λ-calculus

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:

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

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

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

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

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 $f_v(t_1\ t_2) = f_v(t_1) + f_v(t_2) = 1 + f_@(t_1) + 1 + f_@(t_2) = 1 + f_@(t_1\ t_2)$

□

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 :

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

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

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

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 :

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

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 :

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

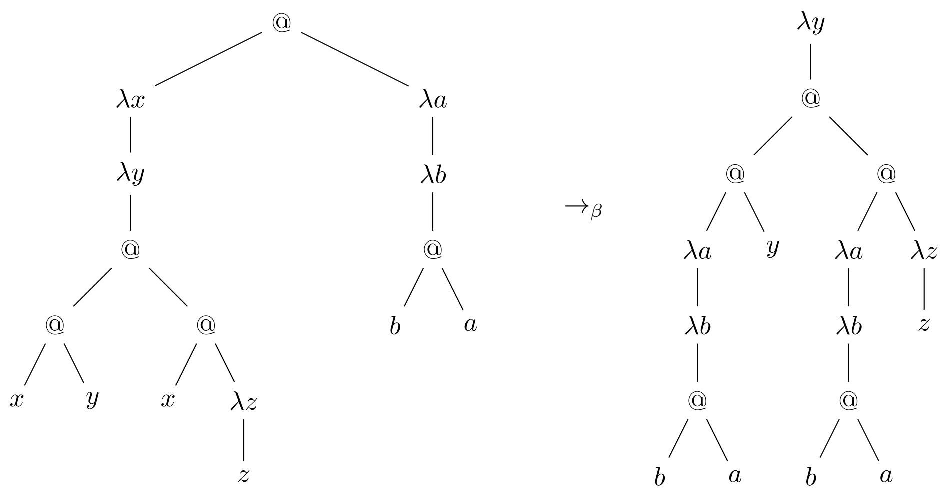
Exercise 2.3 – Prove that $fv(t[x \leftarrow u]) \subseteq (fv(t)\backslash\{x\}) \cup fv(u)$

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 :

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

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

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

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

3 Reduction strategies

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

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

This raises the questions :

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

3.1 Normalization

3.2 Reduction strategies

3.3 Confluence