

Chapitre 1

Introduction au λ -calcul

Le λ -calcul a été inventé par Alonzo Church en 1932 (il était déjà présent en germe dans le travail de Frege au début du siècle). Le but de Church était de définir la notion de calculabilité effective au moyen de la λ -définissabilité. Plus tard, il apparut que la notion ainsi introduite était équivalente aux notions de calculabilité au sens de Turing (machine de Turing) et de Gödel-Herbrand (fonctions récursives). Cette coïncidence incite à penser qu’il existe une notion de calculabilité universelle, indépendante des formalismes particuliers : c’est la thèse de Church.

Le but de cette section est de présenter brièvement le λ -calcul, qui sert de base théorique à tout langage fonctionnel bien conçu. Plus qu’un cours formel, il s’agit d’une introduction qu’on espère motivante et qui pourra inciter le lecteur intéressé à consulter la littérature pour plus de détails (voir bibliographie). C’est pourquoi on n’y trouvera guère de démonstrations.

1.1 Définition et α -conversion

Commençons par donner une intuition. Le λ -calcul est le calcul de la fonctionnalité pure. Il n’y a donc que deux opérations : la formation d’une fonction à partir d’un terme et l’application d’une fonction à un argument. En mathématiques, on dit couramment “considérons la fonction qui à x associe $x + 3$ ”, ce que Bourbaki note $x \mapsto x + 3$. Le λ -calcul introduit la notation $\lambda x.x + 3$ pour cette opération. Dans la suite, le calcul considéré sera *pur* et ne contiendra donc aucune constante (comme ici 3 et +).

On se donne un ensemble V infini dénombrable de variables. On définit les termes inductivement comme suit :

- Si $x \in V$ alors x est un terme
- Si $x \in V$ et M est un terme, alors $\lambda x.M$ est un terme ; c’est la fonction qui, à x , associe M (qui, en général, dépend de x). On dit que la variable x est abstraite dans M .
- Si M et N sont des termes, alors MN est un terme. C’est l’application de M (à considérer comme une fonction) à N (à considérer comme son

argument).

Tout de suite quelques exemples :

- L'identité : $\lambda x.x$, ou bien $\lambda y.y$ ou bien...
- On peut l'appliquer à elle-même : $(\lambda x.x)\lambda x.x$.
- $K = \lambda x.\lambda y.x$

Pour alléger les notations, on écrira souvent $\lambda x_1 x_2 \dots x_n.M$ au lieu de $\lambda x_1.\lambda x_2.\dots\lambda x_n.M$, et $M_1 M_2 \dots M_k$ au lieu de $(\dots((M_1 M_2) M_3) \dots) M_k$. On remarquera que ce sont les conventions habituelles de CAML.

La substitution aux occurrences libres d'une variable x d'un terme M par un terme N sera notée $M[N/x]$ (où N écrase s).

Définition 1 Pour tout terme N, M et pour n'importe quelle variable x , le résultat $(M[N/x])$ de la substitution de toutes les occurrences libres de x dans M par N est définie de la manière suivante :

1. $x[N/x] \equiv N$
2. $y[N/x] \equiv y$ pour tout $y \neq x$
3. $(M_1 M_2)[N/x] \equiv (M_1[N/x])(M_2[N/x])$
4. $(\lambda x.Y)[N/x] \equiv \lambda x.Y$
5. $(\lambda y.Y)[N/x] \equiv (\lambda y.Y[N/x])$ si $y \neq x$, et $y \notin N$ ou $x \notin Y$
6. $(\lambda y.Y)[N/x] \equiv (\lambda z.Y[z/y][N/x])$ si $y \neq x$ et $y \in N$ et $x \in Y$ où z est une nouvelle variable.

On remarque dans ces exemples que le nom des variables liées dans un terme n'a aucune importance. Cela se traduit par la règle d' α -conversion :

$$\lambda x.M \equiv \lambda y.M[y/x]$$

si x n'est pas liée dans M et y ne figure pas libre ou liée dans M . L'importance de cette règle apparaîtra avec la β -conversion.

Exemples :

- $\lambda x.xy \equiv \lambda z.zy$ mais $\lambda x.xy \not\equiv \lambda y.yy$.
- $\lambda x.yx \equiv \lambda z.yz$ mais $\lambda x.yx \not\equiv \lambda y.yy$

1.2 β -réduction

C'est la règle fondamentale du λ -calcul. Un terme de la forme $(\lambda x.M)N$ est appelé *redex*. Voici la réduction d'un redex :

$$(\lambda x.M)N \rightarrow_0 M[N/x]$$

à condition que x n'apparaisse pas libre dans N (si c'est le cas, faire une α conversion sur $\lambda x.M$) et qu'aucune variable libre de N ne soit capturée dans M (là aussi faire une α conversion sur M). Le terme de droite est appelé *réduit*.

Ce qu'on appellera β -réduction, c'est la réduction d'un redex à l'intérieur d'un terme. Plus précisément, c'est la relation de réduction \rightarrow définie par

1.3. EXEMPLE 1. REPRÉSENTATION DU CALCUL PROPOSITIONNEL 3

- Si $M \rightarrow_0 M'$ alors $M \rightarrow M'$.
- Si $M \rightarrow M'$ alors $\lambda x.M \rightarrow \lambda x.M'$.
- Si $M \rightarrow M'$ alors $MN \rightarrow M'N$.
- Si $N \rightarrow N'$ alors $MN \rightarrow MN'$.

C'est grâce à cette règle que le λ -calcul est véritablement un calcul. Voici des exemples.

1.3 Exemple 1. Représentation du calcul propositionnel

Nous sommes habitués à représenter les valeurs de vérité vrai et faux par 1 et 0 (si nous représentons les connecteurs comme des opérations arithmétiques dans l'algèbre booléenne), comme nil et t en lisp, comme 0 et n'importe quelle autre valeur retournée par une fonction en C.

En λ -calcul pur nous pouvons les représenter par des λ -termes. Une représentation possible est la suivante :

- $T = \lambda xy.x$
- $F = \lambda xy.y$

On comprendra la grande habileté de ce choix en construisant un terme qui représente l'opérateur conditionnel. Nous pouvons le concevoir comme une fonction qui prend comme arguments une condition c et deux expressions e_1 et e_2 . Cette fonction doit retourner e_1 , si c évalue à T, e_2 si c évalue à F, ce qui conduit à la définition suivante :

- **cond** = $\lambda ce_1e_2.((c e_1) e_2)$

Essayons. Soient E1 et E2 deux λ -termes quelconques.

$$\begin{aligned} &(((\lambda ce_1e_2.((c e_1) e_2) T) E1) E2) \\ &\quad \rightarrow ((T E1)E2) \\ &\quad = ((\lambda xy.x E1)E2) \\ &\quad \rightarrow (\lambda y.E1)E2 \\ &\quad \rightarrow E1 \end{aligned}$$

Naturellement on peut, pour se faciliter la vie, se définir des abbréviations comme la suivante :

- **if c then e1 else e2** = **cond c e1 e2**

Exercice : définir les autres connecteurs au moyen de λ -termes et démontrer les lois de De Morgan.

1.4 Exemple 2. Comment représenter le couple en λ -calcul

La structure de données couple est “quelque chose” qui, étant donnés deux objets a et b , contient ces deux objets et la possibilité de les “récupérer”. Supposons avoir défini une fonction **cons** qui, à partir de a et de b nous rend le couple (a, b) . Il faudra que l’objet “rendu” par **cons** nous permette de lui appliquer deux fonctions, **fst** et **snd** qui vérifient les équations suivantes :

$$\begin{aligned}fst (cons a b) &= a \\snd (cons a b) &= b\end{aligned}$$

Pour la fonction **cons** nous pouvons donner la définition :

– **cons** = $\lambda xy.f x y$

Appliquée à A et B elle donne :

$$\begin{aligned}cons A B &= \lambda xyf.((f x)y)AB \\&\rightarrow \lambda f.fAB\end{aligned}$$

Or le terme $\lambda f.f A B$ est capable de capturer des termes qui remplaceront f . En particulier nous pouvons remplacer f par des fonctions **first** et **second** capables de sélectionner le premier ou le deuxième parmi les arguments auxquelles elles sont appliquées :

– **first** = $\lambda xy.x$

– **second** = $\lambda xy.y$

fst et **snd** seront ainsi définies :

– **fst** = $\lambda x.(x \text{ first})$

– **snd** = $\lambda x.(x \text{ second})$

1.5 Exemple 3. Deux représentations des entiers

Il existe plusieurs codages des entiers dans le λ -calcul. Le premier, du à Church, fut présenté dans les années 1931-32.

1.5.1 Les numéraux de Church

Un nombre est représenté comme un terme qui représente n applications successives d’une fonction à un argument (si le nombre en question est n). Le nombre n sera représenté par $\lambda fx.f^n x$

– $\bar{0} = \lambda fx.x$

– $\bar{1} = \lambda fx.fx$

– $\bar{n} = \lambda fx.f^n x$ ce qui équivaut à $\lambda fx.(f(f \dots (f x) \dots))$

La fonction successeur, σ , doit prendre un numéral n de la forme $\lambda f x. f^n x$ et rendre $\lambda f x. f(f^n x)$. On peut “ouvrir” le numéral \bar{n} en l’appliquant à deux arguments quelconques, par exemple à f et à x :

$$(\lambda f x. f^n x) f x \rightarrow f^n x$$

Pour obtenir le successeur de \bar{n} il faut donc “capturer” n , l’ouvrir, lui ajouter un f en tête, et “laisser”, à la tête de tout ça un nouveau $\lambda f x$.

Ceci est fait par la fonction :

$$- \sigma = \lambda n f x. f(n f x)$$

Nous pouvons à l’aide de σ définir des simples fonctions comme la somme

$$- \text{add} = \lambda m n. m \sigma n$$

D’autres fonctions, comme la fonction prédécesseur sont toutefois difficiles à définir avec les numéraux de Church. La récursion est aussi assez laborieuse.

1.5.2 Une autre représentation des entiers

Cette représentation des entiers, qui vient de Barendregt, nous permettra d’utiliser le calcul sur les booléens que nous avons déjà défini.

Soient :

$$- \bar{0} = \lambda x. x$$

$$- \sigma = \lambda n. \lambda f. ((f F) n)$$

Si nous appliquons un numéral \bar{n} ($\bar{n} = \sigma^n \bar{0}$) à T (i.e. $\lambda x y. x$) nous obtenons de façon triviale T ou F , selon que \bar{n} est $\bar{0}$ ou un autre numéral :

$$\begin{aligned} \bar{0} T &= (\lambda x. x \lambda x y. x) \\ &\rightarrow \lambda x y. x \\ &= T \end{aligned}$$

Par contre

$$\begin{aligned} \bar{n} T &= (\lambda f. ((f F) \overline{n-1}) (\lambda x y. x)) \\ &\rightarrow ((\lambda x y. x F) \overline{n-1}) \\ &\rightarrow (\lambda y. F \overline{n-1}) \\ &\rightarrow F \end{aligned}$$

Définissons en outre :

$$- \text{is-zero} = \lambda n. (n \text{ first})$$

$$- \text{pred1} = \lambda n. (n \text{ second})$$

$$- \text{pred} = \lambda n. (\text{if } (\text{is-zero } n) \text{ then } \bar{0} \text{ else } (\text{pred1 } n))$$

1.6 La récursion

En essayant de définir l'addition, on serait tenté de la définir de la façon suivante :

– $\text{add} = \lambda mn. \text{if } (\text{iszero } n) \text{ then } m \text{ else } \sigma(\text{add } m \text{ (pred } n))$

Or ceci ne marche pas, car “add” est purement une abbréviation, si bien qu'il faudrait remplacer son occurrence dans le corps de la définition par la définition elle-même, qui contient une occurrence de “add” qui doit être remplacée ...et ainsi de suite.

La solution est donnée par l'opérateur de *point fixe* Y , qui a la propriété suivante : pour toute fonction F ,

$$(Y F) \cong F(Y F)$$

où \cong est l'équivalence engendrée par \rightarrow , appelée β -conversion. Plusieurs opérateurs, munis de cette propriété, ont été définis. Par exemple :

– $Y = (\lambda f.(\lambda s.f (s s)) (\lambda s.f (s s)))$

La somme sera alors définie en faisant d'abord une abstraction sur le symbole *add* dans la définition précédente, puis en appliquant Y :

– $\text{add1} = \lambda fmn. \text{if } (\text{iszero } n) \text{ then } m \text{ else } \text{succ}(f m \text{ (pred } n))$

– $\text{add} = Y \text{ add1}$

Exercice : démontrer que $(Y F) \cong F(Y F)$.

Exemple :

$$\begin{aligned} \text{add } 9 \ 1 &\cong (Y \text{ add1}) \ 9 \ 1 \\ &= (\lambda f.(\lambda s.f (s s)) (\lambda s.f (s s))) \ \text{add1} \ 9 \ 1 \\ &\rightarrow (\lambda s.\text{add1} (s s)) (\lambda s.\text{add1} (s s)) \ 9 \ 1 \\ &\rightarrow \text{add1} ((\lambda s.\text{add1} (s s))(\lambda s.\text{add1} (s s))) \ 9 \ 1 \\ &= \text{add1} (Y \text{ add1}) \ 9 \ 1 \\ &= (\lambda fmn.\text{if } (\text{iszero } n) \text{ then } m \text{ else } \text{succ}(f m \text{ (pred } n))) (Y \text{ add1}) \ 9 \ 1 \\ &\rightarrow \text{if}(\text{iszero } 1) \text{ then } 9 \ \text{else } \text{succ} ((Y \text{ add1}) \ 9 \ 0) \\ &\rightarrow \text{succ } \text{add1} (Y \text{ add1}) \ 9 \ 0 \\ &\rightarrow \text{succ} (\text{if } (\text{iszero } 0) \text{ then } 9 \ \text{else } \text{succ}((Y \text{ add1}) \ 9 \ \text{pred } 0)) \\ &\rightarrow \text{succ } 9 \\ &\rightarrow 10 \end{aligned}$$

1.7 Les théorèmes du λ -calcul

Voici le premier théorème de ce cours. Il affirme que ce qu'on vient de définir est bien un calcul, au sens où le résultat ne dépend pas de l'ordre des

réductions. C'est la moindre des chose ; par exemple, dans l'arithmétique, quand on calcule $(2 + 3) * (1 + 5)$, le résultat n'est pas différent si vous réduisez d'abord $2 + 3$ en 5 ou $1 + 5$ en 6.

On note \rightarrow^* la fermeture réflexive-transitive de la relation de conversion \rightarrow définie précédemment. (Donc $M \rightarrow^* N$ si et seulement si il existe une suite finie (M_1, \dots, M_n) avec $n \geq 1$ telle que $M = M_1$, $N = M_n$ et $M_1 \rightarrow M_2 \cdots M_n$.)

Théorème 1 (*Church-Rosser*) *Soit M un λ -terme. Soient M_1 et M_2 des termes tels que $M \rightarrow^* M_1$ et $M \rightarrow^* M_2$. Il existe un terme N tel que $M_1 \rightarrow^* N$ et $M_2 \rightarrow^* N$.*

On ne donnera pas la preuve qui est plutôt technique et ne présente pas beaucoup d'intérêt en soi (pour le non spécialiste). Ce théorème a une conséquence intéressante.

Définition 2 *On dit qu'un terme est en forme normale s'il ne possède aucun redex. On dit qu'un terme a une forme normale, ou qu'il est normalisable, s'il existe une réduction de ce terme (pour \rightarrow^*) qui mène à une forme normale. On dit qu'il est fortement normalisable si toute réduction pour \rightarrow^* mène à une forme normale.*

Alors, bien sûr

Théorème 2 *Si un terme est normalisable, il a une unique forme normale.*

C'est une conséquence immédiate du théorème de Church-Rosser. (Pourquoi ?)

Remarquons qu'il existe des termes non normalisables : soit $\Omega = (\lambda x.xx)(\lambda x.xx)$.

On a

$$\Omega \rightarrow_0 \Omega \rightarrow_0 \Omega \rightarrow_0 \cdots$$

Il existe aussi des termes normalisables qui ne sont pas fortement normalisables. C'est le cas le $(\lambda xy.y)\Omega$. Si l'on choisit de réduire d'abord le redex qui se trouve à l'intérieur de Ω , et de continuer ensuite dans cette voie, on est perdu. Sinon, c'est-à-dire si l'on réduit le "redex de tête" (qui est le terme lui-même), on a gagné, car Ω disparaît. Cet exemple laisse entrevoir que le choix de la stratégie de réduction est crucial en λ -calcul.

1.8 La réduction standard du λ -calcul

On a vu précédemment qu'un terme normalisable n'est pas nécessairement fortement normalisable. Il existe de "mauvaises" stratégies qui, sur un terme donné, ne terminent pas alors que d'autres terminent. On donne ici une stratégie qui termine sur tout terme normalisable. Si elle est sûre, elle n'est généralement pas optimale, hélas.

On remarque facilement que tout λ -terme M est de la forme

$$\lambda x_1 x_2 \cdots x_n. M_1 M_2 \cdots M_k$$

où M_1 est soit une variable, soit une abstraction.

Définition 3 *Si M_1 est une variable x , cette variable est appelée variable de tête de M . On dit alors que M est en forme normale de tête.*

Dans le cas contraire, on a $M_1 = \lambda y. N_1$, et le redex $(\lambda y. N_1) M_2$ est appelé “redex de tête” de M . La stratégie standard consiste

- dans le cas où il y a un redex de tête, à le réduire,
- et si, par contre, M est en forme normale de tête, à appliquer la réduction standard à chacun des M_2, \dots, M_k (ce sont des réductions indépendantes; on pourrait les mener en parallèle).

Voici (sans démonstration) le résultat annoncé :

Théorème 3 *Appliquée à un terme normalisable, la réduction standard termine.*

Deux situations sont possibles, quand on applique la réduction standard à un terme :

- on aboutit à une forme normale de tête. Le terme est alors dit solvable.
- Cela ne termine pas, et il n’y a jamais de forme normale de tête. Terme non solvable. C’est le cas de Ω .

Un terme solvable n’est pas forcément normalisable; il se peut que la réduction standard produise une suite infinie de formes normales de têtes. On peut alors voir chaque variable de tête successive comme un “bout d’information” sur une donnée infinie qu’on n’atteindra jamais dans sa totalité. Exemple : soit $A = \lambda x y. y(xx)$ alors le terme AA est de cette sorte, puisqu’en effet :

$$AA \rightarrow \lambda y. y(AA) \rightarrow \lambda y. y(\lambda y. y(AA)) \rightarrow \dots$$

et on produit ainsi la liste des variable de tête y, y, \dots

Pour finir, voici un exemple qui montre que la réduction standard n’est pas optimale en général. Soit $I = \lambda x. x$ l’identité. On considère $B = (\lambda y. yy)(II)$. Si on applique la stratégie standard :

$$B \rightarrow II(II) \rightarrow I(II) \rightarrow II \rightarrow I$$

mais on aurait pu réduire d’abord à droite (le redex II) dans B , ce qui aurait donné par exemple :

$$B \rightarrow (\lambda y. yy)I \rightarrow II \rightarrow I$$

soit trois étapes au lieu de quatre.

1.9 Exercices

1. forme normale

Soient les termes suivants :

$$A = \lambda xy.xy \quad \Delta = (\lambda x.xx)$$

$$K = \lambda xy.x \quad S = \lambda xyz.xz(yz)$$

$$B = \lambda xyz.(x (y z)) \quad C = \lambda xyz.(x z y)$$

Donner si elle existe la forme normale des termes suivants :

– ΔA

– $I = SKK$

– $B = S(KS)K$

– $\Omega = \Delta\Delta$

– $C = S(BBS)(KK)$

2. Trouver les λ -termes qui calculent la multiplication et l'exponentielle pour les entiers de Church.

1.10 Bibliographie

- Alonzo Church, "The Calculi of lambda-conversion", Princeton University Press

Le λ -calcul a été présenté par Alonzo Church dans cet ouvrage qui est un texte tout à fait lisible et utile pour comprendre les idées de base et les connections entre λ -calcul, récursivité et théorie de la calculabilité.
- Barendregt, "The lambda-calculus : its syntax and semantics" North Holland, 1984.

est un peu la bible du λ -calcul, mais c'est assez difficile à lire.
- Hindley and Seldin. "Introduction to Lambda-Calculus and combinators" Cambridge University Press, 1986

est une excellente introduction, qui traite très bien aussi le λ -calcul typé.
- Strachey, "Denotational Semantic", MIT Press, 1977

Un texte théorique mais facile et important sur les liens entre λ -calcul et informatique.
- Chantal Berline, 2001-2002, une introduction au λ -calcul, en ligne <http://www.pps.jussieu.fr/~berline/Cours.html>.

très complet. On y trouve les démonstrations de base. Cours donné au DEA de logique.