambda x. t \xrightarrow{0 \cdot p}_{\beta} \lambda x. t'}
 \end{array}$$

2.3.2 Inductive reasoning on reduction

Since the β -reduction has been defined using inference rules, we can resonate by recurrence on the reduction. To prove a formula of the form :

$$\forall t, t', t \rightarrow_{\beta} t' \Rightarrow P(t, t')$$

we need to check the following four points:

- $P((\lambda x. t)u, t\{x \rightarrow u\})$ for any x, y and u
- $P(t\ u, t'\ u)$ for any t, t' and u such that $P(t, t')$
- $P(t\ u, t\ u')$ for any t, u and u' such that $P(u, u')$
- $P(\lambda x. t, \lambda x. t')$ for any x, t and u' such that $P(t, t')$

Example – We want to prove :

$$\forall t\ t', t \rightarrow t' \Rightarrow fv(t') \subseteq fv(t)$$

Proof by induction on the derivation of $t \rightarrow t'$.

- $(\lambda x. t)\ u \rightarrow t\{x \leftarrow u\}$. We already proved: $fv(t\{x \leftarrow y\}) \subseteq (fv(t) \setminus \{x\}) \cup fv(y)$. Moreover, we have

$$\begin{aligned} fv((\lambda x. t)\ u) &= fv(\lambda x. t) \cup fv(u) \\ &= (fv(t) \setminus \{x\}) \cup fv(u) \end{aligned}$$

- $t\ u' \rightarrow t\ u$ with $t \rightarrow t'$. Then

$$\begin{aligned} fv(t'\ u) &= fv(t') \cup fv(u) && \text{by definition} \\ &\subseteq fv(t) \cup fv(u) && \text{by induction hypothesis} \\ &= fv(t\ u) && \text{by definition} \end{aligned}$$

- $t\ u' \rightarrow t\ u'$ with $u \rightarrow u'$ similar.
- $\lambda x. t \rightarrow \lambda x. t'$ with $t \rightarrow t'$. Then

$$\begin{aligned} fv(\lambda x. t') &= fv(t') \setminus \{x\} && \text{by definition} \\ &\subseteq fv(t) \setminus \{x\} && \text{by induction hypothesis} \\ &= fv(\lambda x. t) && \text{by definition} \end{aligned}$$

□

2.3.3 Reduction sequences

Other reduction can be set above the beta reduction :

- \rightarrow_β one step
- \rightarrow_β^* reflexive transitive closure: 0, 1 or many steps
- \leftrightarrow symmetric closure : one step, forward or backward.
- $=_\beta$ reflexive, symmetric, transitive closure (equivalence)

2.3.4 Context

We can also formalize our reduction with a context that is intuitively a lambda term with a hole. So we have two steps, replace a variable with a hole and replace the hole with a lambda term. So there is no need for substitution.

We define a context with the following grammar:

$\mathcal{C} := \square$	(hole)
$ x, y, z \dots$	(variable)
$ \lambda x. \mathcal{C}$	(functions)
$ \mathcal{C} \mathcal{C}$	(application)

The operation $\mathcal{C}[u]$ is the result of filling the hole of \mathcal{C} with the term u .

Exercise 2.5 – Here are some decompositions of $\lambda x.(x \lambda y.xy)$ into a context and a term $\mathcal{C}[u]$

\mathcal{C}	\square	$\lambda x.\square$	$\lambda x.(\square \lambda y.xy)$	$\lambda x.(x \square)$	\dots
u	$\lambda x.(x \lambda y.xy)$	$x \lambda y.xy$	x	$\lambda y. x y$	\dots

What are the other possible decompositions ?

We already showed that

$$(\lambda x.x ((\lambda y.zy)x))z \rightarrow (\lambda x.x (zx)) z$$

What are the context and the redex associated to this reduction ?

Answer Other decompositions of $\lambda x.(x \lambda y.xy)$

\mathcal{C}	$\lambda x.(x \lambda y.\square)$	$\lambda x.(x \lambda y.\square y)$	$\lambda x.(x \lambda y.x \square)$
u	xy	x	y

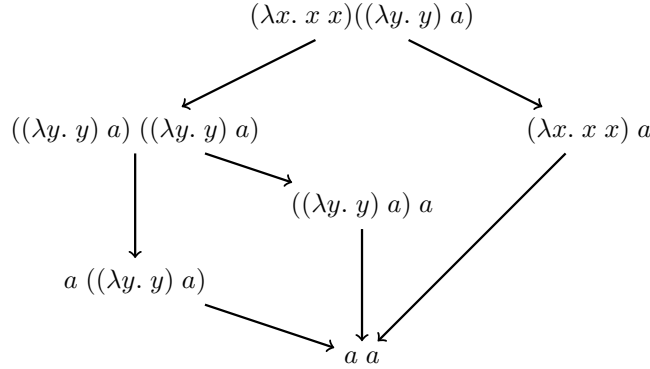
Decomposition of the reduction:

$$\mathcal{C}[(\lambda y.xy)x] \rightarrow \mathcal{C}[zx]$$

with $\mathcal{C} = (\lambda x.x \square) z$

3 Reduction strategies

There are several possibilities when reducing terms. We can draw these possibilities in the form of a graph like this one:



This raises the questions :

- Are some paths better than others ?
- Is there always a result in the ? Is it unique ?

3.1 Normalization

A *normal form* is a term that cannot be reduced anymore, formally : $N(t) = \neg \exists t', t \rightarrow t'$.

Example – .

normal form	not normal form
x	$(\lambda x.x) y$
$\lambda x.xy$	$x((\lambda y.y)(\lambda z.zx))$
$x(\lambda y.y)(\lambda z.zx)$	

If $t \rightarrow^* t'$ and t' is normal, the term t' is said to be a normal form of t . This defines our informal notion of a result of a term.

Some terms do not have a normal form :

$$\begin{aligned}
 \Omega &= \delta \delta \\
 &= (\lambda x.xx) (\lambda x.xx) \\
 &\rightarrow (xx) \{ \lambda x.xx \} \\
 &= x \{ \lambda x.xx \} x \{ \lambda x.xx \} \\
 &= (\lambda x.xx) (\lambda x.xx) \\
 &= \Omega
 \end{aligned}$$

Normalization properties A term t is :

- *strongly normalizing* if every reduction sequence starting from t eventually reaches a normal form :

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

- *weakly normalizing*, or normalizable, if there is at least one reduction sequence starting from t and reaching a normal form :

$$(\lambda xy.y) ((\lambda z.zz) (\lambda z.zz))$$

3.2 Reduction strategies

The purpose of a reduction strategy is to determine a redex reduction order within a term. We have two well-known reduction orders :

- *Normal order* : reduce the most external redex first. Apply functions without reducing the arguments
- *Applicative order* : reduce the most internal redex first. Normalize the arguments before reducing the function application itself.

Exercise 3.1 – normal order vs. applicative order.

Compare normal order reduction and applicative order reduction of the following terms :

1. $(\lambda xy.x) z \Omega$
2. $(\lambda x.xx)((\lambda y.y) z)$
3. $(\lambda x.x(\lambda y.y))(\lambda z.(\lambda a.aa)(z b))$

In each case: does another order allow shorter sequences ?

Answer

1. Normal order

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

Applicative order

$$\begin{aligned} & (\lambda xy.x) z \Omega \\ & \rightarrow (\lambda xy.x) \Omega \\ & \rightarrow \dots \end{aligned}$$

Normal order reduction is as short as possible.

2. Normal order

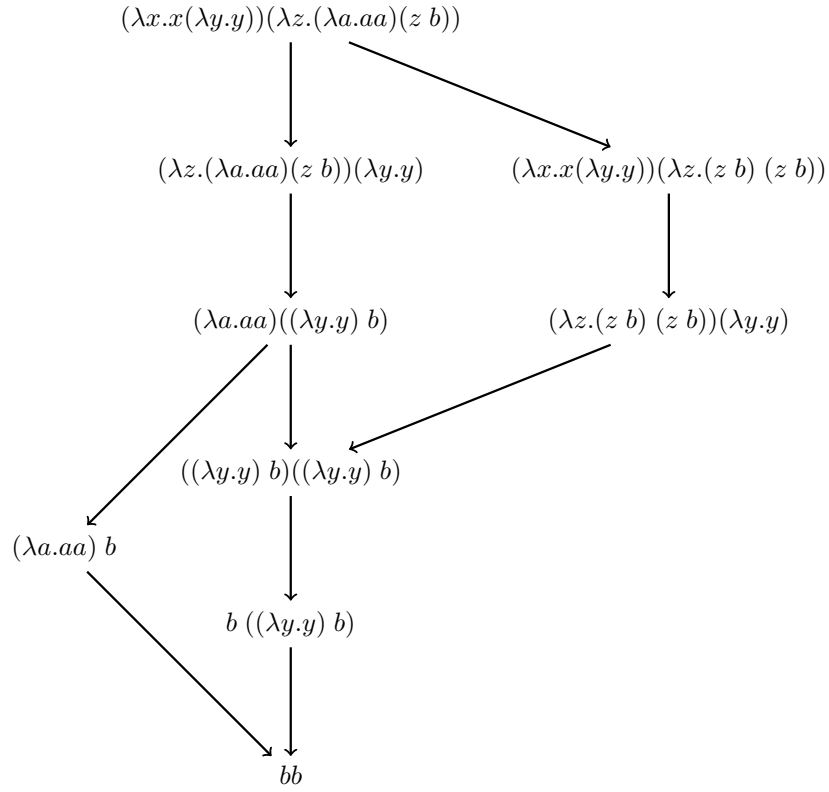
$$\begin{aligned} & (\lambda x.xx) ((\lambda y.y) z) \\ & \rightarrow ((\lambda y.y) z) ((\lambda y.y) z) \\ & \rightarrow z((\lambda y.y) z) \\ & \rightarrow zz \end{aligned}$$

Applicative order

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

Applicative order reduction is as short as possible.

3. Reduction graph :



The middle path is the normal order strategy, the right path is the applicative order and the left path is the shortest reduction.

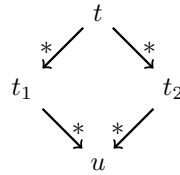
Normal order property If a term t does have a normal form the normal order reduction reaches this normal form.

3.3 Confluence

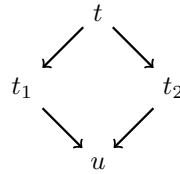
Confluence is a very useful concept to prove the uniqueness of a result of a calculation rule like beta-reduction. It says that if a term can be rewritten in more than one way, then there is always a way to revert to a common term

Formally, the confluence is written like this: $\forall t_1 t_2, t \rightarrow^* t_1 \wedge t \rightarrow^* t_2 \Rightarrow \exists u, t_1 \rightarrow^* u \wedge t_2 \rightarrow^* u$

We can draw the following diagram:



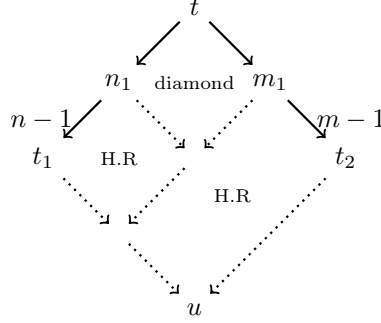
A rewrite rule can have the *diamond property*. In one rewriting step, it can always fall back on the same term in the second rewriting step. $\forall t_1 t_2, t \rightarrow t_1 \wedge t \rightarrow t_2 \Rightarrow \exists u, t_1 \rightarrow u \wedge t_2 \rightarrow u$



Lemma 3.1 – The diamond property implies the confluence.

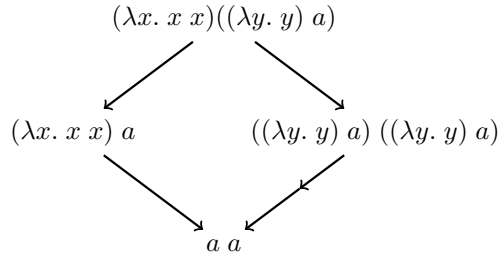
proof We can prove that by induction on $n + m$ with n the number of steps of $t \rightarrow t_1$ and m the number of steps of $t \rightarrow t_2$.

- $n + m = 0$, then $t = t_1 = t_2$, so $t \not\rightarrow t_1$ and $t \not\rightarrow t_2$ the property is true.
- $n + m + 1$ we assume that the diamond property implies the confluence on the condition that t reduce to t_1 less that n steps and t reduce to t_2 less that m steps.

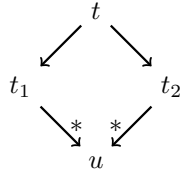


□

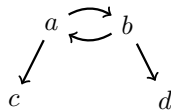
The β -reduction does not have the diamond property :



We can proof that the β -reduction is locally confluent, which is: $\forall t_1 t_2, t \rightarrow t_1 \wedge t \rightarrow t_2 \Rightarrow \exists u, t_1 \rightarrow^* u \wedge t_2 \rightarrow^* u$. It represents with this diagram :



But the local confluence does not imply the confluence, the Curry's counter example. This diagram is locally confluent, but it is not confluent.



The proof of confluence is therefore much more complicated than expected. We will present two possible proofs.

3.3.1 Parallel reduction

Parallel reduction proof consists of defining a relation \Rightarrow_β that is “between” \rightarrow_β and \rightarrow_β^* . This relation need to have the diamond property. Because thanks to this we can prove the confluence of this new relation.

We’re going to start by defining the parallel reduction :

$$\begin{array}{c}
\frac{}{x \Rightarrow_\beta x} \text{ID} \\
\\
\frac{t \Rightarrow_\beta t'}{\lambda x.t \Rightarrow_\beta \lambda x.t'} \text{ABS} \qquad \frac{t_1 \Rightarrow_\beta t'_1 \quad t_2 \Rightarrow_\beta t'_2}{t_1 t_2 \Rightarrow_\beta t'_1 t'_2} \text{APP} \\
\\
\frac{t \Rightarrow_\beta t' \quad u \Rightarrow_\beta u'}{(\lambda x.t)u \Rightarrow_\beta t'\{x \leftarrow u'\}} \text{RED}
\end{array}$$

Example – $(\lambda x.((\lambda y.y) (\lambda z.z)) x) ((\lambda w.w) a)$

$$\frac{\frac{\frac{y \Rightarrow_\beta y}{(\lambda y.y) (\lambda z.z) \Rightarrow_\beta \lambda z.z} \quad \frac{z \Rightarrow_\beta z}{\lambda z.z \Rightarrow_\beta \lambda z.z}}{(\lambda x.((\lambda y.y) (\lambda z.z)) x) \Rightarrow_\beta (\lambda z.z) x} \quad \frac{\frac{w \Rightarrow_\beta w}{(\lambda w.w) a \Rightarrow_\beta a} \quad \frac{a \Rightarrow_\beta a}{(\lambda w.w) a \Rightarrow_\beta a}}{(\lambda w.w) a \Rightarrow_\beta (\lambda z.z) a} \quad \frac{x \Rightarrow_\beta x}{(\lambda x.((\lambda y.y) (\lambda z.z)) x) \Rightarrow_\beta (\lambda z.z) x}}{(\lambda x.((\lambda y.y) (\lambda z.z)) x) ((\lambda w.w) a) \Rightarrow_\beta (\lambda z.z) a}$$

Lemma 3.2 – $\forall t, t \Rightarrow_\beta t$

Proof: We prove by induction on t :

- $t = x$ by definition of \Rightarrow_β we have $x \Rightarrow_\beta x$.
- $t = t_1 t_2$. We have this induction hypothesis $t_1 \Rightarrow_\beta t_1$ and $t_2 \Rightarrow_\beta t_2$. So we have $t_1 t_2 \Rightarrow_\beta t_1 t_2$
- $t = \lambda x.t_0$. We have this induction hypothesis $t_0 \Rightarrow_\beta t_0$. So we have $\lambda x.t_0 \Rightarrow_\beta \lambda x.t_0$.

□

Lemma 3.3 – $\forall t t', t \rightarrow_\beta t' \Rightarrow t \Rightarrow_\beta t' \quad \rightarrow_\beta \subseteq \Rightarrow_\beta$

Proof: We prove by induction on \rightarrow_β

- Case $(\lambda x.t) u \rightarrow_\beta t\{x \leftarrow u\}$:

$$\text{LEMMA-3.2} \frac{\frac{}{t \Rightarrow_\beta t} \quad \frac{}{u \Rightarrow_\beta u} \text{LEMMA-3.2}}{(\lambda x.t) u \Rightarrow_\beta t\{x \leftarrow u\}}$$

- Case $t_1 t_2 \rightarrow_\beta t'_1 t_2$ with $t_1 \rightarrow_\beta t'_1$. With the induction hypothesis $t_1 \Rightarrow_\beta t'_1$

$$\text{H.R} \frac{\frac{}{t_1 \Rightarrow_\beta t'_1} \quad \frac{}{t_2 \Rightarrow_\beta t_2} \text{LEMMA-3.2}}{t_1 t_2 \Rightarrow_\beta t'_1 t_2}$$

- The other case are similar.

Lemma 3.4 – $\forall t t', t \Rightarrow_\beta t' \Rightarrow t \rightarrow_\beta^* t' \quad \Rightarrow_\beta \subseteq \rightarrow_\beta^*$

Proof: by induction on \Rightarrow_β

- Case $x \Rightarrow_\beta x$. We have $x \rightarrow_\beta^0 x$

- Case $\lambda x.u \Rightarrow_\beta \lambda x.u'$ with $t \Rightarrow_\beta t'$. We assume that $t \rightarrow_\beta^* t'$ (induction hypothesis). We can easily prove by recurrence on the length of \rightarrow_β^* , that we have well $\lambda x.t \rightarrow_\beta^* \lambda x.t'$
- The application rule is similar.
- Case $(\lambda x.t) u \Rightarrow_\beta t'\{x \leftarrow u'\}$ with $t \Rightarrow_\beta t'$ and $u \Rightarrow_\beta u'$. Then we have $(\lambda x.t)u \rightarrow_\beta^* (\lambda x.t')u$ by induction hypothesis on t , $(\lambda x.t')u' \rightarrow_\beta^* (\lambda x.t')u'$ by induction hypothesis on u . And finally we have $(\lambda x.t')u' \rightarrow_\beta t'\{x \leftarrow u'\}$

□

method of Tait and Martin-Löf We want to proof that a relation has the diamond property

Lemma 3.5 – If a relation \rightarrow has the diamond property then \rightarrow^* has the diamond property to.

Proof : We suppose that \rightarrow has the diamond property [TODO] □

Lemma 3.6 – If two relation \rightarrow and \Rightarrow are such that $\rightarrow \subseteq \Rightarrow \subseteq \rightarrow^*$ then $\Rightarrow^* = \rightarrow^*$.

Proof: From $\rightarrow \subseteq \Rightarrow \subseteq \rightarrow^*$ we deduce $\rightarrow^* \subseteq \Rightarrow^* \subseteq \rightarrow^{**}$. However we have $\rightarrow^* = \rightarrow^{**}$. Then $\rightarrow^* \subseteq \Rightarrow^* \subseteq \rightarrow^*$ so we have $\Rightarrow^* = \rightarrow^*$.

□

To proof that \rightarrow^* has the diamond property we just need to show that \Rightarrow_β has the diamond property. To do this we need the following lemma :

Lemma 3.7 – $a \Rightarrow_\beta a' \wedge b \Rightarrow_\beta b' \Rightarrow a\{x \leftarrow b\} \Rightarrow_\beta a'\{x \leftarrow b\}$

Proof: By induction one the derivation $a \Rightarrow_\beta a'$.

- Case $y \Rightarrow_\beta y$
 - If $x = y$, then $x\{x \leftarrow b\} = b \Rightarrow_\beta b' = x\{x \leftarrow b'\}$
 - If $x \neq y$, then $y\{x \leftarrow b\} = y \Rightarrow_\beta y = y\{x \leftarrow b'\}$
- Case $\lambda y.a_0 \Rightarrow_\beta \lambda y.a'_0$ with $a_0 \Rightarrow_\beta a'_0$
Then $(\lambda y.a_0)\{x \leftarrow b\} = \lambda y.a_0\{x \leftarrow b\}$. By induction hypothesis we have $a_0\{x \leftarrow b\} \Rightarrow_\beta a'_0\{x \leftarrow b'\}$.
Therefore, we have $\lambda y.a_0\{x \leftarrow b\} \Rightarrow_\beta \lambda y.a'_0\{x \leftarrow b'\} = (\lambda y.a'_0)\{x \leftarrow b'\}$
- Case $a_1 a_2 \Rightarrow_\beta a'_1 a'_2$ with $a_1 \Rightarrow_\beta a'_1$ and $a_2 \Rightarrow_\beta a'_2$.
It is similar to the case above.
- Case $(\lambda x.a_1) a_2 \Rightarrow_\beta (\lambda x.a'_1) a'_2$ with $a_1 \Rightarrow_\beta a'_1$ and $a_2 \Rightarrow_\beta a'_2$.
Then $((\lambda y.a_1)a_2)\{x \leftarrow b\} = (\lambda y.a_1\{x \leftarrow b\})a_2\{x \leftarrow b\}$.
By the induction hypothesis we have $a_1\{x \leftarrow b\} \Rightarrow_\beta a'_1\{x \leftarrow b'\}$ and $a_2\{x \leftarrow b\} \Rightarrow_\beta a'_2\{x \leftarrow b'\}$.
Therefore $(\lambda y.a_1\{x \leftarrow b\})(a_2\{x \leftarrow b\}) \Rightarrow_\beta (a'_1\{x \leftarrow b'\})(a'_2\{x \leftarrow b'\})$. Thanks to the substitution lemma-2.4

□

Lemma 3.8 – \Rightarrow_β has the diamond property.

$\forall t s r, t \Rightarrow_\beta s \wedge t \Rightarrow_\beta r \Rightarrow \exists u, s \Rightarrow_\beta u \wedge r \Rightarrow_\beta u$.

Proof: By induction on the derivation of $t \Rightarrow_\beta re$.

- Case $x \Rightarrow_\beta x$. Then $s = x$ so we can take $u = x$.
- Case $\lambda x.t_0 \Rightarrow_\beta \lambda x.r_0$ with $t_0 \Rightarrow_\beta r_0$. Then $s = \lambda x.s_0$ with $t_0 \Rightarrow_\beta s_0$.
By induction hypothesis we have u_0 such that $s_0 \Rightarrow_\beta u_0$ and $r_0 \Rightarrow_\beta u_0$.
Therefore, $\lambda x.s_0 \Rightarrow_\beta \lambda x.u_0$ And $\lambda x.r_0 \Rightarrow_\beta \lambda x.u_0$
- Case $t_1 t_2 \Rightarrow_\beta r_1 r_2$ with $t_1 \Rightarrow_\beta r_1$ and $t_2 \Rightarrow_\beta r_2$. Two case for $t_1 t_2 \Rightarrow_\beta s_0$:
 - if $s = s_1 s_2$ with $t_1 \Rightarrow_\beta s_1$ and $t_2 \Rightarrow_\beta s_2$ by induction hypothesis

□