CALCULS ALGEBRIQUES ET FONCTIONNELS

De l'ordre supérieur au premier ordre

Année 2004-2005

Thérèse HARDIN-ACCART

Laboratoire d'Informatique de Paris 6 (LIP6)
Université Pierre et Marie Curie
8, Rue du Capitaine Scott, 75015 Paris
Therese.Hardin@lip6.fr
http:\\www-spi.lip6.fr/~hardin

Les notes qui suivent servent de base au cours du module de tronc commun "Lambda-calcul". En les rédigeant, j'ai d'abord cherché à donner une présentation intuitive des notions abordées, quitte à sacrfier parfois à une exposition complètement rogoureuse, qui pourra être trouvée dans les ouvrages donnés en référence. Ces notes ne constituent qu'une version provisoire. Toute remarque, suggestion est la bienvenue.

1 Cadre et Rappels

En guise de bilan, posons-nous la question suivante. A quoi sert le λ-calcul ?

Une première réponse est que le λ-calcul est le paradigme de la programmation, au moins pour les aspects fonctionnels. Cela s'illustre par l'étude de la définissabilité dans le λ-calcul.

Ce paradigme permet d'étudier différentes stratégies de calcul et d'obtenir les propriétés de Church-Rosser, de standardisation, des développements finis qui servent d'indicateurs pour les implantations des langages de programmation. L'ordre des réductions est sans importance dans la mesure où il ne compromet pas le calcul de la forme normale quand elle existe. La notion de "stratégie meilleure qu'une autre" ne peut avoir de sens que si on accepte de réécrire en parallèle les résidus des radicaux d'une même famille.

Considéré par un programmeur, un résultat comme la terminaison forte des calculs simplement typés n'est pas forcément positif puisqu'il interdit de fait les boucles while. Il met donc l'accent soit sur la nécessité d'ajouter un opérateur de point fixe ou sur la nécessité de disposer de types plus expressifs comme les Constructions, ces deux solutions ne rpéondant pas aux mêmes besoins.

Le λ-calcul fonde donc une partie des activités de compilation/interprétation et permet de décrire et d'étudier des méthodes comme les continuations, les machines abstraites, certaines optimisations.

Pour renforcer le côté paradigme des langages de programmation, il est commode d'enrichir le λ-calcul avec des structures de données, d'y modéliser les traits impératifs, les mécanismes d'exceptions, les traits objet et les mécanismes de modules. D'archétype des langages de programmation tel que nous le montre l'étude de la définissabilité, le λ-calcul devient ainsi langage de modélisation/simulation et donc un archétype des langages d'expression de sémantiques. C'est là que se greffe l'étude des modèles du λ-calcul avec d'abord l'étude des formes normales de tête puis des arbres de Böhm.

• Réécriture d'ordre supérieur, syntaxe abstraite d'ordre supérieur λ-calcul • primitives pour modéliser le synchronisme (Lucid Synchrone), la concurrence (Π-calcul de Milner, join-calcul de Gonthier, ambients de Cardelli,..), pour internaliser la notion de réécriture (ρ-calcul - C. Kirchner).

2 Logique combinatoire

Ces notes de cours sont établies à partir des ouvrages de Barendregt[1], Curien[8], Hindley[10]

La Logique Combinatoire a d'abord été introduite par Schöchnfinkel vers 1920 puis redécouverte par Curry. Il s'agit d'une théorie équationnelle du premier ordre, qui sert à décrire la fonctionnalité : on définit un ensemble d'opérateurs primitifs et les moyens de les composer. Il s'agit donc d'une théorie de la fonctionnalité où la notion de variable liée n'existe pas. Elle met l'accent sur le procédé opérationnel du calcul : on combine des éléments atomiques (S, K et I) pour décrire une fonction.

Définition 2.1 *La Logique Combinatoire, notée CL, est définie par la donnée d'un ensemble de variables V, d'un opérateur d'arité 2, noté par juxtaposition, et par trois constantes S, K et I. La*

théorie est engendrée par deux équations :

$$K\,x\,y \to x \quad S\,x\,y\,z \to (x\,z)(y\,z) \quad I\,x \to x$$

On peut aussi ne prendre que les constantes S et K et définir I par $I = S\,K\,K$.

Appelons *remplacement* la substitution du premier ordre et notons $a[b/x]$ le remplacement de b aux occurrences de x dans a.

On démontre sans difficulté, en utilisant simplement les résultats généraux sur la réécriture du premier ordre, les lemmes suivants:

Lemme 2.2 *Si $M \xrightarrow{\star} N$, alors :*

1. $A\{x_1/N_1, \ldots, x_p/N_p\} \xrightarrow{\star} B\{x_1/N_1, \ldots, x_p/N_p\}$
2. $PA\{x/N\} \xrightarrow{\star} PB\{x/N\}$

Théorème 2.3 *CL vérifie la propriété de Church-Rosser.*

En effet, ce système est orthogonal (i.e. linéaire gauche, sans paire critique) et donc confluent. On rappelle que la confluence d'un système orthogonal se démontre en prouvant que la réduction d'une famille de radicaux disjoints est fortement confluente.

2.1 La logique Combinatoire en tant que Théorie Logique

La logique Combinatoire en tant que Théorie Logique est définie par les termes de CL et les deux axiomes:

$$(C0) \quad Kxy = x \quad Sxyz = (xz)(yz)$$

Il s'agit donc d'une théorie équationnelle. On a donc :

$$(C0) \vdash \forall x.(([x]M)x = M)$$
$$(C0) \vdash \forall x.(([x]M)N = M\{x/N\})$$
$$(C0) \vdash ([x_1] \ldots [x_n]M)x_1 \ldots x_n = M$$

Cette propriété peut être exprimée par le schéma d'axiome:

$$(CC) \quad \exists M, \forall x_1, \ldots, x_p.(M\,x_1 \ldots x_p = P)$$

Donc $CO \vdash CC$, i.e. "Tout terme est une fonction".

Posons $E \equiv [x][y].xy$. Alors, $(C0) \vdash Exy = xy$. Or, $E \equiv [x].S(Kx)I$ donc $(C0) \vdash Ex = S(Kx)I$. Soit M un terme tel que $x \notin Var(M)$. Alors:

$$[x](Mx) = S([x]M)I = S(KM)I = EM$$

Soient les axiomes $(C1)$:

$$(C1) \quad [x][y]Kxy = K \quad [x][y][z]Sxyz = S$$

Alors, $(C0), (C1) \vdash Kx = [y]Kxy \quad Sxy = [z]Sxyz$
Donc: $(C0), (C1) \vdash Kx = E(Kx) \quad Sxy = E(Sxy) \quad (C2)$
On a :
$(C0) + (C2) \vdash [x].M = E([x].M) = [x].([x]M)x$
$(C0) + (C2) \vdash EE = E \quad E(Ex) = Ex$

Extensionnalité faible

Soit l'axiome $(EXTF)$ (ou ξ), dit *Schéma d'extensionnalité faible*

$$(EXTF) \quad \forall x.(M = N) \Longrightarrow [x]M = [x]N$$

(EXTF) peut être exprimé par un seul axiome (Extf), dans la théorie $(C0) + (C2)$:

$$(Extf) \quad \forall f, \forall g(\forall x.(fx = gx) \Longrightarrow (Ef = Eg)$$

La *Logique Combinatoire* est définie par les axiomes $(C0) + (C1) + (Extf)$ (ou $EXTF$).

Extensionnalité

L'axiome d'extensionnalité est le suivant:

$$(Ext) \quad \forall f, \forall g((\forall x, (fx = gx)) \Longrightarrow f = g$$

On montre alors que:

$$(C0), \vdash Ext \equiv ((Extf) + (E = I))$$

Donc,

$$(C0), \vdash Ext \equiv \forall f, \forall g(\forall x.(fx = gx) \Longrightarrow (f = g)$$

La *Logique Combinatoire avec extensionnalité* est définie par les axiomes $(C0) + (Ext)$.

2.2 Algorithme d'abstraction

Pour établir le fait que CL est bien une théorie de la fonctionnalité, on va définir un algorithme d'abstraction d'une variable x dans une terme M. Le résultat de cet algorithme est un terme de CL, noté $[x]M$.

Définition 2.4 *Soit $x \in V$. L'abstraction de x dans $M \in CL$ est définie par :*

1. $[x]x = I$
2. $[x]M = K\,M$ *si* $x \notin Var(M)$
3. $[x](M\,N) = S([x]M)([x]N)$

Théorème 2.5

$$([x]M)\,N \xrightarrow{\star} M[N/x]$$

Grâce aux propriétés de la réécriture du premier ordre, il suffit de montrer que :

$$([x]M)x \xrightarrow{\star} M$$

Il suffit de le montrer pour une étape de réduction, ce qui se fait par cas sur la construction de $[x]M$.

On en déduit la version suivante du théorème de complétude combinatoire:

Théorème 2.6 *Si* $\forall i, j,\ x_i \notin Var(N_j)$, *alors:*

$$([x_1, x_2, \ldots, x_n]M)\, N_1\, N_2\, \ldots\, N_n \xrightarrow{\star} M[N_1/x_1, \ldots, N_n/x_n]$$

Théorème 2.7 *Si* $x \notin Var(N)$,

$$([x]M)[N/y] =_{CL} [x](M[N/y])$$

La preuve se fait par cas, sur la construction de $[x]M$.

2.3 intertraduction entre $\Lambda_{\mathbf{V}}$ et CL

Définition 2.8 *On définit une traduction* $(\_)_{CL}$ *de* $\mathbf{\Lambda_V}$ *vers* CL *par:*

1. $(x)_{CL} = x$
2. $(MN)_{CL} = (M)_{CL}(N)_{CL}$
3. $(\lambda x.M)_{CL} = [x](M)_{CL}$

et une traduction $(\_)_\lambda$ *de* CL *vers* $\mathbf{\Lambda_V}$ *par:*

1. $K_\lambda = \lambda xy.x$
2. $S_\lambda = \lambda xyz.(x\, z)(y\, z)$
3. $(M\, N)_\lambda = M_\lambda\, N_\lambda$

Le but est de montrer que ces deux traductions définissent un isomorphisme entre les théories $\mathbf{\Lambda}$ et CL. Le résultat sera vrai à condition d'ajouter un certain nombre d'axiomes. En fait, aucun algorithme d'abstraction ne permet de démontrer le résultat précédent, seulement sous la théorie $(C0)$. Prouvé par Barendregt et Koymans [11] (par une méthode comparant les modèles des deux théories) puis par Chauvin et Mezguiche, avec deux méthodes syntaxiques différentes.

Théorème 2.9 *On peut étendre la théorie* CL *par une famille d'axiomes, les axiomes de Curry (donnés plus loin), tels que :*

1. $M =_\beta N \ \Rightarrow\ (M)_{CL} =_{CL} (N)_{CL}$
2. $A =_{CL} B \ \Rightarrow\ A_\lambda =_\beta B_\lambda$
3. $(A)_{\lambda,CL} =_{CL} A$
4. $(M)_{CL,\lambda} =_\beta M$

Ces résultats s'étendent à l'égalité $=_{\beta\eta}$ *en ajoutant un axiome d'extensionalité.*

Le point 1. s'obtient facilement si $M = (\lambda x.P)Q$ et $N = P\{Q/x\}$. Pour obtenir le résultat pour tout M, il suffit de montrer que cette propriété reste vraie en passant au contexte. Cela nécessite de disposer de l'étape de déduction suivante:

$$A = B \Rightarrow [x]A =_{CL} [x]B \quad (\xi)$$

Cette étape ne peut pas être une conséquence des axiomes de CL déjà introduits. En effet:

$$[x]S\,x\,y\,z = S([x]S\,x\,y)([x]z) = S(S([x]S\,x)([x]y))([x]z) = S(S(S(K\,S)I)(K\,y))(K\,z)$$

terme qui est en forme normale, à comparer avec $[x](x\,z)(y\,z)$.

Supposons savoir obtenir cette étape ξ. Pour le point 3., il suffit de montrer le point 3. pour S et pour K. Or,

$$(K)_{\lambda,CL} = (\lambda xy.x)_{CL} = [x](\lambda y.x)_{CL} = [x][y](x)_{CL}$$

Or, $K\,x\,y =_{CL} x$, si ξ est obtenue, on en déduit que

$$[xy].x =_{CL} [xy].K\,x\,y$$

De la même manière, on montre que:

$$(S)_{\lambda,CL} = S\,x\,y\,z$$

Posons $1_n = [ux_1 \dots x_n].u\,x_1 \dots x_n$ et introduisons les axiomes :

$$1_2K = K \quad (K\lambda); \;\; 1_3S = S \quad (S\lambda)$$

Comme

$$1_2K = ([u]([x\,y].u\,x\,y))\,K = [x\,y]\,K\,x\,y$$

On en déduit que $(K)_{\lambda,CL} = K$. De même, $(S)_{\lambda,CL} = S$. De plus,

$$K\,M = ([u\,x\,y].u\,x\,y)K\,M = [y]K\,M\,y = ([t\,y]ty)(K\,M)$$

$$S\,M\,N = ([u\,x\,y\,z].u\,x\,y\,z)S\,M\,N = [z]S\,M\,N\,z = ([t\,z]tz)(S\,M\,N)$$

Or, tout terme obtenu par l'algorithme d'abstraction est de la forme $K\,M$ ou $S\,M\,N$ (car $I = S\,K\,K$). On en déduit donc que :

$$1_1([x]A) = [x]A$$

Théorème 2.10 *Si on ajoute à CL les axiomes $K\lambda$, $S\lambda$ et l'axiome*

$$M\,x = N\,x \Rightarrow 1_1\,M = 1_1\,N$$

alors, la théorie obtenue permet d'obtenir le théorême 2.9 pour la théorie β.

Cette réponse n'est pas entièrement satisfaisante car la théorie ci-dessus n'est pas une théorie équationnelle. Pour obtenir la règle ξ comme conséquence équationnelle, il faut introduire trois axiomes supplémentaires :

$$[x\,y]K\,(x\,y) = [x\,y]S\,(K\,x)(K\,y) \quad (abs)$$

$$[x\,y]S(S(K\,K)x)y = [x\,y]x \quad (Kx)$$

$$[x\,y\,z]S(S(S(K\,S)x)y)z = [x\,y\,z]S(S\,x\,z)(S\,y\,z) \quad (Sx)$$

Pour obtenir le théorème 2.9 pour la théorie $\beta\eta$, il faut encore ajouter un autre axiome:

$$[x]S\,(K\,x)I = [x].x$$

Nous n'avons pas expliqué la genèse de ces axiomes dont la formulation est assez compliquée. Nous n'avons pas non plus fait la preuve du théorème 2.9. Ces notes ne donnent donc qu'un très petit aperçu de la Logique Combinatoire. La conclusion de ce survol peut être la suivante : on peut arriver à coder le λ-calcul sans utiliser de noms de variable avec les combinateurs S et K, mais, pour retrouver la puissance de calcul de $\mathbf{\Lambda}$, il faut ajouter des axiomes dont la formulation n'est pas simple. En fait, la théorie CL avec ses axiomes S et K correspond bien à la β-réduction faible, c'est-à-dire sans la règle ξ.

La lecture des ouvrages cités en référence est vivement conseillée.

3 Les λ-calculs avec couples

On a montré comment coder les notions de couple et de projection dans le λ-calcul.

Soient M et $N \in \mathbf{\Lambda_V}$. Soient les termes :

$$C \equiv \lambda f\, s\, d\, .\, d\, f\, s \qquad C_1 \equiv \lambda\, c.c\, (\lambda\, x\, y.x) \qquad C_2 \equiv \lambda\, c.c\, (\lambda\, x\, y.y)$$

On a alors :

$$C_1(C\, M\, N) \xrightarrow{\beta^\star} M \qquad C_2(C\, M\, N) \xrightarrow{\beta^\star} N$$

projection.

Ayant choisi ces termes C, *Fst* et *Snd*, a-t-on réellement ajouté un *opérateur* de construction de couples au λ-calcul? Autrement dit, cette construction est-elle canonique? Est-il possible de prouver l'unicité, modulo β-réduction, des couples bâtis à l'aide d'un terme C fixé?

La réponse est négative : quels que soient les termes utilisés pour représenter le couple et les projections, il existe des termes M tels que :

$$C(Fst\ M)\ (Snd\ M) \not\xrightarrow{\beta^\star} M$$

LA β-réduction ne permet donc pas d'affirmer l'unicité de la définition d'un couple. De plus, Barendregt [2] a montré que s'il existait un triplet de termes du λ-calcul (C, C_1, C_2) tel que l'égalité (SP) :

$$C(C_1M)\ (C_2M) =_{\beta\eta} M$$

soit vraie pour tout terme M, alors le λ-calcul serait une théorie incohérente. On obtient le même résultat pour la Logique Combinatoire. La démonstration repose sur le fait suivant :
Si on a l'égalité (SP) , alors :

$$C(C_1x)\ (C_2x) =_{\beta\eta} x$$

donc :

$$C(C_1x)\ (C_2x) \longrightarrow^{\beta\eta^\star} x$$

Soit $\Omega \equiv (\lambda x.x\, x)(\lambda x.x\, x)$. Alors,

$$C(C_1\Omega)\ (C_2\Omega) \longrightarrow^{\beta\eta^\star} \Omega$$

On montre alors, en les marquant, que nécessairement l'une des occurrences de Ω dans le membre gauche disparaît dans cette dérivation. On remplace ensuite cette occurrence par une nouvelle variable z obtenant ainsi :

$$C(C_1z)\ (C_2\Omega) \longrightarrow^{\beta\eta^\star} \Omega$$

D'où, en composant avec C_1 à gauche puis en réduisant :

$$C_1z =_{\beta\eta} C_1\Omega$$

Remplaçons alors z par $C\, t\, s$ puis par $C\, s\, t$. On obtient :

$$\forall s, t, \quad s =_{\beta\eta} C_1\Omega =_{\beta\eta} t$$

Par contre, la théorie obtenue en ajoutant explicitement trois constantes et les règles de projection ainsi que la règle (SP) reste cohérente. Nous allons donc l'étudier.

3.0.1 le λc-calcul applicatif

Définition 3.1 Le λc- calcul applicatif, noté $\mathbf{\Lambda_{c,a}}$ est obtenu en ajoutant au λ-calcul pur les constantes D, F, S et les règles :

$$\begin{array}{lrcl} \text{(Fst)} & F\,x\,y & \longrightarrow & x \\ \text{(Snd)} & S\,x\,y & \longrightarrow & y \\ \text{(SP)} & D\,(\,Fx)(\,Sx) & \longrightarrow & x \end{array}$$

définissant ainsi, avec la règle β (respectivement $\beta\eta$), la théorie (βSP) (resp. ($\beta\eta SP$)). Si on enlève la règle (SP), la théorie obtenue s'appelle βP.

Théorème 3.1 $\mathbf{\Lambda_{c,a}}$ *, muni de la théorie βP, vérifie la propriété de Church-Rosser.*
$\mathbf{\Lambda_{c,a}}$ *muni de la théorie βSP, est localement confluent mais ne vérifie pas la propriété de Church-Rosser.*

Démonstration
La démonstration du premier résultat ne présente pas de difficultés. Le second résultat a été obtenu par J. W. Klop [12] en 1980 . Mann avait posé le problème en 1972. R. Hindley , J. Staples avaient fourni des versions simplifiées de cette conjecture en 1974-1975 . Elle fut également soulevée par G. Huet et J.J. Levy à propos de l'opération de branchement dans les Schémas de Programmes Récursifs. J.W. Klop a montré qu'aucun de ces systèmes n'est confluent. ■

Donnons d'abord la construction du contre-exemple de J. W. Klop. Celui-ci montre qu'il existe deux dérivés d'un même terme qui ne peuvent être égaux.

Notations
Soit $P = \lambda\, x \lambda\, y.y\,(\,(xx)\,y)$. Soit $Y_T = P\,P$ le point fixe de Turing.

Lemme 3.2 *Soit $M \in \mathbf{\Lambda_{c,a}}$. Alors :*

$$Y_T\,M \xrightarrow{\beta^\star} M\,(Y_T\,M)$$

Proposition 3.3 Le contre-exemple de Klop
Soit E une variable libre. Posons :

$$V \equiv \lambda\, x \lambda\, y.E(\,D(F\,y)(S\,xy)) \qquad C = Y_T\,V \qquad B = Y_T\,C$$

Alors, le terme B admet deux dérivés $E(C\,B)$ et $C(E(C\,B))$ qui n'ont aucun dérivé commun.

Démonstration
D'après le lemme 3.2, on a :

$$C \xrightarrow{\beta^\star} V\,C \xrightarrow{\beta^\star} \lambda y.E(\,D(F\,y)(S\,C\,y))$$

$$B \xrightarrow{\beta^\star} C\,B \xrightarrow{\beta^\star} E(\,D(F\,B)(S\,CB))$$

Donc :

$$B \xrightarrow{\beta^\star} E(\,D(F\,(C\,B))(S\,(C\,B)))$$

En utilisant la règle (SP), on obtient :

$$B \xrightarrow{\beta^\star} C\,B \xrightarrow{\beta^\star} E(C\,B)$$

On peut aussi construire la dérivation suivante :

$$B \xrightarrow{\beta^\star} C\,B \xrightarrow{\beta^\star} C(\,E(\,C\,B))$$

Notons (SPE) une (SP)-réduction ainsi instanciée :

$$E\,(\,D\,(F\,x)(S\,x)) \longrightarrow E\,x$$

J. W. Klop montre alors que, dans toute dérivation de CB, on peut faire commuter les étapes de β-réduction et les étapes (SPE). Toute conversion entre $E\,(C\,B)$ et $C(\,E(CB))$ peut donc être factorisée ainsi :

$$E\,(C\,B) \xrightarrow{\beta^\star} \circ\,(SPE)^\star \circ (\,(SPE)^\star)^{-1} \circ \xleftarrow{\beta^\star} C(\,E(CB))$$

Les deux β-dérivations exhibées ci-dessus peuvent être remplacées par des dérivations standard. Or, on montre que toute dérivation standard de $C(\,E(CB))$ conduit au terme $D\,(E\,(C\,B))\,(C(\,E(CB))$. Donc, toute conversion n'utilisant que des étapes standard contient une conversion utilisant moins de symboles et pouvant aussi être standardisée. D'où le résultat. ■

Un autre Contre-Exemple

Le contre-exemple que nous proposons est bâti sur la constatation suivante : la réduction d'un (SP)-radical peut créer un β-radical, qui, une fois réduit, peut faire disparaître toute l'information contenue dans le (SP)-radical. Il ne nécessite pas le théorème de Standardisation. De plus, sa démonstration consiste à montrer qu'un terme $C\,I$ ne peut avoir pour forme normale le terme $\lambda x.x$, en explorant toutes les dérivations possibles de $C\,I$.

Notations
Soit $U = \lambda x\lambda y.D(F(\lambda z.z(xy))(S(\lambda z.zy))(\lambda z.I)$ où $I = \lambda x.x$.
Soit $C = Y_T\,U$ et $B = Y_T\,C$

Lemme 3.4 *Le terme B admet les termes I et $C\,I$ pour dérivés.*

Démonstration
D'après le lemme 3.2, on sait que :

$$C \xrightarrow{\beta^\star} UC \qquad B \xrightarrow{\beta^\star} C\,B$$

Donc, quel que soit le terme M, on a :

$$CM \xrightarrow{\beta^\star} D(F(\lambda z.z\,(C\,M)))(S(\lambda z.z\,M))(\lambda z.I)$$

On en déduit la dérivation D_1 de B :

B
$\xrightarrow{\beta^\star}$
$C\,B$
$\xrightarrow{\beta^\star}$
$D(F(\lambda z.z\,(C\,B)))(S(\lambda z.z\,B))(\lambda z.I)$
$\xrightarrow{\beta^\star}$
$D(F(\lambda z.z\,(C\,B)))(S(\lambda z.z\,(C\,B)))(\lambda z.I)$

$\xrightarrow{(SP)}$
$(\lambda z.z\ (C\ B))(\lambda z.I)$
$\xrightarrow{\beta}$
$(\lambda z.I)(C\ B)$
$\xrightarrow{\beta}$
I
Construisons alors la dérivation D_2 de B :

B
$\xrightarrow{\beta^\star}$
$C\ (C\ B)$
$\xrightarrow{D_1}$
$C\ I$ ■

Lemme 3.5 *Soit $M \in \mathbf{\Lambda_{c,a}}$, admettant une forme normale N distincte de I.*
Si βSP (resp. $\beta\eta SP$) vérifie la propriété d'unicité des formes normales, alors $C\ M$ et M ne peuvent avoir de dérivé commun.

Démonstration
Le terme

$$X \equiv D\ (F\ (\lambda z.z(C\ M))\ )\ (S\ (\lambda z.zM)\ )\ (\lambda z.I)$$

est un dérivé de $C\ M$. Soit A un dérivé commun à M et $C\ M$. X peut être réécrit sur :

$$D(F\ (\lambda z.zA\ )\ )(S\ (\lambda z.zA\ )\ )\ (\lambda z.I)$$

puis sur I. D'où le résultat. ■

Lemme 3.6 Si $M \xrightarrow{(\beta SP)^\star} N$, alors :

$$M\ [y \longleftarrow U\ ] \xrightarrow{(\beta SP)^\star} N\ [y \longleftarrow U\ ]$$

Démonstration
Elle est évidente.
On a le même résultat pour $\beta\eta SP$.

Proposition 3.7 *I ne peut pas être un β-dérivé (resp. un $\beta\eta$-dérivé) de $C\ I$.*

Démonstration
Faisons la pour les β-dérivations. Elle est identique pour les $\beta\eta$-dérivations.
Nous allons examiner toutes les dérivations possibles de $C\ I$ vers son éventuelle forme normale et montrer qu'aucune ne peut aboutir à I. On appellera R une telle dérivation. $C\ I$ ne contient qu'un seul radical : celui de Y_T. Sa réduction conduit au terme A_1 :

$$A_1 = (\ (\ \lambda y.y\ (Y_T\ y)\ )\ U\ )\ I$$

R peut se prolonger en dérivant le sous-terme $Y_T\ y$. Soit $Red(Y_T\ y)$ le sous-terme ainsi obtenu. R doit nécessairement réduire le radical le plus externe. R contient donc un terme A_2 :

$$A_2 = U(Red(Y_T\ y)[y \longleftarrow U]\ )\ I = U(Red(Y_T U)\ )\ I = U(Red(C))\ I$$

R peut se prolonger en dérivant le sous-terme $Red(C)$. Elle doit nécessairement réduire le radical le plus externe. R contient donc un terme A_3 :

$$A_3 \equiv (\lambda y.(D\ (F\ (\lambda z.z(Red\,(C)y)))\ (S\ (\lambda z.zy))\ )\ (\lambda z.I)\ )\ I$$

A_3 contient un sous-terme $D(F..)(S..)$. Si (βSP) est confluente, alors elle vérifie la propriété d'unicité des formes normales et, par conséquent, d'après le lemme 3.5, Cy ne peut se dériver sur y. Donc, ce contexte ne peut pas disparaître avant la réduction du radical le plus externe. R contient donc un terme A_4 :

$$A_4 \equiv D(\ (F\ (\lambda z.zRed(CI))\ )\ (S\ (\lambda z.zI)\ ))\ (\lambda z.I)$$

Appelons longueur d'une dérivation le nombre de (SP)-étapes qu'elle contient. Soit $Rmin$ une dérivation de CI sur I de longueur minimale. $Rmin$ doit faire disparaître le contexte $DF(..)S(..)(\lambda z.I)$ Elle contient donc une dérivation de CI sur I et n'est donc pas de longueur minimale. Il n'existe donc pas de dérivation de CI sur I. D'où la contradiction. ■

3.0.2 Le λc-calcul fonctionnel

Définition 3.8 Le λc- calcul fonctionnel, noté $\mathbf{\Lambda_{c,f}}$ est obtenu en ajoutant au λ-calcul pur un opérateur binaire noté $<,>$, deux opérateurs unaires notés fst et snd ainsi que les règles :

$$\begin{array}{lrcl} \text{(Fst)} & fst\,(<\,x,\,y\,>) & \longrightarrow & x \\ \text{(Snd)} & snd\,(<\,x,\,y\,>) & \longrightarrow & y \\ \text{(SP)} & <\,fst\,(x)\,,\,snd\,(x)\,> & \longrightarrow & x \end{array}$$

définissant ainsi, avec la règle β, une théorie encore notée (βSP). Si on enlève la règle (SP), la théorie obtenue s'appelle encore βP.

Théorème 3.2 $\mathbf{\Lambda_{c,f}}$*, muni de la théorie* βP*, vérifie la propriété de Church-Rosser.*
$\mathbf{\Lambda_{c,f}}$*, muni de la théorie* (βSP)*, ne vérifie pas la propriété de Church-Rosser.*

Démonstration
Elle est obtenue à partir de la précédente en remplaçant le terme U du cas applicatif par le terme W :

$$W \equiv \lambda x\ \lambda y(<\ fst\,(\lambda z.\ z(xy)\ )\ ,\ snd\,(\ \lambda z.zy\ )\ >\ (\lambda z.\ I)\ )$$

■

Remarque 3.1 Pour le λ-calcul, quelle que soit la méthode d'extension choisie, la propriété de Church-Rosser n'est pas conservée. Il n'en est pas de même dans le cas de la Logique Combinatoire. Etendons la Logique Combinatoire, CL, avec trois constantes D, F, S et les règles (Fst), (Snd) et (SP). Soit CLa la théorie ainsi obtenue. J. W. Klop a montré que CLa ne vérifie pas la propriété de Church-Rosser.

Etendons alors CL en ajoutant les opérateurs $<,>$, $Fst\,(\,)$ et $Snd\,(\,)$. Cette fois, la théorie obtenue CLf vérifie la propriété de Church-Rosser. Ce résultat est une conséquence du théorème de Y. Toyama sur les sommes disjointes de systèmes confluents [20]. En fait, on peut considérer que $\mathbf{\Lambda_{c,a}}$ comme CLa contiennent la curryfication de l'opération de couple : Dx est en quelque sorte le curryfié de $<x,->$. CLa contient donc des termes fondamentalement différents de ceux de CLf. Ces différences peuvent donc expliquer les propriétés différentes de ces théories.

4 La notation de de Bruijn

Pour éviter le problème d'α-conversion tout en conservant la structure des termes du λ-calcul, le plus simple consiste à supprimer les noms des variables liées. Pour effectuer correctement la substitution, il suffit de connaitre les occurrences du terme liées par l'abstracteur disparaissant dans la β-réduction. Dans $\mathbf{\Lambda_V}$ ce lien est indiqué par l'identité nom de la variable à l'occurrence examinée, nom de la variable de l'abstracteur. Ce lien peut tout aussi bien être indiqué par la *hauteur de liaison* de l'occurrence : le nombre de λ à franchir pour remonter de l'occurrence examinée jusqu'à l'abstracteur qui la lie. Si on décide de supprimer également le nom des variables libres $\{x_0, \ldots, x_n\}$ d'un terme a, il suffit de considérer a comme sous-terme de $\lambda x_0 \ldots x_n.a$. Cette idée a été développée par N.G. de Bruijn dans le cadre du projet Automath à l'Université d'Eindohven. [3].

Définition 4.1 $\mathbf{\Lambda}$, l'ensemble des termes du λ-calcul, décrit avec la notation de de Bruijn, est le plus petit ensemble de termes vérifiant :

1. Si $n \geq 1$, alors $n \in \mathbf{\Lambda}$
2. Si a et $b \in \mathbf{\Lambda}$, alors $a\, b \in \mathbf{\Lambda}$
3. Si $a \in \mathbf{\Lambda}$, alors $\lambda(a) \in \mathbf{\Lambda}$

Définition 4.2 Soit $a \in \mathbf{\Lambda_V}$ tel que $FV(a) \subseteq \{x_1\, , \ldots x_n\}$. La traduction de a, notée $a_{DB_{\{x_1 \ldots x_n\}}}$, dans $\mathbf{\Lambda}$, est définie par :

1. $x_{DB_{\{x_1\, ,\ldots x_n\}}} = i$ si i est le plus petit indice tel que $x = x_i$
2. $(\lambda\, x.a)_{DB_{\{x_1\, ,\ldots x_n\}}} = \lambda\, .a_{DB_{\{x\}\cup\{x_1\, ,\ldots x_n\}}}$
3. $(a\, b)_{DB_{\{x_1\, ,\ldots x_n\}}} = a_{DB_{\{x_1\, ,\ldots x_n\}}}\, b_{DB_{\{x_1\, ,\ldots x_n\}}}$

L'ensemble des variables libres d'un terme est donc représenté par une liste, que nous appellerons *référentiel*. Pour obtenir la liste des variables libres d'un sous-terme "sous" un abstracteur, il suffit d'ajouter, en tête de la liste, la variable associée à l'abstracteur comme indiqué dans le point 2 de la définition précédente.

Exemple

$$((\lambda\, x.xy)y)_{DB_{\{y\}}} = (\lambda\, x.xy)_{DB_{\{y\}}}(y_{DB_{\{y\}}})$$

$$(\lambda\, x.xy)_{DB_{\{y\}}} = \lambda.(xy)_{DB_{\{x,y\}}} = \lambda\, .1\; 2 \qquad (y_{DB_{\{y\}}}) = 1$$

Donc :

$$((\lambda\, x.xy)\, y)_{DB_{\{y\}}} = (\lambda.1\; 2)\; 1$$

Il n'y a donc pas de correspondance directe entre numéros et noms de variables : y est représentée à la fois par 2 et 1.

Remplacer $(\lambda\, x.a)b$ par $a\{b/x\}$ nécessite d'identifier parmi les numéros ceux concernés par la β-réduction, autrement dit ceux qui sont égaux à leur hauteur de liaison. On peut calculer de façon récursive ces hauteurs de liaison à l'aide d'une famille de compteurs gérée ainsi : on démarre la

traversée du terme a avec un seul compteur initialisé à 1. Au passage d'un noeud application, on crée deux exemplaires du compteur de façon à en obtenir un pour chaque branche. Au passage d'un noeud abstraction, on incrémente le compteur. Lorsqu'on arrive à l'occurrence d'un numéro, s'il est supérieur à la valeur du compteur, on le diminue de 1 puisqu'il existe un λ de moins sur le chemin entre ce numéro et l'abstracteur qui le lie. Sinon, si ce numéro est égal à la valeur du compteur, il doit être remplacé par le terme b, qui va donc se trouver ainsi placé sous $(n-1)$ λ, si n est la valeur du compteur. Il faut alors ajuster les numéros de b de façon à ne pas modifier les liaisons dans b. Si le numéro est inférieur à la valeur du compteur, alors il est lié par un abstracteur interne à a et il n'a donc pas à être modifié.

La substitution, dans le formalisme de De Bruijn, est donc définie comme suit :

Définition 4.3 La substitution du terme b à la hauteur n dans le terme a, notée $\sigma_n(a,b)$ ainsi que l'incrémentation avec n à partir de i, notée $\tau_i^n(a)$, sont définies par récurrence ainsi :

$$\begin{array}{rcl}
\sigma_n(a\ c, b) & = & \sigma_n(a,b)\ \sigma_n(c,b) \\
\sigma_n(\lambda.a, b) & = & \lambda(\sigma_{n+1}(a,b)) \\
\sigma_n(m,b) & = & \left\{ \begin{array}{rcl} m-1 & \text{si} & m > n \\ \tau_0^n(b) & \text{si} & m = n \\ m & \text{si} & m < n \end{array} \right. \\
\\
\tau_i^n(m) & = & \left\{ \begin{array}{rcl} m+n-1 & \text{si} & m > i \\ m & \text{si} & m \leq i \end{array} \right. \\
\tau_i^n(a\ c) & = & \tau_i^n(a)\ \tau_i^n(c) \\
\tau_i^n(\lambda(a)) & = & \lambda(\tau_{i+1}^n(a))
\end{array}$$

Définition 4.4 Dans $\mathbf{\Lambda}$, la β-réduction est la relation de réécriture définie par la règle :

$$(\lambda\ .a)b \longrightarrow \sigma_1(a,b)$$

References

[1] Barendregt H., *The Lambda-Calculus.* Volume 103, Elsevier Science Publishing Company, Amsterdam, 1984. *Généralement considéré comme la Bible du λ-calcul*

[2] H. P. Barendregt. *Pairing without conventional restraints.* Zeitschr. J. Math And Logik und Grundlagen d. Math, 20:289–306, 1974.

[3] N. de Bruijn. *Lambda-Calculus notation with nameless dummies, a tool for automatic formula manipulation, with application to the Church-Rosser Theorem.* Indag. Math., 34(5):381–392, 1972.

[4] N. de Bruijn. *Lambda-Calculus notation with namefree formulas involing symbols that represent reference transforming mappings* Indag. Math.,40 :348–356, 1978.

[5] N. de Bruijn. *A namefree lambda calculus with facilities for internal definition of expressions and segments.* T.H.-Report 78-WSK-03, Technological University Eindhoven, Eindhoven - The Netherlands, August 1978.

[6] J. M. Carthy. *Recursive Functions of Symbolic Expressions and their Computation by by machine.* C.A.C.M, 3(4):184–195, 1960.

[7] A. Church. *The Calculi of Lambda-Conversion.* Ann. of Math. Studies, 6, 1941.

[8] Curien P.-L., *Categorical Combinators, Sequential algorithms and Functional Programming.* Second Edition, Birkäuser Boston, 1993.

[9] H. B. Curry. *Combinatory Logic.* Volume 1, North-Holland, 1958.

[10] Hindley R. J., Seldin J., *Introduction to Combinators and λ-calculus* London Mathematical Society Student Texts, Cambridge University Press 1986. *Contient une présentation du λ-calcul et de la logique combinatoire : syntaxe, modèles et typage*

[11] Hindley R. J., Seldin J., *To H. B. Curry : Essays on Combinatory Logic, Lambda-calculus and Formalism* Academic Press, 1980. *Un recueil d'articles devenus, pour la plupart, des classiques du domaine.*

[12] J. W. Klop. *Combinatory Reduction Systems.* Thèse, Mathematisch Centrum Amsterdam, 1982.

[13] Huet G., *Polycopiés de cours de Lambda-calcul*,

[14] Klop J.W. , *Combinatory Reduction Systems*, Math. Center Tracts 129, Amsterdam, 1980. *Thèse de J. W. Klop, assez difficile à trouver. Lecture recommandée comme complément au cours.*

[15] Krivine J.L.,

Lambda-Calcul, types et modèles Masson 1991 *Lecture recommandée comme complément au cours.*

[16] Lalement R. *Logique, réduction, résolution* Masson 1990. *Une présentation parfois succinte de toutes les notions abordées en tronc commun.*

[17] Lévy J.-J., *Réductions correctes et optimales dans le λ-calcul*, Thèse d'Etat, 1978, Univ. Paris 7. *Lecture recommandée comme complément au cours.*

[18] Krivine J.L., *Théorie Axiomatique des Ensembles* Collection SUP, Presses Universitaires de France

[19] Steelund S. *Combinators, λ-terms and proof theory* Reidel Publishing Compagny

[20] Y. Toyama. *On the Church-Rosser Property for the direct sum of Term Rewriting Systems.* 1985. J.A.C.M.