λ-calcul : Une introduction

Notes du cours de Loïc Colson prises par N. Noble

dernière révision : 24 janvier 2005

Contents

Chapitre 0

Motivations

– Notion de fonction

– Distinction entre

$$f(x)$$

et

$$x \mapsto (f(x)) \quad (Bourbaki)$$

$\longrightarrow$ en λ-calcul on note $\lambda x.(fx)$

– Notion de **fonctionnelle**

$$f \mapsto \int_0^1 f(x)dx$$

$\longrightarrow$ une fonction peut être passée en argument à une fonctionnelle.

– En **informatique**, langages fonctionnels : LISP, SCHEME, ML, HASKELL...

$$\begin{array}{c} \text{fun } x \longrightarrow \text{fun } f \longrightarrow (fx) \\ :\alpha \longrightarrow (\alpha \longrightarrow \beta) \longrightarrow \beta \end{array}$$

$\longrightarrow$ en λ-calcul

$$\lambda f \lambda x.fx$$

ou

$$\lambda fx.fx$$

Chapitre 1

Les termes du λ-calcul

On considère un ensemble infini dénombrable $V_{av} = \{x, y, z, ...\}$ de **variables** (identificateurs).

L'ensemble des termes du λ-calcul est un ensemble d'arbres défini comme suit :

Définitions:

- Toute variable est un terme.
- Si M et N sont des termes, alors MN est un terme. ("M appliqué à N")
- Si M est un terme et x une variable, alors $\lambda x.M$ est un terme. ("la fonction qui a x associe M")

Exemples:

$$Id = \lambda x.x$$

$$App = \lambda x \lambda y.xy$$

Définition: L'ensemble des λ-termes sera noté Λ.

Notations:

– $\lambda x_1 x_2 ... x_n.M$ est une abréviation pour $\lambda x_1.\lambda x_2.....\lambda x_n.M$.

– $MN_1N_2...N_p$ est une abréviation pour $(...((MN_1)N2)...N_p)$.

Chapitre 2

Variables libres

MOTIVATIONS:
Dans une expression comme $\int_0^1 f(x)dx$, x n'apparaît pas vraiment. On dit que c'est une variable **muette** ou **liée**. Intuitivement, $\int_0^1 f(x)dx$ est la même expression que $\int_0^1 f(y)dy$.

$\rightarrow$ On dit que ces deux expressions ne diffèrent que par un renommage de leurs variables liées.

$\rightarrow$ On **identifiera** en λ-calcul des expressions comme ($\lambda x.x$ et $\lambda y.y$) ou ($\lambda xy.xy$ et $\lambda yx.yx$) que l'on considèrera comme **égales**.

Par contre dans $\int_0^1 f(x,y)dx$ la variable y apparaît vraiment : on dit que y est **libre** dans cette expression.

$\rightarrow$ On définit l'ensemble des **variables libres** $VL(M)$ d'un λ-terme M par récurrence sur M :

Définitions:

– $VL(x) = \{x\}$

– $VL(MN) = VL(M) \cup VL(N)$

– $VL(\lambda x.M) = VL(M) \setminus \{x\}$

Exemples:

– $VL(\lambda xy.xy) = \emptyset$

– $VL(\lambda x.xy) = \{y\}$

– $VL((\lambda xy.xy)y) = \{y\}$

Chapitre 3

Substitution

Soient $M, N_1, ..., N_p \in \Lambda$ et $x_1, ..., x_p \in V_{av}$

On note $M[x_1 \leftarrow N_1, ..., x_p \leftarrow N_p]$ le résultat de la **substitution** des x_i par les N_i dans M.

Définition: $M[x_1 \leftarrow N_1, ..., x_p \leftarrow N_p]$ est défini par :

– $z[] = z \quad \text{si } z \neq x_i \quad \forall i$
$\qquad = N_i \quad \text{si } z = x_i$

– $(MN)[] = (M[])(N[])$

– $(\lambda x.M)[] = \lambda x.(M[])$

$\Diamond$ Dans le cas du λ (λ abstraction) il faut renommer parfois les variables libres pour éviter que des variables libres des N_i ne deviennent liées dans $\lambda x.(M[])$

Exemples:

– $xy[x \leftarrow N_1, y \leftarrow N_2] = N_1 N_2$

– $(\lambda x.xy)[y \leftarrow z] = \lambda x.xz$

– $(\lambda x.xy)[y \leftarrow xx] = \lambda x'.x'(xx)$

– $(\lambda x.x(\lambda y.yz))[z \leftarrow y] = \lambda x.x(\lambda y'.y'y)$

Chapitre 4

La β-réduction

Motivations: Quand on applique la fonction $x \mapsto 3x+4$ à l'entier 5, on calcule le résultat en $(3\times5)+4$. On dit que l'on a effectué la réduction de $(x \mapsto 3x+4)5$ à $(3\times5)+4$.

Définitions:

– Un λ-terme de la forme

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

est appelé un **rédex** (réductible expression)

– Le terme $M[x \leftarrow N]$ est alors appelé le **contractum** de ce rédex.

Intuition:
Une étape de calcul amène du redex au contractum.

La β-réduction

Définition:
$\longrightarrow$ est la plus petite relation entre λ-termes telle que :

- $(\lambda x.M)N \longrightarrow M[x \leftarrow N]$
- Si $M \longrightarrow M'$ alors $MN \longrightarrow M'N$
- Si $N \longrightarrow N'$ alors $MN \longrightarrow MN'$
- Si $M \longrightarrow M'$ alors $\lambda x.M \longrightarrow \lambda x.M'$

Si $M \longrightarrow N$ on dit que **M se β-réduit en une étape à N**.

Exemples:

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

Remarque:
Il peut y avoir plusieurs rédex dans un λ-terme.

Définition:
$\xrightarrow{*}$ est la fermeture réflexive et transitive de $\rightarrow$. En d'autres termes, $M \xrightarrow{*} N$ si $\exists P_1, ..., P_n \in \Lambda$ tels que

$$M = P_1 \rightarrow P_2 \rightarrow ... \rightarrow P_n = N$$

Exemples:

– $(\lambda x.x)((\lambda y.y)z) \xrightarrow{*} z$ car
$(\lambda x.x)((\lambda y.y)z) \rightarrow (\lambda x.x)z \rightarrow z$ ou bien
$(\lambda x.x)((\lambda y.y)z) \rightarrow (\lambda y.y)z \rightarrow z$

– $(\lambda f.fx)(\lambda y.y) \rightarrow (\lambda y.y)z \rightarrow z$

Remarque:
Un rédex peut en créer un autre.

Si $M \xrightarrow{*} N$ on dit que M **se réduit à** N.

Chapitre 5

La β-conversion

Motivations:
Quand on raisonne sur des expressions, on peut avoir envie de "remonter le cours du temps" en passant de $(3\times5)+4$ à $(x \mapsto 3x+4)5$.

Définition:
$=_\beta$ est la fermeture réflexive, **symétrique** et transitive de $\rightarrow$.

En d'autres termes, $M =_\beta M'$ si $\exists N_1, ..., N_p$ tels que $M = N_1$, $M' = N_p$, et

$$N_i \rightarrow N_{i+1}$$

ou

$$N_{i+1} \rightarrow N_i$$

Chapitre 6

Formes normales

Définition:

Un terme $M \in \Lambda$ est dit en **forme normale** s'il n'existe aucun $N \in \Lambda$ tel que $M \to N$.

En d'autres termes, M ne contient **aucun rédex** : il est **complètement calculé**.

Exemple:

$$\lambda x.y(\lambda z.z(yz))$$

Lemme:

Un terme en forme normale est de la forme $\lambda x_1 ... x_p.zM_1 ... M_k$ avec les M_i **en forme normale**.

Définition:

– On dit que M est **normalisable** si $\exists M, N \in \Lambda$ tels que $M \xrightarrow{*} N$ et N en **forme normale**.

– N est alors une **forme normale** de M.

Exemples:

– $(\lambda x.xz)(\lambda y.y)$ est normalisable et admet z pour forme normale.

– Posons $\Delta = \lambda x.xx$. On a

$$\Delta\Delta \to \Delta\Delta \to \Delta\Delta \to ...$$

et c'est la seule suite de réductions issues de $\Delta\Delta$.

$\to$ Donc $\Delta\Delta$ est **non-normalisable**.

Définition:
On dit que $M \in \Lambda$ est **fortement normalisable** si toutes les réductions issues de M sont **finies**.

Exemples:

– $\Delta\Delta$ n'est pas fortement normalisable (il n'est pas normalisable tout court)

– $(\lambda x.x)((\lambda y.y)z)$ est fortement normalisable (vu en exemple plus tôt)

– $(\lambda x.y)(\Delta\Delta)$ est **normalisable** mais **pas** fortement normalisable.

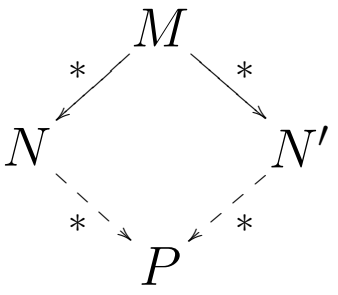
$\rightarrow$ Tous les termes fortement normalisables sont normalisables, mais la réciproque est fausse.

Chapitre 7

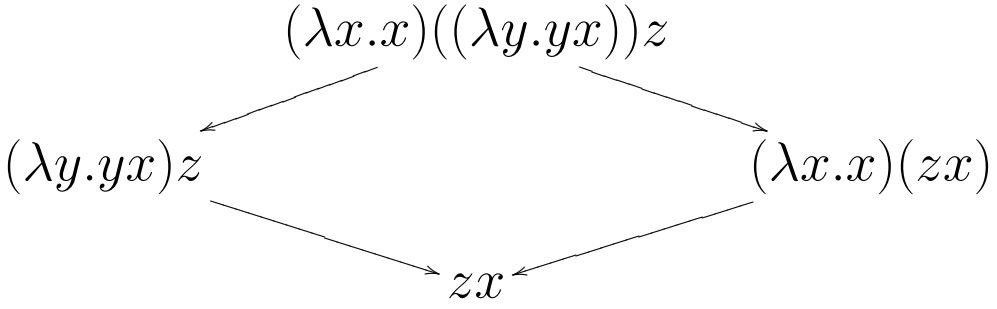
Propriété de *Church-Rosser*

Théorème (Church-Rosser)

Soient $M, N \in \Lambda$. Supposons que $M \xrightarrow{*} M'$ et $N \xrightarrow{*} N'$. Alors il existe $P \in \Lambda$ tel que $M' \xrightarrow{*} P$ et $N' \xrightarrow{*} P$.



Exemple:



Terminologie: On dit que la relation $\xrightarrow{*}$ est **confluente**.

Corollaire 1

Soit $M \in \Lambda$. Alors M a au plus une forme normale.

Corollaire 2

Le λ-calcul est cohérent, i.e. on n'a pas $M =_\beta N$ pour tout M et N.

Lemme: $M =_\beta N \Rightarrow \exists P$ tq $M \xrightarrow{*} P$ et $N \xrightarrow{*} P$.

Chapitre 8

Standardisation

Définition:
On écrit $M \underset{g}{\longrightarrow} N$ si N est obtenu en contractant le rédex le plus à gauche de M.

Théorème:
Si $M \in \Lambda$ est normalisable en N, alors $M \underset{g}{\overset{*}{\longrightarrow}} N$.

$\rightarrow$ On dit que la "stratégie gauche" est "sûre".

$\rightarrow$ On va maintenant donner les outils de base pour programmer sur les structures de données les plus courantes en λ-calcul.

Chapitre 9

Booléens

Définition :

– true $= \lambda xy.x$

– false $= \lambda xy.y$

– if $= \lambda bxy.bxy$

Lemme :

– if true $MN =_\beta M$

– if false $MN =_\beta N$

Exemple :

$$M \text{ and } N = \text{if } M \text{ then } \left| \begin{array}{l} \text{if } N \left| \begin{array}{l} \text{then true} \\ \text{else false} \end{array} \right. \\ \text{else false} \end{array} \right.$$

Chapitre 10

Produit cartésien

Définition :

– $< M, N >= \lambda z.(zM)N$

– $\Pi_1 = \lambda c.c(\lambda xy.x)$

– $\Pi_2 = \lambda c.c(\lambda xy.y)$

Lemme :

– $\Pi_1 < M, N >=_\beta M$

– $\Pi_2 < M, N >=_\beta N$

Remarque : On peut généraliser aux triplets...

Chapitre 11

Entiers

$\rightarrow$ On représentera l'entier n par :

$$\overline{n} = \lambda f x.f^n x$$

avec $M^0 N = N$ et $M^{p+1} N = M(M^p N)$.

Définition:

– $z = \lambda f x.x$ (zéro)

– succ $= \lambda n f x.f(nfx)$

Lemme:

succ $\overline{n} =_\beta \overline{n+1}$

Définition par récurrence:

– $\varphi(0) = a$

– $\varphi(n+1) = \theta(\varphi(n))$

Exemple: $n \mapsto 2^n$

Supposons que l'on a déjà programmé θ en λ-calcul

$\rightarrow$ terme M_θ

On veut un terme P tel que

$$P\overline{n} =_\beta \underbrace{M_\theta(M_\theta ... M_\theta(a)...)}_{n \text{ fois}}$$

$\rightarrow$ on prend

$$P = \lambda n.n\ M_\theta\ a$$

Définition: iterate $= \lambda nfx.n\ f\ x$

Lemme:

– iterate $\overline{0}\ M\ a =_\beta a$

– iterate $\overline{n+1}\ M\ a =_\beta M(\text{iterate}\ \overline{n}\ M\ a)$

Remarque: iterate $x\ y\ z =_\beta xyz$

Exemple :

– addition

$$\begin{aligned} 0+p &= p \\ (n+1)+p &= (n+p)+1 = \text{succ}\ (n+p) \\ \text{add} &= \lambda np.\text{iterate}\ n\ \text{succ}\ p \\ &= \lambda np.n\ \text{succ}\ p \end{aligned}$$

– multiplication

$$\begin{aligned} 0\times p &= 0 \\ (n+1)\times p &= p+(n\times p) \\ \text{mult} &= \lambda np.\text{iterate}\ n\ (\text{add}\ p)\ 0 \\ &= \lambda np.n\ (\text{add}\ p)\ 0 \end{aligned}$$

– $n \mapsto 2^n$

$$\begin{aligned} 2^0 &= 1 \\ 2^{n+1} &= 2^n + 2^n \end{aligned}$$

– $\text{exp} = \lambda np.\text{iterate}\ n\ (\lambda p.\text{add}\ p\ p)\ \overline{1}$

$\rightarrow$ Un programme difficile : $n \mapsto n \dot{-} 1$

Définition:

$$\begin{aligned} \varphi(0) &= <0,0> \\ \varphi(n+1) &= <\Pi_2(\varphi(n)), \text{succ } (\Pi_2(\varphi(n)))> \end{aligned}$$

Lemme: $\varphi(n) = <n \dot{-} 1, n>$

Corollaire: $\Pi_1(\varphi(n)) = n \dot{-} 1$

$\rightarrow$ en λ-calcul

pred $= \lambda n.\Pi_1(\text{iterate } n\ \lambda c. <\Pi_2(c), \text{succ}(\Pi_2(c))> <\overline{0}, \overline{0}>)$

Lemme: pred $\overline{n} =_\beta \overline{n \dot{-} 1}$

Soustraction :

– $n \dot{-} 0 = n$

– $n \dot{-} (p+1) =$pred $n \dot{-} p$

$$\begin{aligned} \text{sub} &= \lambda np.\text{iterate } p \text{ pred } n \\ &= \lambda np.p \text{ pred } n \end{aligned}$$

Lemme: sub $\overline{n}\ \overline{p} =_\beta \overline{n \dot{-} p}$

Chapitre 12

Listes

$\rightarrow$ On représentera la liste $l = [a_1; a_2; ...; a_n]$ par

$$\bar{l} = \lambda gy.ga_1(ga_2...(ga_n y)...)$$

Définitions:

– nil $= \lambda gy.y$

– cons $= \lambda algy.ga(lgy)$

Lemme:

cons a $\overline{[b_1; ...; b_n]} =_\beta \overline{[a; b_1; ...; b_n]}$

Définition par récurrence sur les listes :

– $\varphi(\text{nil}) = c$

– $\varphi(a :: l) = \theta(a, \varphi(l))$

Exemple: Somme des éléments d'une liste

– $\sigma(\text{nil}) = 0$

– $\sigma(a :: l) = a + \sigma(l)$

$$\sigma[1; 2; 3] = 6$$

Si θ est représentée par M_θ on programme φ par

$$P = \lambda l.l\ M_\theta\ c$$

Définition: list_it $= \lambda lgy.l\ g\ y$

Lemme:

– list_it nil $M\ c =_\beta c$

– list_it (cons $a\ l$) $M\ c =_\beta\ M\ a$ (list_it $l\ M\ c$)

Exemples:

– addition

$$\begin{aligned} \text{sum} &= \lambda l.\text{list\_it}\ l\ \text{add}\ \overline{0} \\ &= \lambda l.l\ \text{add}\ \overline{0} \end{aligned}$$

– produit

$$\begin{aligned} \text{prod} &= \lambda l.\text{list\_it}\ l\ \text{mult}\ \overline{1} \\ &= \lambda l.l\ \text{mult}\ \overline{1} \end{aligned}$$

Tête d'une liste

Définition: $\text{hd} = \lambda l.\text{list\_it } l\ (\lambda ay.a)(\lambda x.x)$

Lemme: $\text{hd}\ [a_1; ...; a_n] =_\beta a_1$

Queue d'une liste

Définition:
- T nil =<nil,nil>

- $T\ a :: l =< \Pi_2(T(l)), a :: \Pi_2(T(l)) >$

Lemme:
- T nil =<nil,nil>
- $T\ a :: l =< l, a :: l >$

Définition: $\text{tail}\ l = \Pi_1(T(l))$

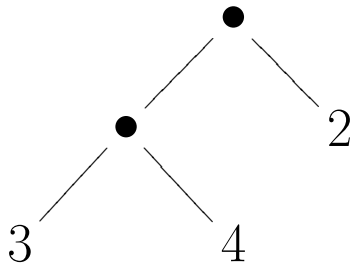
$\rightarrow$ en λ-calcul

Définition:
$\text{tail} = \lambda l.\Pi_1(\text{list\_it } l\ \lambda ac. < \Pi_2(c), a :: \Pi_2(c) > <\text{nil,nil}>)$

Chapitre 13

Arbres binaires

→ On représente en λ-calcul l'arbre

$t =$ 

par $\lambda hb.h\ (h\ (b3)\ (b4))\ (b2)$

Définition:

– leaf $= \lambda nhb.b\ n$

– tcons $= \lambda t_1 t_2 hb.h\ (t_1\ h\ b)\ (t_2\ h\ b)$

Exemple:

tcons (tcons (leaf 3) (leaf 4)) (leaf 2) $=_\beta t$

Définition par récurrence sur les arbres

– $f(\text{leaf } n) = (l\ n)$

– $f(\text{tcons } t_1\ t_2) = g\ (f\ t_1)\ (f\ t_2)$

Exemple: Nombre de feuilles d'un arbre

– $N(\text{leaf } \_) = 1$

– $N(\text{tcons } t_1\ t_2) = N(t_1) + N(t_2)$

$\rightarrow$ En λ-calcul on introduit :

Définition: tree_it $= \lambda tfg.t\ f\ g$ et on a :

Lemme:

– tree_it (leaf n) $g\ l = (l\ n)$

– tree_it (tcons $t_1\ t_2$) $g\ l = g$ (tree_it $t_1\ g\ l$) (tree_it $t_2\ g\ l$)

Exemple: Nombre de feuilles

$$\begin{aligned} N &= \lambda t.\text{tree\_it } t \text{ add } (\lambda x.\overline{1}) \\ &= \lambda t.t \text{ add } (\lambda x.\overline{1}) \end{aligned}$$

Exercice : Mise à plat d'un arbre, fils droit, fils gauche.

Chapitre 14

Utilisation de la fonctionnalité

Problème :

Comment programmer en λ-calcul la fonction d'Ackermann :

- $A\ 0\ p = p + 1$
- $A\ (n+1)\ 0 = A\ n\ 1$
- $A\ (n+1)\ (p+1) = A\ n\ (A\ (n+1)\ p)$

Solution :

$$\begin{array}{l} A\ 0 = \lambda p.(\text{succ } p) \\ A\ (n+1) = G(A\ n) \end{array} \quad \text{avec} \quad \left| \begin{array}{l} G\ u\ 0 = u\ 1 \\ G\ u\ (p+1) = u\ (G\ u\ p) \end{array} \right.$$

$\rightarrow$ en λ-calcul

$$A = \lambda n.\text{iterate } n\ G\ (\lambda p.\text{succ } p)$$

avec

$$G = \lambda u n.\text{iterate } n\ u\ (u\ 1)$$

$\rightarrow$ G est une fonctionnelle

Chapitre 15

Combinateur de point fixe

Définition: $Y = \lambda f.(\lambda x.f(xx))(\lambda x.f(xx))$

Lemme: $Y =_\beta \lambda f.f(Yf)$

Corollaire: $Y\ f$ vérifie l'équation

$$Y\ f =_\beta f(Y\ f)$$

soit $Y\ f = f(f(f...f(f...$

Remarque: $Y \xrightarrow{*} \lambda f.f(f...f(f...$

$\rightarrow$ on peut voir Y comme un entier de *Church* **infini**

Utilisation:
Quand une fonction g est définissable comme solution d'une équation

$$g = F\ g$$

où F est une fonctionnelle appropriée, alors on définit g en λ-calcul par

$$g = Y\ F$$

Exemple: Fonction d'Ackermann

$$\begin{array}{l} F(A) = \\ \lambda np.\text{if } n = 0 \left| \begin{array}{l} \text{then succ } p \\ \text{else if } p = 0 \left| \begin{array}{l} \text{then } A(n-1)\ 1 \\ \text{else } A\ (n-1)\ (A\ n\ (p-1)) \end{array} \right. \end{array} \right. \end{array}$$

On pose ensuite :

$$A_\bullet = Y\ F$$

et on a bien $A_\bullet$ qui vérifie les équations de la fonction d'Ackermann.

Cette documentation a été réalisée sous Linux Slackware 8.0 à l'aide de LaTeX et cvs.