λ-calculus

Thibaut BALABONSKI

noted by Valeran MAYTIE

Contents

1 Presentation

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

Functions:

- maths : $f : A \to B$ 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:

$$\begin{aligned} \Lambda := & \ x, y, z... && \text{(variable)} \\ & \mid \lambda.\Lambda && \text{(functions)} \\ & \mid \Lambda\ \Lambda && \text{(application)} \end{aligned}$$

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
$\lambda x_1.(\ldots(\lambda x_n.t)\ldots)$	$\lambda x_1 \ldots \lambda x_n.t$
$(\ldots(t\ u_1)\ldots)$	$t\ u_1 \ldots u_n$
$t\ u_1\ \ldots_;\ u_n$	$t\ \vec{u}$ with $\vec{u} = u_1\ \ldots\ u_n$

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:	$(x, y) \mapsto t$	$\lambda x\ y.\ t$
• An application	$f(x, y)$	$f\ x\ y$

2 Computing with the λ-calculus

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

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

Here are some examples of slightly more subtle calculations:

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

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 :

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

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

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

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

1. Prove that $\forall x \in \mathrm{Var}, P(x)$ is valid
2. Prove that $\forall x \in \mathrm{Var}, \forall t, P(t) \Rightarrow P(\lambda x.\ t)$ 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 $f_v(t_1\ t_2) = f_v(t_1) + f_v(t_2) = 1 + f_@(t_1) + 1 + f_@(t_2) = 1 + f_@(t_1\ t_2)$

$\square$

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 :

$$\begin{cases} fv(x) & = \{x\} \\ fv(\lambda x.t) & = fv(t) \backslash \{x\} \\ fv(u\ v) & = fv(u) \cup fv(v) \end{cases} \qquad \begin{cases} bv(x) & = \emptyset \\ bv(\lambda x.t) & = \{x\} \cup bv(t) \\ bv(u\ v) & = bv(u) \cup bv(v) \end{cases}$$

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 :

$$y\{x \leftarrow y\} = \begin{cases} u & \text{if } x = y \\ y & \text{if } x \neq y \end{cases}$$

$$(t_1\ t_2)\{y \leftarrow u\} = t_1\{y \leftarrow u\}\ t_2\{y \leftarrow u\}$$

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

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 :

$$y\{x \leftarrow u\} = \begin{cases} u & \text{if } x = y \\ y & \text{if } x \neq y \end{cases}$$

$$(t_1\ t_2)\{y \leftarrow u\} = t_1\{y \leftarrow u\}\ t_2\{y \leftarrow u\}$$

$$(\lambda y.\ t)\{x \leftarrow u\} = \lambda y.\ t\{x \leftarrow u\}$$

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

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

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 :

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

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 $(\lambda f.\ f\ f)\ (\lambda ab.b\ a\ b)$
Answer :

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

Exercise 2.3 – Prove that $fv(t[x \leftarrow u]) \subseteq (fv(t) \backslash \{x\}) \cup fv(u)$
Answer : Proof by induction on the structure of t

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

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

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

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

□

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:
  - $z = x$:

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

  - $z = y$:

$$
\begin{aligned}
& y\{x \leftarrow u\}\{y \leftarrow v\} \\
& = y\{y \leftarrow v\} \\
& = v
\end{aligned}
\qquad\qquad
\begin{aligned}
& y\{y \leftarrow v\}\{x \leftarrow u\{y \leftarrow v\}\} \\
& = v\{x \leftarrow u\{y \leftarrow v\}\} \\
& = v && x \text{ is not free in } v
\end{aligned}
$$

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

- $t = t_1\ t_2$:

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

- $t = \lambda z.\ t_0$

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

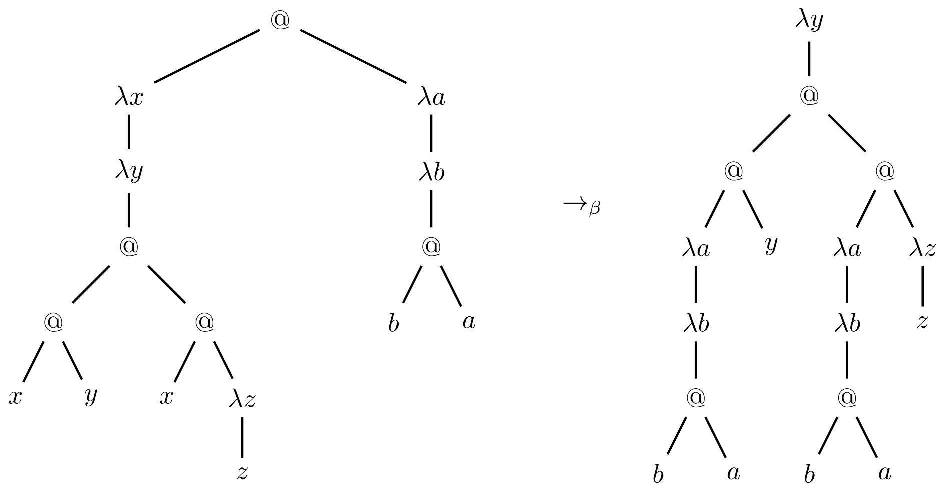
□

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 :

$$(\lambda x.\, t)\; u \to_\beta t\{x \leftarrow u\}$$

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 :

$$\frac{}{(\lambda x.\, t)\; u \quad \to_\beta \quad t\{x \leftarrow u\}}$$

$$\frac{t \quad \to_\beta \quad t'}{t\; u \quad \to_\beta \quad t'\; u} \qquad\qquad \frac{u \quad \to_\beta \quad u'}{t\; u \quad \to_\beta \quad t\; u'}$$

$$\frac{t \quad \to_\beta \quad u'}{\lambda x.\, t \quad \to_\beta \quad \lambda x.\, t'}$$

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 :

$$\frac{}{(\lambda x.\, t)\; u \quad \xrightarrow{\epsilon}_\beta \quad t\{x \leftarrow u\}}$$

$$\frac{t \quad \xrightarrow{p}_\beta \quad t'}{t\; u \quad \xrightarrow{1\cdot p}_\beta \quad t'\; u} \qquad\qquad \frac{u \quad \xrightarrow{p}_\beta \quad u'}{t\; u \quad \xrightarrow{2\cdot p}_\beta \quad t\; u'}$$

$$\frac{t \quad \xrightarrow{p}_\beta \quad u'}{\lambda x.\, t \quad \xrightarrow{0\cdot p}_\beta \quad \lambda x.\, t'}$$

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: $fv(t\{x \leftarrow y\} \subseteq (fv(t) \backslash \{x\} \cup fv(u)$. Moreover, we have

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

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

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

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

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

□

2.3.3 Reduction sequences

Other reduction can be set above the beta reduction :

- $\rightarrow_\beta$ one step
- $\rightarrow_\beta^* / \twoheadrightarrow_\beta$ 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:

$$
\begin{aligned}
\mathcal{C} := \;& \Box && \text{(hole)} \\
& \mid x, y, z ... && \text{(variable)} \\
& \mid \lambda x.\, \mathcal{C} && \text{(functions)} \\
& \mid \mathcal{C}\; \mathcal{C} && \text{(application)}
\end{aligned}
$$

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 $\mathcal{C}[u]$

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

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

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

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

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

Decomposition of the reduction:

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

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

3 Reduction strategies

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

$$
\begin{tikzcd}
 & (\lambda x.\ x\ x)((\lambda y.\ y)\ a) \arrow[dl] \arrow[dr] & \\
((\lambda y.\ y)\ a)\ ((\lambda y.\ y)\ a) \arrow[dr] \arrow[dd] & & (\lambda x.\ x\ x)\ a \arrow[dddl] \\
 & ((\lambda y.\ y)\ a)\ a \arrow[dd] & \\
a\ ((\lambda y.\ y)\ a) \arrow[dr] & & \\
 & a\ a &
\end{tikzcd}
$$

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 : $N(t) = \neg\exists t', t \to t'$.

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 $t \to^* t'$ 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 :

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

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

$$\begin{array}{l}(\lambda xy.x)\ z\ \Omega\\ \to (\lambda y.z)\ \Omega\\ \to z\end{array}$$

Applicative order

$$\begin{array}{l}(\lambda xy.x)\ z\ \Omega\\ \to (\lambda xy.x)\ \Omega\\ \to \ldots\end{array}$$

Normal order reduction is as short as possible.

2. Normal order

$$\begin{array}{l}(\lambda x.xx)\ ((\lambda y.y)\ z)\\ \to ((\lambda y.y)\ z)\ ((\lambda y.y)\ z)\\ \to z((\lambda y.y)\ z)\\ \to zz\end{array}$$

Applicative order

$$\begin{array}{l}(\lambda x.xx)\ ((\lambda y.y)\ z)\\ \to (\lambda x.xx)\ z\\ \to zz\end{array}$$

Applicative order reduction is as short as possible.

3. Reduction graph :

$$
\begin{tikzcd}
 & (\lambda x.x(\lambda y.y))(\lambda z.(\lambda a.aa)(z\ b)) \arrow[d] \arrow[dr] & \\
 & (\lambda z.(\lambda a.aa)(z\ b))(\lambda y.y) \arrow[d] & (\lambda x.x(\lambda y.y))(\lambda z.(z\ b)\ (z\ b)) \arrow[d] \\
 & (\lambda a.aa)((\lambda y.y)\ b) \arrow[d] \arrow[ddl] & (\lambda z.(z\ b)\ (z\ b))(\lambda y.y) \arrow[dl] \\
 & ((\lambda y.y)\ b)((\lambda y.y)\ b) \arrow[dd] & \\
(\lambda a.aa)\ b \arrow[ddr] & & \\
 & b\ ((\lambda y.y)\ b) \arrow[d] & \\
 & bb &
\end{tikzcd}
$$

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: $\forall t\ t_1\ t_2, t \to^* t_1 \wedge t \to^* t_2 \Rightarrow \exists u, t_1 \to^* u \wedge t_2 \to^* u$

We can draw the following diagram:

$$
\begin{tikzcd}
 & t \arrow[dl, ->>] \arrow[dr, ->>] & \\
t_1 \arrow[dr, ->>] & & t_2 \arrow[dl, ->>] \\
 & u &
\end{tikzcd}
$$

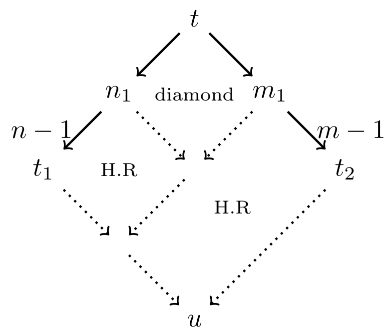
A rewrite rule can have the *diamond property*. In one rewriting step, it can always fall back on the same term in the second rewriting step. $\forall t\ t_1\ t_2, t \to t_1 \wedge t_2 \Rightarrow \exists u, t_1 \to u \wedge t_2 \to u$

$$
\begin{tikzcd}
 & t \arrow[dl] \arrow[dr] & \\
t_1 \arrow[dr] & & t_2 \arrow[dl] \\
 & u &
\end{tikzcd}
$$

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 $t \to t_1$ and m the number of steps of $t \to t_2$.

- $n+m=0$, then $t = t_1 = t_2$, so $t \not\to t_1$ and $t \not\to t_2$ 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 :

$$
\begin{tikzcd}
 & (\lambda x.\ x\ x)((\lambda y.\ y)\ a) \arrow[dl] \arrow[dr] & \\
(\lambda x.\ x\ x)\ a \arrow[dr] & & ((\lambda y.\ y)\ a)\ ((\lambda y.\ y)\ a) \arrow[dl, "*"] \\
 & a\ a &
\end{tikzcd}
$$

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

$$
\begin{tikzcd}
 & t \arrow[dl] \arrow[dr] & \\
t_1 \arrow[dr, "*"] & & t_2 \arrow[dl, "*"'] \\
 & u &
\end{tikzcd}
$$

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

$$
\begin{tikzcd}
a \arrow[r, bend left] \arrow[d] & b \arrow[l, bend left] \arrow[d] \\
c & d
\end{tikzcd}
$$

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 $\rightrightarrows_\beta$ that is "between" $\to_\beta$ and $\to_\beta^*$. 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 :

$$\frac{}{x \quad \rightrightarrows_\beta \quad x}\ \textsc{Id}$$

$$\frac{t \quad \rightrightarrows_\beta \quad t'}{\lambda x.t \quad \rightrightarrows_\beta \quad \lambda x.t}\ \textsc{Abs} \qquad \frac{t_1 \rightrightarrows_\beta t_1' \qquad t_2 \rightrightarrows_\beta t_2'}{t_1\ t_2 \quad \rightrightarrows_\beta \quad t_1'\ t_2'}\ \textsc{App}$$

$$\frac{t \rightrightarrows_\beta t' \qquad u \rightrightarrows_\beta u'}{(\lambda x.t)u \rightrightarrows_\beta t'\{x \leftarrow u'\}}\ \textsc{Red}$$

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

$$\dfrac{\dfrac{\dfrac{\dfrac{}{y \rightrightarrows_\beta y} \qquad \dfrac{\dfrac{}{z \rightrightarrows_\beta z}}{\lambda z.z \rightrightarrows_\beta \lambda z.z}}{(\lambda y.y)\ (\lambda z.z) \rightrightarrows_\beta \lambda z.z} \qquad \dfrac{}{x \rightrightarrows_\beta x}}{(\lambda x.((\lambda y.y)\ (\lambda z.z))\ x) \rightrightarrows_\beta (\lambda z.z)\ x} \qquad \dfrac{\dfrac{}{w \rightrightarrows_\beta w} \qquad \dfrac{}{a \rightrightarrows_\beta a}}{(\lambda w.w)a \rightrightarrows_\beta a}}{(\lambda x.((\lambda y.y)\ (\lambda z.z))\ x)\ ((\lambda w.w)\ a) \rightrightarrows_\beta (\lambda z.z)\ a}$$

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

Proof: We prove by induction on t:

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

□

Lemma 3.3 – $\forall tt', t \to_\beta t' \Rightarrow t \rightrightarrows_\beta t' \qquad \to_\beta \subseteq \rightrightarrows_\beta$

Proof: We prove by induction on $\to_\beta$

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

$$\dfrac{\textsc{Lemma-3.2}\ \dfrac{}{t \rightrightarrows_\beta t} \qquad \dfrac{}{u \rightrightarrows_\beta u}\ \textsc{Lemma-3.2}}{(\lambda x.t)\ u \rightrightarrows_\beta t\{x \leftarrow u\}}$$

- Case $t_1\ t_2 \to_\beta t_1'\ t_2$ with $t_1 \to_\beta t_2$. With the induction hypothesis $t_1 \rightrightarrows_\beta t_1'$

$$\dfrac{\text{H.R}\ \dfrac{}{t_1 \rightrightarrows_\beta t_1'} \qquad \dfrac{}{t_2 \rightrightarrows_\beta t_2}\ \textsc{Lemma-3.2}}{t_1\ t_2 \rightrightarrows_\beta t_1'\ t_2}$$

- The other case are similar.

Lemma 3.4 – $\forall tt', t \rightrightarrows_\beta t' \Rightarrow t \to_\beta^* t' \qquad \rightrightarrows_\beta \subseteq \to_\beta^*$

Proof: by induction on $\rightrightarrows_\beta$

- Case $x \rightrightarrows_\beta x$. We have $x \to_\beta^0 x$

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

□

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

Lemma 3.5 – If a relation $\to$ has the diamond property then $\to^*$ has the diamond property to.
Proof : We suppose that $\to$ has the diamond property [TODO] □

Lemma 3.6 – If two relation $\to$ and $\rightrightarrows$ are such that $\to \subseteq \rightrightarrows \subseteq \to^*$ then $\rightrightarrows^* = \to^*$.
Proof: From $\to \subseteq \rightrightarrows \subseteq \to^*$ we deduce $\to^* \subseteq \rightrightarrows^* \subseteq \to^{**}$. However we have $\to^* = \to^{**}$. Then $\to^* \subseteq \rightrightarrows \subseteq \to^*$ so we have $\rightrightarrows^* = \to^*$.
□

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

Lemma 3.7 – $a \rightrightarrows_\beta a' \wedge b \rightrightarrows_\beta b' \Rightarrow a\{x \leftarrow b\} \rightrightarrows_\beta a'\{x \leftarrow b\}$
Proof: By induction one the derivation $a \rightrightarrows_\beta a'$.

- Case $y \rightrightarrows_\beta y$
  - If $x = y$, then $x\{x \leftarrow b\} = b \rightrightarrows_\beta b' = x\{x \leftarrow b'\}$
  - If $x \neq y$, then $y\{x \leftarrow b\} = y \rightrightarrows_\beta y = y\{x \leftarrow b'\}$
- Case $\lambda y.a_0 \rightrightarrows_\beta \lambda y.a'_0$ with $a_0 \rightrightarrows_\beta a'_0$

  Then $(\lambda y.a_0)\{x \leftarrow b\} = \lambda y.a_0\{x \leftarrow b\}$. By induction hypothesis we have $a_0\{x \leftarrow b\} \rightrightarrows_\beta a'_0\{x \leftarrow b'\}$.

  Therefore, we have $\lambda y.a_0\{x \leftarrow b\} \rightrightarrows_\beta \lambda y.a'_0\{x \leftarrow b'\} = (\lambda y.a'_0)\{x \leftarrow b'\}$
- Case $a_1\ a_2 \rightrightarrows_\beta a'_1\ a'_2$ with $a_1 \rightrightarrows_\beta a'_1$ and $a_2 \rightrightarrows_\beta a'_2$.

  It is similar to the case above.
- Case $(\lambda x.a_1)\ a_2 \rightrightarrows_\beta (\lambda x.a'_1)a'_2$ with $a_1 \rightrightarrows_\beta a'_1$ and $a_2 \rightrightarrows_\beta a'_2$.

  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 $a_1\{x \leftarrow b\} \rightrightarrows_\beta a'_1\{x \leftarrow b'\}$ and $a_2\{x \leftarrow b\} \rightrightarrows_\beta a'_2\{x \leftarrow b'\}$.

  Therefore $(\lambda y.a_1\{x \leftarrow b\})(a_2\{\leftarrow b\}) \rightrightarrows_\beta (a'_1\{x \leftarrow b'\}\{y \leftarrow a'_2\{x \leftarrow b'\}\}$. Thanks to the substitution lemma-2.4

□

Lemma 3.8 – $\rightrightarrows_\beta$ has the diamond property. $\forall t\ s\ r, t \rightrightarrows_\beta s \wedge t \rightrightarrows_\beta r \Rightarrow \exists u, s \rightrightarrows_\beta u \wedge r \rightrightarrows_\beta u$.
Proof: By induction on the derivation of $t \rightrightarrows_\beta re$.

- Case $x \rightrightarrows_\beta x$. Then $s = x$ so we can take $u = x$.
- Case $\lambda x.t_0 \rightrightarrows_\beta \lambda x.r_0$ with $t_0 \rightrightarrows_\beta r_0$. Then $s = \lambda x.s_0$ with $t_0 \rightrightarrows_\beta s_0$.

  By induction hypothesis we have u_0 such that $s_0 \rightrightarrows_\beta u_0$ and $r_0 \rightrightarrows_\beta u_0$.

  Therefore, $\lambda x.s_0 \rightrightarrows_\beta \lambda x.u_0$ And $\lambda x.r_0 \rightrightarrows_\beta \lambda x.u_0$
- Case $t_1\ t_2 \rightrightarrows_\beta r_1\ r_2$ with $t_1 \rightrightarrows_\beta r_1$ and $t_2 \rightrightarrows_\beta r_2$. Two case for $t_1\ t_2 \rightrightarrows_\beta s_0$:
  - if $s = s_1\ s_2$ with $t_1 \rightrightarrows_\beta s_1$ and $t_2 \rightrightarrows_\beta s_2$ by induction hypothesis there are u_1 and u_2 such that $s_1 \rightrightarrows_\beta u_1$, $r_1 \rightrightarrows_\beta u_1$ and $s_2 \rightrightarrows_\beta u_2$, $r_2 \rightrightarrows_\beta u_2$, therefore $s_1 s_2 \rightrightarrows_\beta u_1 u_2$ and $r_1 r_2 \rightrightarrows_\beta u_1 u_2$.

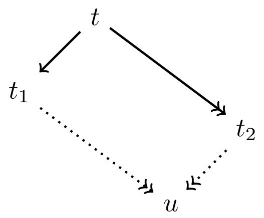
– if $s = s_1\{x \leftarrow s_2\}$ with $t_1 = \lambda x.t_1$ and $t_1 \rightrightarrows_\beta s_1$ and $t_2 \rightrightarrows_\beta s_2$, then $r_1 = \lambda x.r_1$ with $t_1' \rightrightarrows_\beta r_1'$ and by induction hypothesis there are u_1 and u_2 such that $s_1 \rightrightarrows_\beta u_1$ and $r_1' \rightrightarrows_\beta u_1$ and $s_2 \rightrightarrows_\beta u_2$ and $r_2 \rightrightarrows_\beta u_2$.
Therefore, $(\lambda x.r_1')r_2 \rightrightarrows_\beta u_1\{x \leftarrow s_2\}$. And we conclude by the lemma-3.7

- The last case is almost the same.

□

3.3.2 Strip lemma

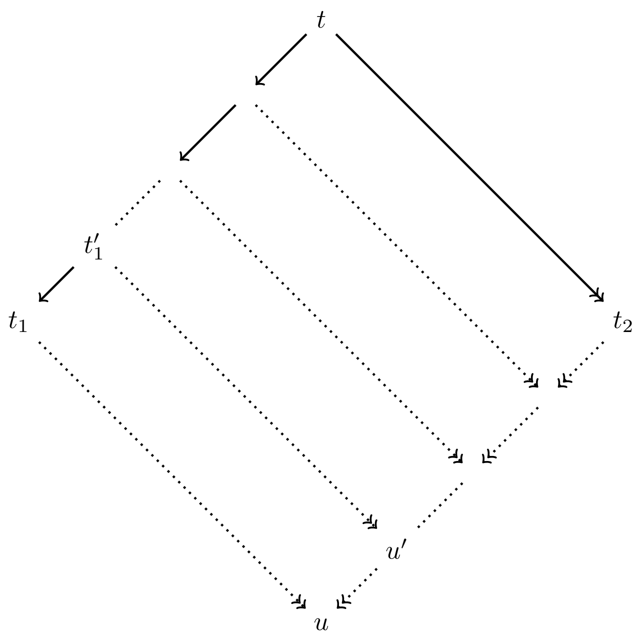
Since confluence is difficult to show duo to the infinity of the two branches, we define a new property called 'strip lemma'. The concept is simple, instead of having the relation $\rightarrow^*$ on both sides of the confluence we use the relation $\rightarrow$ and $\rightarrow^*$. It is formally defined as follows : $\forall t\ t_1\ t_2, t \rightarrow t_1 \wedge t \rightarrow t_2 \Rightarrow \exists u, t_1 \rightarrow^* u \wedge t_2 \rightarrow^* u$. We can draw this diagram :



This property given the confluence :
Proof : We can proof by induction on the length on the left reduction (confluence diagram).

- When $n = 0$ is trivial, $t_1 = t$ then $t_1 \rightarrow^* t_2$.
- For $n + 1$, we have $t \rightarrow_n t_1' \rightarrow t_1$, such that exists u' with $t_2 \rightarrow^* u'$ and $t_1' \rightarrow u$ (induction hypothesis). We just need to apply the strip lemma to have u such that $t_1 \rightarrow u$ and $u' \rightarrow^* u$

It's easier to understand the proof with this drawing :



□

To proof that the λ-calculus respect the "strip lemma" we need to define the $\underline{\lambda}$-calculus ($\underline{\Lambda}$ the set of $\underline{\lambda}$-terms).

Definition – $\underline{\Lambda}$ Set:

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

We write $|T|$ the operation who remove every mark on a $\underline{\lambda}$-term.

$$\begin{cases} |x| & = x \\ |\lambda x.t| & = \lambda x.|t| \\ |t_1\ t_2| & = |t_1|\ |t_2| & \text{if } t_1 \neq \underline{\lambda} x.t' \\ |(\underline{\lambda} x.t_1)\ t_2| & = (\lambda x.|t_1|)\ |t_2| \end{cases}$$

We need to define a new substitution and a new rewrite rule :

Definition – Substitution on $\underline{\Lambda}$
Substitution is defined as for normal lambda terms. We only add the following rule:

$$((\underline{\lambda} x.t)u)\{y \leftarrow v\} \equiv (\underline{\lambda} x.t\{y \leftarrow v\})u\{y \leftarrow v\}$$

Definition – For the $\underline{\beta}$-reduction we define 2 rules that propagate recursively on the other cases :

$$\beta_0 : (\underline{\lambda} x.t)u \to t\{x \leftarrow u\}$$
$$\beta_0 : (\lambda x.t)u \to t\{x \leftarrow u\}$$

Definition – We define the function φ who contract all marked redex on a $\underline{\lambda}$-term :

$$\begin{cases} \varphi(x) & = x \\ \varphi(u\ v) & = \varphi(u)\ \varphi(v) & \text{if } u \neq \underline{\lambda} x.t \\ \varphi(\lambda x.t) & = \lambda x.\varphi(t) \\ \varphi((\underline{\lambda} x.t)\ u) & = \varphi(t)\{x \leftarrow \varphi(u)\} \end{cases}$$

Lemma 3.9 – $|t|$ property:

1. $\forall m' \in \underline{\lambda}, mn \in \lambda, m' \xrightarrow{|\,|} m \wedge M \xrightarrow{\beta} N \Rightarrow \exists N' \in \underline{\Lambda}, M' \xrightarrow{\underline{\beta}} N' \wedge N' \xrightarrow{|\,|} N$

$$\begin{tikzcd} M' \arrow[r, two heads, dotted, "\underline{\beta}"] \arrow[d, "|\,|"'] & N' \arrow[d, dotted, "|\,|"] \\ M \arrow[r, two heads, "\beta"'] & N \end{tikzcd}$$

2. $\forall m' \in \underline{\lambda}, mn \in \lambda, m' \xrightarrow{|\,|} m \wedge M \xrightarrow{\beta} N \Rightarrow \exists N' \in \underline{\Lambda}, M' \xrightarrow{\underline{\beta}} N' \wedge N' \xrightarrow{|\,|} N$

$$\begin{tikzcd} M' \arrow[r, two heads, "\underline{\beta}"] \arrow[d, dotted, "|\,|"'] & N' \arrow[d, "|\,|"] \\ M \arrow[r, two heads, dotted, "\beta"'] & N \end{tikzcd}$$

Proof:

1. We proceed by induction on the reduction $M \to_\beta^* N$.
   - $M \to_\beta^* M$, then $M = N$. We just take $M' = M$ (easy).
   - $M \to_\beta M_t$ and $M_t \to_\beta^* N$. By induction hypothesis we have M'_t such that $|M'_t| = M_t$ and $M'_t \to_\beta^* N'$. The reduction $M' \to_{\underline{\beta}} M'_t$ exists, we can contract the same redex (if it was marked we would contract the new unmarked).

     So we have $M' \to_{\underline{\beta}} M'_t \to_{\underline{\beta}} N' \xrightarrow{|\,|} N$
2. Similar.

□

Lemma 3.10 – φ property

1. $\forall uv \in \underline{\Lambda}, \varphi(u\{x \leftarrow v\}) = \varphi(u)\{x \leftarrow \varphi(v)\}$
2. $\forall MN \in \underline{\Lambda}, M \to_{\underline{\beta}}^* N \Rightarrow \varphi(M) \to_\beta^* \varphi(N)$

$$\begin{tikzcd}
M \arrow[r, two heads] \arrow[d] & N \arrow[d] \\
\varphi(M) \arrow[r, dotted, two heads] & \varphi(N)
\end{tikzcd}$$

Proof:

1. We proceed by induction on the term u.
   - Case $u = z$
     - if $z = x$, then $\varphi(x\{x \leftarrow v\}) = \varphi(v) = \varphi(x)\{x \leftarrow \varphi(v)\}$
     - if $z \neq x$, then $\varphi(z\{x \leftarrow v\}) = z = \varphi(z)\{x \leftarrow \varphi(v)\}$
   - Case $t = \lambda y.t_0$,

$$\begin{aligned}
\varphi((\lambda y.t_0)\{x \leftarrow v\}) &= \varphi(\lambda y.t_0\{x \leftarrow v\}) \\
&= \lambda y.\varphi(t_0\{x \leftarrow v\}) && \text{By convention} \\
&= \lambda y.\varphi(t_0)\{x \leftarrow \varphi(v)\} && \text{By induction hypothesis} \\
&= \varphi(\lambda y.t_0)\{x \leftarrow \varphi(v)\}
\end{aligned}$$

   - Case $t = t_1\ t_2$

$$\begin{aligned}
\varphi((t_1\ t_2)\{x \leftarrow v\}) &= \varphi(t_1\{x \leftarrow v\})(\varphi(t_2\{x \leftarrow v\})) \\
&= \varphi(t_1)\{x \leftarrow \varphi(v)\}\varphi(t_2)\{x \leftarrow \varphi(v)\}) && \text{By induction hypothesis} \\
&= (\varphi(t_1)\ \varphi(t_2))\{x \leftarrow \varphi(v)\}) \\
&= (\varphi(t_1\ t_2))\{x \leftarrow \varphi(v)\})
\end{aligned}$$

   - Case $t = (\underline{\lambda} y.t_0)\ t_1$

$$\begin{aligned}
\varphi(((\underline{\lambda} y.t_0)\ t_1)\{x \leftarrow v\}) &= \varphi((\underline{\lambda} y.t_0\{x \leftarrow v\})\ t_1\{x \leftarrow v\}) \\
&= \varphi(t_0\{x \leftarrow v\})\{y \leftarrow \varphi(t_1\{x \leftarrow v\})\} \\
&= \varphi(t_0)\{x \leftarrow \varphi(v)\})\{y \leftarrow \varphi(t_1)\{x \leftarrow \varphi(v)\}\} && \text{By induction hypothesis} \\
&= \varphi(t_0)\{y \leftarrow \varphi(t_1)\}\{x \leftarrow \varphi(v)\} && \text{By the lemma 2.4} \\
&= \varphi((\underline{\lambda} y.t_0)\ t_1)\{x \leftarrow \varphi(v)\}
\end{aligned}$$

2. By induction on $\to_{\underline{\beta}}^*$, using (1)

□

Lemma 3.11 – $\forall t, |t| \to_\beta^* \varphi(t)$

$$\begin{tikzcd}
 & t \arrow[dl, "| \ |"'] \arrow[dr, "\varphi"] & \\
|t| \arrow[rr, dotted, two heads, "\beta"'] & & \varphi(t)
\end{tikzcd}$$

Proof: We will show our property by induction on the term t.

- Case $t = x$, $|x| = x = \varphi(x)$, so we have $|t| \to_\beta^0 \varphi(t)$.
- Case $t = \lambda x.t_0$, by the induction hypothesis we know that $|t_0| \to_\beta^* \varphi(t_0)$.

$$\begin{aligned}
|(\lambda x.t_0)| &= \lambda x.|t_0| \\
&\to_\beta^* \lambda x.\varphi(t_0) && \text{By induction hypothesis} \\
&= \varphi(\lambda x.t_0)
\end{aligned}$$

- Case $t = t_1\ t_2$, by the induction hypothesis we know that $|t_1| \to_\beta^* \varphi(t_1)$ and $|t_2| \to_\beta^* \varphi(t_2)$.

$$\begin{aligned}
|t_1\ t_2| &= |t_1|\ |t_2| \\
&\to_\beta^* \varphi(t_1)\ \varphi(t_2) && \text{By induction hypothesis} \\
&= \varphi(t_1\ t_2)
\end{aligned}$$

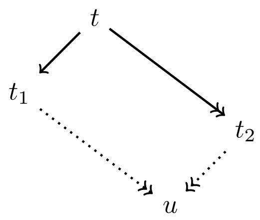
- Case $t = (\underline{\lambda} x.t_0)\ t_1$, by the induction hypothesis we know that $|t_0| \to_\beta^* \varphi(t_0)$ and $|t_1| \to_\beta^* \varphi(t_1)$.

$$\begin{aligned}
|(\underline{\lambda} x.t_0)\ t_1| &= (\lambda x.|t_0|)\ |t_1| \\
&\to_\beta^* (\lambda x.\varphi(t_0))\ \varphi(t_1) && \text{By induction hypothesis} \\
&\to \varphi(t_0)\{x \leftarrow \varphi(t_1)\} \\
&= \varphi((\underline{\lambda} x.t_0)\ t_1)
\end{aligned}$$

$\square$

Lemma 3.12 – Strip-lemma:

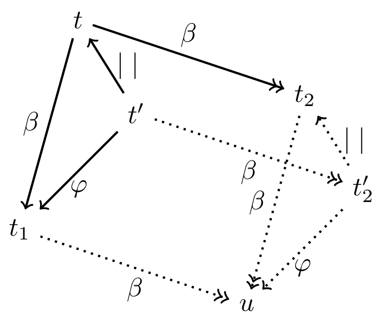
$$\forall t\ t_1\ t_2, t \to t_1 \wedge t \to t_2 \Rightarrow \exists u, t_1 \to^* u \wedge t_2 \to^* u$$



Proof: Let t', such that $|t'| = t$ and $\varphi(t') = t_1$. By hypothesis, we know that, $t \to_\beta^* t_2$, so we have by the Lemma-3.9 $t'_2 \in \underline{\Lambda}$ with $|t'_2| = t_2$. We take $u = \varphi(t'_2)$.

Then by the Lemma-3.10, we have $t_1 \to_\beta^* u$ and $\varphi(t'_2) = u$. We finally have $t'_2 \to_\beta^* u$ thanks to the Lemma-3.11.

We can resume this proof with the diagram below:



Resume:

1. Lemma-3.9, give $t'_2 : t' \to_\beta^* t'_2$ and $|t'_2| = t_2$.
2. Lemma-3.11, give $t_2 \to_\beta^* \varphi(t'_2) = u$.
3. Lemma-3.10, give $t_1 \to_\beta^* \varphi(t'_2) = u$

□

3.3.3 Church-Rosser theorem

If $t_1 =_\beta t_2$ the there is u such that $t_1 \to_\beta^* u$ and $t_2 \to_\beta^* u$

Consequences :

- if t has a normal form n then $t \to_\beta^* n$
- any λ-term can has only one normal form
- if two normal form n and m are syntactically different, then $n \neq_\beta m$.