Expression Logique et Fonctionnelle . . . Évidemment

Cours n° 4 : λ-calcul

Ces notes de cours sont inspirées du livre *An introduction to lambda calculi for computer scientists* de Chris Hankin (King's College London Publications, 2004), ainsi que des notes *Introduction to Lambda Calculus* de Henk Barendregt et Erik Barendsen, (`ftp://ftp.cs.ru.nl/pub/CompMath.Found/lambda.pdf`).

1 Introduction

Le λ-calcul a été créé dans les années 1930 par Alonzo Church, pour répondre (négativement) à une question de mécanisation (Entscheidungsproblem) des mathématiques, posée en 1928 par David Hilbert. Il est à noter qu'Alan Turing a simultanément apporté la même réponse au même problème en introduisant un modèle théorique de machine, les machines de Turing. Il s'avère que les deux formalismes, λ-calcul et machines de Turing, permettent de définir la même classe de fonctions.

Le λ-calcul sert aujourd'hui de base théorique à la programmation fonctionnelle, tout comme le calcul des prédicats sert de fondement à la programmation logique.

2 Définitions de base

2.1 Les λ-termes

Alphabet $\mathcal{A}$ des λ-termes :
- un symbole λ ;
- les parenthèses (et) ;
- le point . et l'espace [1] ;
- un ensemble $\mathcal{V}$ (infini dénombrable) de *variables* : x, y, z, ...

Les λ-termes sont des mots construits à partir de ces symboles en suivant les règles suivantes exprimées sous forme de grammaire BNF.

⟨λ-terme⟩	::=	⟨variable⟩	(variable)
	|	(λ⟨variable⟩.⟨λ-terme⟩)	(abstraction)
	|	(⟨λ-terme⟩ ⟨λ-terme⟩)	(application)
⟨variable⟩	::=	$x \mid y \mid z \ \ldots$	

L'ensemble Λ des λ-termes est donc le plus petit sous-ensemble de $\mathcal{A}^*$

1. contenant toutes les variables : $\mathcal{V} \subset \Lambda$;
2. fermé pour l'opération d'abstraction :

$$\forall M, x \ (M \in \Lambda \ \wedge \ x \in \mathcal{V}) \Rightarrow (\lambda x.M) \in \Lambda \ ;$$

3. et fermé pour l'opération d'application :

$$\forall M_1, M_2 \ (M_1 \in \Lambda \ \wedge \ M_2 \in \Lambda) \Rightarrow (M_1 \ M_2) \in \Lambda.$$

Exemple 1 :
Voici quelques exemples de λ-termes : x, $(x\ z)$, $((x\ z)\ (y\ z))$, $(\lambda x.z)$, $(\lambda x.(\lambda y.(\lambda z.((x\ z)\ (y\ z)))))$.

L'intuition qui se cache derrière ce formalisme est qu'une abstraction correspond à la définition d'une fonction, et qu'une application correspond à l'application d'une fonction à un argument. En particulier, une abstraction $\lambda x.M$ peut être vue comme l'équivalent en λ-calcul de la forme **`function x -> M`** en CAML.

1. Dans certaines présentations du λ calcul, ces deux symboles sont parfois omis. Nous les mettons pour plus de lisibilité.

Convention : On écrira $M \equiv N$ pour signifier que M et N sont (syntaxiquement) les mêmes termes ou pour désigner des abbréviations comme la suppression de certaines parenthèses.

$$\begin{aligned}(M\ N) &\equiv M\ N\\ ((M\ N)\ P) &\equiv M\ N\ P\\ (\ldots((M_1\ M_2)\ M_3)\ldots M_n) &\equiv M_1\ M_2\ M_3\ \ldots\ M_n\\ (\lambda x.M) &\equiv \lambda x.M\\ (\lambda x_1.(\lambda x_2.(\ldots(\lambda x_n.M)))) &\equiv \lambda x_1 \ldots x_n.M.\end{aligned}$$

Mais attention, il est des contextes dans lesquels les parenthèses sont nécessaires.

$$\begin{aligned}(M\ (N\ P)) &\not\equiv M\ N\ P\\ \lambda x.(M\ N) &\not\equiv \lambda x.M\ N.\end{aligned}$$

2.2 Variables libres, liées

Dans un λ-terme, une variable est dite *libre* si elle n'est pas sous la portée d'un λ. Plus formellement, l'ensemble $FV(M)$ des variables libres d'un λ-terme M est défini par induction sur la structure de M par les règles :

$$\begin{aligned}FV(x) &= \{x\}\\ FV(\lambda x.M) &= FV(M) \setminus \{x\}\\ FV(M\ N) &= FV(M) \cup FV(N).\end{aligned}$$

Lorsque $FV(M) = \emptyset$ on dit que le terme est *clos* ou encore un *combinateur*. Par exemple, $\lambda x.x$ est un terme clos.

Une variable est dite *liée* si elle est sous la portée d'un λ. L'ensemble $BV(M)$ des variables liées d'un λ-terme M se définit par induction avec les règles :

$$\begin{aligned}BV(x) &= \emptyset\\ BV(\lambda x.M) &= BV(M) \cup \{x\}\\ BV(M\ N) &= BV(M) \cup BV(N).\end{aligned}$$

Attention, une même variable peut être à la fois libre et liée dans un même λ-terme.

Exemple 2 :
Avec $M = (x\ (\lambda x.(x\ x)\ y))$, on a
- $FV(M) = \{x, y\}$
- et $BV(M) = \{x\}$.

La variable x est donc à la fois libre et liée.

2.3 Sous-termes

Un *sous-terme* d'un λ-terme est un facteur de ce λ-terme qui est lui-même un λ-terme. L'ensemble $Sub(M)$ des sous-termes d'un λ-terme M se définit inductivement par les règles :

$$\begin{aligned}Sub(x) &= \{x\}\\ Sub(\lambda x.M) &= Sub(M) \cup \{\lambda x.M\}\\ Sub(M\ N) &= Sub(M) \cup Sub(N) \cup \{(M\ N)\}.\end{aligned}$$

Exemple 3 :
Avec $M = (x\ (\lambda x.(x\ x)\ y))$, on a

$$Sub(M) = \{x, y, (x\ x), \lambda x.(x\ x), (\lambda x.(x\ x)\ y), (x\ (\lambda x.(x\ x)\ y))\}.$$

2.4 Contextes

Intuitivement, un contexte est un λ-terme avec un ou plusieurs « trous ». Si on remplit ces trous avec un λ-terme, on obtient un λ-terme.

Formellement l'ensemble $\mathcal{C}[]$ des λ-contextes est un ensemble de mots construits sur le même alphabet $\mathcal{A}$ que les λ-termes auquel on ajoute un symbole $[]$ (le trou). Sa définition inductive est donnée par les règles :

1. $\mathcal{V} \subset \mathcal{C}[]$;
2. $[] \in \mathcal{C}[]$;
3. si $x \in \mathcal{V}$ et $C[] \in \mathcal{C}[]$ alors $(\lambda x.C[]) \in \mathcal{C}[]$;
4. si $C_1[] \in \mathcal{C}[]$ et $C_2[] \in \mathcal{C}[]$, alors $(C_1[]\ C_2[]) \in \mathcal{C}[]$.

Exemple 4 :

1. Tous les λ-termes sont des contextes (sans trous)
$$\Lambda \subset \mathcal{C}[].$$
2. $\lambda x.([]\ x)M$ est un λ-contexte avec un trou.
3. $[]\ (\lambda x.([][])\ y)$ est un λ-contexte avec trois trous.

Si M est un λ-terme et $C[]$ un λ-contexte, la notation $C[M]$ désigne le λ-terme obtenu en remplaçant toutes les occurrences des trous par le terme M.

Exemple 5 :
Avec $C[] = \lambda x.([]\ x)$ et $M = \lambda y.y$, on a
$$C[M] = \lambda x.(\lambda y.y\ x).$$

À noter que dans cette opération de remplissage de trous, des variables libres de M peuvent devenir liées. C'est le cas avec le contexte de l'exemple précédent et $M = x$.

2.5 Substitutions

L'opération de substitution d'une variable par un terme N dans un terme M consiste à remplacer toutes les occurrences libres de la variable par le terme N. On note $M[x \leftarrow N]$ le terme ainsi obtenu.

Cette opération peut être définie précisément inductivement par les règles

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

3 La théorie du λ-calcul

Il s'agit ici de définir une relation entre les termes du λ-calcul qui intuitivement exprime que ces termes ont mêmes « valeurs ».

Nous noterons $M = N$ cette relation que nous définirons par des règles de déduction.

La première de ces règles, nommée β, donne le sens intuitif de l'application d'une abstraction à un terme.

$$\frac{}{(\lambda x.M)\ N = M[x \leftarrow N]}\ (\beta)$$

Cette règle sans hypothèse peut être considérée comme un schéma d'axiomes.

Les trois règles qui suivent font que la relation $=$ est une relation d'équivalence comme il se doit.

$$\frac{}{M = M}\ (\text{réflexivité}) \qquad \frac{M = N}{N = M}\ (\text{symétrie}) \qquad \frac{M = N \qquad N = P}{M = P}\ (\text{transitivité})$$

La règle qui suit, nommée α-conversion porte sur le changement de noms des variables liées.

$$\frac{}{(\lambda x.M) = \lambda y.M[x \leftarrow y]}\ (\alpha)$$

Enfin les trois règles qui suivent sont les règles de compatibilité.

$$\frac{M = N}{M\ Z = N\ Z}\ (\text{appl 1}) \qquad \frac{M = N}{Z\ M = Z\ N}\ (\text{appl 2}) \qquad \frac{M = N}{\lambda x.M = \lambda x.N}\ (\text{abstr})$$

Une conséquence de ces règles de compatibilité est que pour tout contexte $C[]$, si $M = N$ alors $C[M] = C[N]$.

On note

$$\lambda \vdash M = N$$

pour exprimer qu'on peut prouver que $M = N$ avec ces règles.

Exemple 6 :
Définissons les combinateurs

$$\begin{aligned} \mathbf{I} &\equiv \lambda x.x \\ \mathbf{K} &\equiv \lambda xy.x \\ \mathbf{K}_* &\equiv \lambda xy.y \\ \mathbf{S} &\equiv \lambda xyz.xz(yz). \end{aligned}$$

Alors pour tout λ-terme M, N et P, on a :

$$\begin{aligned} &\lambda \vdash \mathbf{I}\ M = M \\ &\lambda \vdash \mathbf{K}\ M\ N = M \\ &\lambda \vdash \mathbf{K}_*\ M\ N = N \\ &\lambda \vdash \mathbf{S}\ M\ N\ P = M\ P\ (N\ P) \end{aligned}$$

Exemple 7 :
Il existe un λ-terme G tel que pour tout λ-terme M, on a

$$\lambda \vdash G\ M = M\ M.$$

Il suffit de prendre $G \equiv \lambda x.(x\ x)$.

Théorème 1. Point fixe

1. *Pour tout terme $F \in \Lambda$, il existe un terme $X \in \Lambda$ tel que*

$$\lambda \vdash F\ X = X.$$

2. *Le combinateur*

$$\mathbf{Y} \equiv \lambda f.(\lambda x.(f\ (x\ x))\ \lambda x.(f\ (x\ x)))$$

est tel que pour tout terme $F \in \Lambda$, on a

$$\lambda \vdash F\ (\mathbf{Y}\ F) = \mathbf{Y}\ F.$$

Démonstration. 1. On pose $W \equiv \lambda x.(F\ (x\ x))$ et $X \equiv W\ W$. On a alors

$$X \equiv W\ W \equiv (\lambda x.F\ (x\ x))\ W.$$

Or

$$\frac{}{(\lambda x.F\ (x\ x))\ W = F\ (W\ W)}\ (\beta)$$

et $F\ (W\ W) \equiv F\ X$.

Donc le terme X ainsi défini est bien un point fixe de F.

2. On a

$$\mathbf{Y}\ F \equiv (\lambda f.(\lambda x.(f\ (x\ x))\ \lambda x.(f\ (x\ x))))\ F,$$

et

$$\frac{}{(\lambda f.(\lambda x.(f\ (x\ x))\ \lambda x.(f\ (x\ x))))\ F = \lambda x.(F\ (x\ x))\ \lambda x.(F\ (x\ x))}\ (\beta)$$

Or comme on l'a vu ci-dessus, en posant $W \equiv \lambda x.(F\ (x\ x))$, ce dernier terme est équivalent à $W\ W$ dont on sait qu'il est un point fixe de F. Par conséquent on a bien

$$\lambda \vdash F\ (\mathbf{Y}\ F) = \mathbf{Y}\ F.$$

□

Autrement dit tout terme du λ-calcul admet un point fixe, et le combinateur de point fixe appliqué à ce terme est équivalent à un de ses points fixes.

4 Le pouvoir d'expression du λ-calcul

On va voir qu'on peut assez naturellement représenter les données usuelles des langages de programmation comme les booléens, les entiers, les couples, par des λ-termes, et qu'on peut également représenter les fonctions « intuitivement » calculables [2].

4.1 Les booléens

Les booléens sont au nombre de deux, et sont classiquement utilisés dans des expressions booléennes, comme **`if cond then expr1 else expr2`** en CAML.

On peut définir les deux booléens en utilsant deux combinateurs précédemment définis :

– **True** $\equiv \mathbf{K}$,

– et **False** $\equiv \mathbf{K}_*$.

Avec cette définition, on peut alors définir un nouveau combinateur **if** qui représentera l'expression conditionnelle.

– **If** $\equiv \lambda bxy.(b\ x\ y)$.

Ces définitions se justifient si on remarque que pour tous termes B, M et N on a :

– $\lambda \vdash$ **If** $B\ M\ N = B\ M\ N$,

et si B est l'un des deux termes **vrai** ou **faux** on a bien les équations attendues :

– $\lambda \vdash$ **If True** $M\ N = M$,

– et $\lambda \vdash$ **If False** $M\ N = N$.

Les opérateurs logiques classiques sur les booléens peuvent être aussi représentés. Voici par exemple la conjonction :

– **And** $\equiv \lambda b_1 b_2.$(**If** $b_1\ b_2$ **False**).

4.2 Les couples

La représentation des couples nécessite non seulement un moyen de les construire à partir des deux λ-termes qui les composent, mais aussi un moyen d'exttraire l'une ou l'autre des deux composantes.

Nous noterons $[M, N]$ les couples de λ-termes comme un raccourci syntaxique pour

– $[M, N] \equiv \lambda b.$(**If** $b\ M\ N$)

– ou plus simplement $[M, N] \equiv \lambda b.(b\ M\ N)$.

Les deux projections qui permettent d'extraire chacune des deux composantes d'un couple sont définies par les combinateurs

– **Fst** $\equiv \lambda c.(c$ **True**),

– et **Snd** $\equiv \lambda c.(c$ **False**).

Ces définitions se justifient par le fait que pour tous termes M et N, si $C \equiv [M, N]$, on a

– $\lambda \vdash$ **Fst** $C = M$,

– $\lambda \vdash$ **Snd** $C = N$,

2. En fait, on peut montrer que toute fonction récursive au sens de Kleene peut être représentée en λ-calcul, et réciproquement. Il en est de même pour toute fonction calculable par une machine de Turing.

– et $\lambda \vdash [\mathbf{Fst}\ C, \mathbf{Snd}\ C] = C$.

On peut étendre la définition des couples pour définir des triplets (et plus généralement des n-uplets), en considérant qu'un triplet est un couple :

$$[M_1, M_2, M_3] \equiv [M_1, [M_2, M_3]].$$

4.3 Les entiers

Pour chaque entier $n \in \mathbb{N}$ on peut définir un λ-terme noté $\mathbf{c}_n$ le représentant par

– $\mathbf{c}_n \equiv \lambda fx.(f^n\ x)$,

où la notation $f^n\ x$ signifie $f\ (f\ (\ldots(f\ x)\ldots))$, avec n occurrences de f.

Les entiers 0 et 3 sont ainsi représentés par les λ-termes

– $\mathbf{c}_0 \equiv \lambda fx.x$,

– et $\mathbf{c}_3 \equiv \lambda fx.(f\ (f\ (f\ x)))$.

En fait cette représentation des entiers, due à Church, n'est autre qu'un itérateur de fonction.

La fonction successeur peut être représentée par le combinateur

– $\mathbf{S}^+ \equiv \lambda nfx.(n\ f\ (f\ x))$,

dans

Le prédicat **Zero** qui teste si un entier est nul se définit par

– $\mathbf{Zero} \equiv \lambda n.(n\ (\lambda x.\mathbf{False})\ \mathbf{True})$.

L'addition se définit par

– $\mathbf{Add} \equiv \lambda nm.(n\ \mathbf{S}^+\ m)$,

et la multiplication par

– $\mathbf{Mult} \equiv \lambda nmf.(n\ (m\ f))$.

Pour la soustraction, on va l'obtenir en itérant le terme $\mathbf{P}^-$ qui définit le prédécesseur. La définition du prédécesseur nécessite l'utilisation de couples. En effet on obtient le prédecesseur de l'entier $\mathbf{c}_n$ en itérant n fois la fonction $(a, b) \to (b, b+1)$ à partir du couple $(0, 0)$. Le prédécesseur de $\mathbf{c}_n$ est la première composante du couple résultant.

– $\mathbf{P}^- \equiv \lambda n.(\mathbf{Fst}(n\ (\lambda c.[\mathbf{Snd}\ c, \mathbf{S}^+\ (\mathbf{Snd}\ c)])\ [\mathbf{c}_0, \mathbf{c}_0]))$

La soustraction $n - m$ s'obtient alors en itérant m fois le prédécesseur sur n.

– $\mathbf{Sub} \equiv \lambda nm.(m\ \mathbf{P}^-\ n)$.

On peut noter que si $n \leq m$ alors

$$\lambda \vdash \mathbf{Sub}\ \mathbf{c}_n\mathbf{c}_m = \mathbf{c}_0.$$

On en déduit un terme pour représenter la comparaison $n \leq m$ de deux entiers :

– $\mathbf{Inf} \equiv \lambda nm.(\mathbf{Zero}\ (\mathbf{Sub}\ n\ m))$,

et un prédicat pour le test d'égalité de deux entiers :

– $\mathbf{Equal} \equiv \lambda nm.(\mathbf{And}\ (\mathbf{Inf}\ n\ m)\ (\mathbf{Inf}\ m\ n))$.

4.4 Quelles fonctions peut-on représenter en λ-calcul ?

4.5 Les fonctions primitives récursives

Peut-on représenter la fonction factorielle ?

En itérant n fois la fonction $(a, b) \to (a+1, (a+1)b)$ depuis le couple $(0, 1)$, alors la seconde composante du couple résultant est $n!$.

– $\mathbf{Fact} \equiv \lambda n.(\mathbf{Snd}\ (n\ (\lambda c.[\mathbf{S}^+\ (\mathbf{Fst}\ c), \mathbf{Mult}\ (\mathbf{S}^+\ (\mathbf{Fst}\ c))\ (\mathbf{Snd}\ c)])\ [\mathbf{c}_0, \mathbf{c}_1]))$.

4.6 Les fonctions récursives

Comment représenter en λ-calcul les fonctions récursives ?

Par exemple, la version récursive habituelle de la fonction factorielle serait

– $\mathbf{Fact} \equiv \lambda n.(\mathbf{If}\ (\mathbf{Zero}\ n)\ \mathbf{c}_1\ (\mathbf{Mult}\ n\ (\mathbf{Fact}\ (\mathbf{P}^-\ n))))$,

mais cela n'est pas possible en λ-calcul puisque un terme ne peut pas se contenir strictement (les λ-termes sont des mots finis).

En revanche, il est tout à fait possible de définir le terme $\mathbf{\Phi}_{fact}$ en ajoutant une abstraction :

– $\mathbf{Phi}_{fact} \equiv \lambda fn.(\mathbf{If}\ (\mathbf{Zero}\ n)\ \mathbf{c}_1\ (\mathbf{Mult}\ n\ (f\ (\mathbf{P}^-\ n))))$.

Si le terme **Fact** ci-dessus était définissable en λ-calcul, on aurait alors

$$\lambda \vdash \mathbf{\Phi}_{fact}\ \mathbf{Fact} = \mathbf{Fact},$$

autrement dit, il serait un point fixe du terme $\mathbf{\Phi}_{fact}$.

Or nous avons vu un moyen de calculer un point fixe pour chaque terme à l'aide du combinateur $\mathbf{Y}$. On peut alors définir le terme **Fact** par

– $\mathbf{Fact} \equiv \mathbf{Y}\ \mathbf{\Phi}_{fact}$.

Et on peut prouver que

$$\lambda \vdash \mathbf{Fact}\ \mathbf{c}_n = \mathbf{c}_{n!}.$$

Ce schéma est très général, et s'applique à toute fonction récursive. Ainsi le lambda-calcul permet d'exprimer toute fonction récursive.

5 Réduction

La théorie équationnelle du λ-calcul décrit les λ-termes équivalents. Cette théorie ne reflète pourtant l'idée intuitive de direction que prend un calul afin de réduire un terme pour en déterminer sa valeur.

Par exemple, il est vrai que

$$\lambda \vdash \mathbf{add}\ \mathbf{c}_n\ \mathbf{c}_m = \mathbf{c}_{n+m}.$$

Mais une intention cachée derrière cette égalité est que la « valeur » du terme de gauche de l'équation est le terme de droite et non le contraire. Autrement dit, on a intuitivement envie d'orienter les équations.

C'est ce que nous allons faire en définition les notions de β-réduction et de forme normale.

5.1 Relations de réduction

On va donc définir deux nouvelles relations binaires sur les λ-termes, notées $\rightarrow_\beta$ et $\twoheadrightarrow_\beta$.

1. La relation $M \rightarrow_\beta N$ entre deux λ-termes exprime que le premier terme M se réduit en le second en une étape de calcul qu'on appelle une étape de *β-réduction*.
   – $(\lambda x.M)\ N \rightarrow_\beta M[x \leftarrow N]$;
   – $M \rightarrow_\beta N \Rightarrow M\ P \rightarrow_\beta N\ P$, $P\ M \rightarrow_\beta P\ N$ et $\lambda x.M \rightarrow_\beta \lambda x.N$.
2. La relation $M \twoheadrightarrow_\beta N$ entre deux λ-termes exprime que le premier terme se réduit en le second en plusieurs étapes de calcul y compris aucune.
   – $M \twoheadrightarrow_\beta M$;
   – $M \rightarrow_\beta N \Rightarrow M \twoheadrightarrow_\beta N$;
   – $M \twoheadrightarrow_\beta N$, $N \twoheadrightarrow_\beta P \Rightarrow M \twoheadrightarrow_\beta P$.

Ces deux relations $\rightarrow_\beta$ et $\twoheadrightarrow_\beta$ sont compatibles avec les opérations d'application et d'abstraction. La relation $\twoheadrightarrow_\beta$ est réflexive et transitive (c'est la fermeture réflexive et transitive de la relation $\rightarrow_\beta$).

Exemple 8 :

– En posant $\mathbf{\Omega} \equiv (\lambda \mathbf{x}.(\mathbf{x}\ \mathbf{x}))(\lambda \mathbf{x}.(\mathbf{x}\ \mathbf{x}))$, on a

$$\Omega \rightarrow_\beta \Omega.$$

–

$$\mathbf{K}\ \mathbf{I}\ \mathbf{\Omega} \twoheadrightarrow_\beta \mathbf{I}.$$

– On a $\lambda \vdash \mathbf{K}\ \mathbf{I}\ \mathbf{\Omega} = \mathbf{I}\ \mathbf{I}$ mais on a ni $\mathbf{K}\ \mathbf{I}\ \mathbf{\Omega} \twoheadrightarrow_\beta \mathbf{I}\ \mathbf{I}$, ni $\mathbf{I}\ \mathbf{I} \twoheadrightarrow_\beta \mathbf{K}\ \mathbf{I}\ \mathbf{\Omega}$.

On appelle *redex* (*red*ucible *ex*pression) toute application d'une abstraction à un terme, c'est–à–dire tout terme de la forme $(\lambda x.M)\ N$. Il est clair qu'un redex se réduit en une étape

$$(\lambda x.M)\ N \rightarrow_\beta M[x \leftarrow N],$$

et qu'un terme est réductible si et seulement s'il contient un sous-terme qui est un redex.

Un terme *est en forme normale* s'il est irréductible, c'est–à–dire si aucun de ses sous-termes n'est un redex.

Enfin on dit qu'un terme *possède une forme normale* s'il est équivalent à un terme en forme normale. Formellement, M possède une forme normale N si

1. $M \twoheadrightarrow_\beta N$

2. et N est en forme normale.

Exemple 9 :

- Les combinateurs **I**, **K**, $\mathbf{K}_*$ et **S** sont en forme normale.
- Si x est une variable, et M un λ-terme en forme normale, alors les termes $x\ M$ et $\lambda x.M$ sont en forme normale.
- Les combinateurs $\mathbf{\Omega}$ et **Y** ne sont pas en forme normale, et n'en possèdent pas.

Le théorème de Church-Rosser indique que si deux réductions divergent à une étape donnée, il est toujours possible de les ramener au même terme par la suite.

Théorème 2. Church-Rosser
Pour tout terme $M \in \Lambda$, s'il existe deux termes $N_1, N_2 \in \Lambda$ tels que $M \twoheadrightarrow_\beta N_1$ et $M \twoheadrightarrow_\beta N_2$, alors il existe un terme $N_3 \in \Lambda$ tel que $M \twoheadrightarrow_\beta N_3$.

La figure 1 illustre ce théorème.

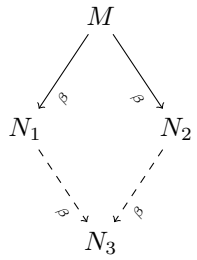

FIGURE 1 – Confluence de la relation $\twoheadrightarrow_\beta$ (théorème de Church-Rosser)

Corollaire 1. *1. Si N est une forme normale de M, alors $M \twoheadrightarrow_\beta N$.*
2. Un terme peut avoir au plus une seule forme normale.

5.2 Stratégie de réduction

Le terme $\mathbf{K\ I\ \Omega}$ est un exemple de terme ayant une forme normale (**I**) mais pour lequel il existe des chemins infinis de calcul.

Y a-t-il des stratégies de réduction menant à coup sûr à la forme normale d'un terme s'il en possède une. La stratégie de réduction du redex le plus à gauche en est une.

$$\begin{aligned}
\mathbf{K\ I\ \Omega} &=\equiv \underline{(\lambda xy.x)\ (\lambda x.x)}\ ((\lambda x.(x\ x))\ (\lambda x.(x\ x))) \\
&\to_\beta \underline{(\lambda y.(\lambda x.x))\ ((\lambda x.(x\ x))\ (\lambda x.(x\ x)))} \\
&\to_\beta \lambda x.x \\
&\equiv \mathbf{I}.
\end{aligned}$$

Une telle stratégie est déterministe, et le problème de la confluence n'en est plus un.

Si on note $\to_{\beta g}$ la relation de réduction du redex le plus à gauche, et $\twoheadrightarrow_{\beta g}$ sa clôture réflexive et transitive, alors on a le théorème de normalisation suivant.

Théorème 3. *Si M est un terme admettant N pour forme normale, alors*

$$M \twoheadrightarrow_{\beta g} N.$$

Corollaire 2. *Si pour un terme M la réduction $\twoheadrightarrow_{\beta g}$ ne mène à aucune forme normale (ie. si la réduction peut se poursuivre à l'infini), alors M ne possède pas de forme normale.*

Cette stratégie de réduction est celle adoptée par les langages de programmation fonctionnelle paresseux (comme HASKELL par exemple).

5.3 Indécidabilité de l'existence d'une forme normale

Théorème 4.
Il n'exite pas de combinateur $\mathbf{A}$ *tel que pour tout terme* M *ayant une forme normale, on ait*

$$\mathbf{A}\ M\ \twoheadrightarrow_\beta\ \mathbf{True},$$

et pour tout terme N *n'en ayant pas, on ait*

$$\mathbf{A}\ N\ \twoheadrightarrow_\beta\ \mathbf{False}.$$

Démonstration. Raisonnons par l'absurde et supposons qu'un tel terme existe.

Alors à partir de ce terme on peut construire les termes

$$\mathbf{\Phi}_{absurde} \equiv \lambda fx.(\mathbf{If}\ (\mathbf{A}\ (f\ x))\ (f\ x)\ \mathbf{True}),$$

et

$$absurde \equiv \mathbf{Y}\ \mathbf{\Phi}_{absurde}.$$

Soit M un terme quelconque. Le terme *absurde* M possède-t-il une forme normale ?

Calculons les premiers termes obtenus par réduction du redex le plus à gauche (si le terme a une forme normale, on l'obtient à coup sûr par cette stratégie).

$$\begin{aligned} absurde\ M &\equiv \mathbf{Y}\ \mathbf{\Phi}_{absurde}\ M \\ &\twoheadrightarrow_\beta\ \mathbf{\Phi}_{absurde}\ (\mathbf{Y}\ \mathbf{\Phi}_{absurde})\ M \\ &\twoheadrightarrow_\beta\ \mathbf{If}\ (\mathbf{A}\ (\mathbf{Y}\ \mathbf{\Phi}_{absurde}\ M))\ (\mathbf{Y}\ \mathbf{\Phi}_{absurde}\ M)\ \mathbf{True} \\ &\twoheadrightarrow_\beta\ (\mathbf{A}\ (\mathbf{Y}\ \mathbf{\Phi}_{absurde}\ M))\ (\mathbf{Y}\ \mathbf{\Phi}_{absurde}\ M)\ \mathbf{True} \end{aligned}$$

Deux cas de figures se présentent :

1. ou bien *absurde* M possède une forme normale, et alors selon notre hypothèse sur $\mathbf{A}$, on a

$$\mathbf{A}\ (absurde\ M)\ \twoheadrightarrow_\beta\ \mathbf{True}\ ;$$

2. ou bien *absurde* M ne possède pas de forme normale, et alors on a

$$\mathbf{A}\ (absurde\ M)\ \twoheadrightarrow_\beta\ \mathbf{False}.$$

Envisageons pour commencer la première de ces hypothèses, et poursuivons la réduction toujours avec la même stratégie.

$$\begin{aligned} absurde\ M\ &\twoheadrightarrow_\beta\ \mathbf{True}\ (\mathbf{Y}\ \mathbf{\Phi}_{absurde}\ M)\ \mathbf{True} \\ &\twoheadrightarrow_\beta\ \mathbf{Y}\ \mathbf{\Phi}_{absurde}\ M \\ &\equiv absurde\ M. \end{aligned}$$

On le voit si *absurdeM* possède une forme normale la stratégie de réduction du redex le plus à gauche mène en un nombre fini non nul d'étapes au terme initial. La réduction en suivant cette stratégie ne se finit donc jamais, et nous devons conclure que le terme *absurde* M ne possède pas de forme normale. Il n'est donc pas possible que ce terme possède une forme normale.

Envisageons maintenant l'autre hypothèse, à savoir *absurde* M ne possède pas de forme normale, et poursuivons la réduction du terme entamée ci-dessus.

$$\begin{aligned} absurde\ M\ &\twoheadrightarrow_\beta\ \mathbf{False}\ (\mathbf{Y}\ \mathbf{\Phi}_{absurde}\ M)\ \mathbf{True} \\ &\twoheadrightarrow_\beta\ \mathbf{True}. \end{aligned}$$

On constate que par réductions successives du redex le plus à gauche, le terme *absurde* M se réduit en le terme **True** qui est en forme normale. Or nous sommes partis de l'hypothèse que ce terme n'en avait pas. Cette hypothèse ne peut pas être logiquement admise.

Pour résumer, le terme *absurde* M ne peut logiquement pas avoir de forme normale, ni ne pas en avoir. Cette situation absurde résulte donc de notre hypothèse initiale, à savoir l'existence d'un terme $\mathbf{A}$ qui permette de distinguer les termes ayant une forme normale des autres.

Un terme tel que $\mathbf{A}$ ne peut donc pas exister. □