RAIRO. INFORMATIQUE THÉORIQUE ET APPLICATIONS

JEAN-LOUIS KRIVINE

Lambda-calcul, évaluation paresseuse et mise en mémoire

RAIRO. Informatique théorique et applications, tome 25, n° 1 (1991), p. 67-84

<http://www.numdam.org/item?id=ITA_1991__25_1_67_0>

Informatique théorique et Applications/Theoretical Informatics and Applications
(vol. 25, n° 1, 1991, p. 67 à 84)

LAMBDA-CALCUL, ÉVALUATION PARESSEUSE ET MISE EN MÉMOIRE (*)

par Jean-Louis KRIVINE (1)

Communiqué par J.-E. PIN

Résumé. – *En λ-calcul pur, l'évaluation paresseuse est identifiée avec la β-réduction à gauche. Cette stratégie de réduction a beaucoup d'avantages (en particulier, elle aboutit toujours à la forme normale lorsqu'il y en a une); elle a l'inconvénient qu'une fonction qui utilise plusieurs fois son argument doit le recalculer à chaque fois. Pour y remédier, on introduit, pour chaque type de données (booléens, entiers, listes...), des opérateurs dits de « mise en mémoire ». Par exemple, si* ν *est* β*-équivalent à un entier, et* φ *un terme quelconque, on remplace* φν *par* $T\nu\varphi$, *où* T *est un opérateur de mise en mémoire pour les entiers; au cours de l'évaluation paresseuse de* $T\nu\varphi$, ν *ne sera calculé qu'une seule fois.*

On donne la définition générale de la notion d'opérateur de mise en mémoire pour un ensemble de termes du λ-calcul. On montre comment on peut construire de tels opérateurs, au moyen d'un système de λ-calcul typé du second ordre. On utilise, pour cela, la $\neg\neg$*-traduction de Gödel, qui permet de passer d'une preuve en logique classique à une preuve intuitionniste.*

Abstract. – *In pure λ-calculus, we identify lazy evaluation with leftmost β-reduction. This strategy of reduction has many advantages (in particular, it always gives normal form when there exists one); but it has the following drawback: the argument of a function must be evaluated as many times as it is used. To avoid this defect, we introduce, for each data type (like booleans, integers, lists...), some terms called "storage operators". For example, if* ν *is* β*-equivalent to an integer, and* φ *is any term, we replace* φν *by* $T\nu\varphi$, *where* T *is a storage operator for integers; during lazy evaluation of* $T\nu\varphi$, ν *will be evaluated only once.*

We give a general definition for the notion of storage operator for a set of λ-calculus terms. We show how to construct such operators, using a system of typed λ-calculus of second order. To this end, we use Gödel's $\neg\neg$*-translation, which transforms a proof in classical logic into an intuitionistic proof.*

NOTATIONS : *Étant donné des termes* t, u, v,... *du λ-calcul pur, l'application de* t *à* u *sera notée* $(t)u$ *(ou* tu*);* $((t)u)v$ *sera noté, pour abréger* $(t)uv$ *ou* tuv*; etc.*

La notation $t \simeq u$ *signifie que* t *et* u *sont* β*-équivalents.*

(*) Reçu janvier 1989, version finale en juillet 1989.

(1) Équipe de Logique Mathématique, Université Paris-VII; 2, place Jussieu, 75251 Paris Cedex 05, France.

$\lambda x_1 \ldots \lambda x_k$ pourra être abrégé en $\lambda \mathbf{x}$ avec $\mathbf{x}=(x_1, \ldots, x_k)$; de même, si t, $u_1, \ldots, u_k$ sont des termes, le terme $(t)u_1 \ldots u_k$ pourra être noté $(t)\mathbf{u}$ avec $\mathbf{u}=(u_1, \ldots, u_k)$.

La notation $t[u_1/x_1, \ldots, u_k/x_k]$ représente le résultat de la substitution simultanée de u_1 à $x_1, \ldots, u_k$ à x_k dans le terme t.

Si n est un entier $\geqq 0$, $(t)^n u$ désigne le terme $(t) \ldots (t)u$ (t répété n fois).

On désigne par $\mathbf{0}$ et $\mathbf{1}$ les deux booléens réduits $\lambda x \lambda y\, y$ et $\lambda x \lambda y\, x$.

En λ-calcul pur, nous considérons que l'évaluation paresseuse (ou « appel par nécessité ») est la stratégie de réduction par la gauche : étant donné un terme, son évaluation paresseuse consistera donc en la suite de β-réductions opérant à chaque instant sur le redex le plus à gauche du terme obtenu.

Dans cet article, on adopte la stratégie de réduction par la gauche et on va tenter de remédier à ses inconvénients. Remarquons tout de suite qu'elle a beaucoup d'avantages : le plus convaincant est que c'est la stratégie de réduction qui est suggérée par la preuve de normalisation des λ-calcul typés suivant la méthode de réductibilité; également, on sait (théorème de standardisation) qu'elle aboutit toujours si le terme auquel on l'applique est normalisable. De plus, elle semble être plus économique, puisqu'on n'évalue un terme que s'il faut le faire pour la suite du calcul : par exemple, si b est un booléen et t, u des termes quelconques, l'évaluation paresseuse de $(b)tu$ (qui signifie « si b alors t sinon u ») ne calculera que le terme t ou u qui est utile (c'est évident si b est réduit; pour le cas général, *cf.* théorème 1). Dans une telle situation, l'évaluation paresseuse est indispensable.

Il semble y avoir une objection majeure, qui fait que l'évaluation paresseuse n'est pas vraiment utilisée comme principe de fonctionnement général. On peut exprimer, en gros, cette objection ainsi : une fonction qui utilise plusieurs fois son argument doit le recalculer à chaque fois.

Exemple : soit b un booléen (non réduit), et t, u, v des termes. Le therme $(\lambda x((x)t)(x)uv)b$ (qui signifie « si b alors t sinon si b alors u sinon v ») donnera par évaluation paresseuse $((b)t)(b)uv$. Supposons que $b \simeq \mathbf{0}$. On obtiendra alors, en réduisant b, $((\mathbf{0})t)(b)uv$ puis $(b)uv$; on doit alors recommencer la réduction de b. Il eût évidemment mieux valu réduire tout de suite b en $\mathbf{0}$ dans le terme initial, puis évaluer $(\lambda x((x)t)(x)uv)\mathbf{0}$.

Notons cependant que si l'évaluation de b produit un « effet de bord » (entrée-sortie, communication avec un autre processus, affectation...), les deux façons d'évaluer ne sont pas équivalentes; selon l'effet recherché, il sera peut-être nécessaire de répéter l'évaluation de b.

ÉVALUATION PARESSEUSE DE (t) u

Rappelons qu'un terme t du λ-calcul est nécessairement de la forme $\lambda \vec{\mathbf{z}}(a)\,\mathbf{b}$, où a est soit une variable, soit un redex $(a=(\lambda xu)\,v)$. Dans le premier cas, on dit que t est une *forme normale de tête* (f. n. t. en abrégé). Dans le second cas, le redex a est appelé le *redex de tête* de t.

Un terme t est dit *résoluble* s'il est β-équivalent à une f. n. t.

Lorsqu'un terme t n'est pas une f. n. t., la *réduction de tête* de t consiste à réduire le redex de tête d'abord dans t, puis dans le terme obtenu, et ainsi de suite. On sait (*cf.* [1]) que, si t est résoluble, la réduction de tête de t aboutit à un terme sous forme normale de tête, appelé *forme normale de tête principale* de t.

Soient $\mathbf{z}=(z_1, \ldots, z_k)$ une suite de variables et $\mathbf{u}=(u_1, \ldots, u_n)$ une suite de termes. Alors $(\lambda \mathbf{z}\, t)\,\mathbf{u}$ sera appelé un redex multiple [de multiplicité $m=\inf(k, n)$]. Sa réduction consiste à réduire les m redex qui le constituent.

Ce redex multiple sera désigné en abrégé par $\lambda \mathbf{z}$ s'il n'y a pas d'ambiguïté.

LEMME : *A partir du terme* $\tau=(\lambda \mathbf{x}\,(\lambda y\, v)\,\mathbf{w})\,\mathbf{u}$, *on obtient le même résultat en réduisant d'abord le redex* $\lambda \mathbf{x}$ *puis* λy, *ou bien d'abord le redex* λy *puis* $\lambda \mathbf{x}$.

On écrit $\tau=(\lambda x_1 \ldots \lambda x_k\,(\lambda yv)\,w_0 \ldots w_m)\,u_1 \ldots u_n$; on peut supposer $n \leqq k$ (si $n>k$, il suffit de supprimer $u_{k+1}, \ldots, u_n$; en effet ces termes ne sont pas modifiés par les réductions considérées).

Posons $w_i'=w_i[u_1/x_1, \ldots, u_n/x_n]\,(0 \leqq i \leqq m)$. Si on réduit successivement les redex $\lambda \mathbf{x}$ puis λy, on obtient :

$$\lambda x_{n+1} \ldots \lambda x_k\,(\lambda yv[u_1/x_1, \ldots, u_n/x_n])\,w_0' \ldots w_m',$$

puis

(i) $\lambda x_{n+1} \ldots \lambda x_k\,(v[u_1/x_1, \ldots, u_n/x_n][w_0'/y])\,w_1' \ldots w_m'$.

Si on réduit successivement les redex λy puis $\lambda \mathbf{x}$, on obtient :

$$(\lambda x_1 \ldots \lambda x_k\,(v[w_0/y])\,w_1 \ldots w_m)\,u_1 \ldots u_n,$$

puis

(ii) $\lambda x_{n+1} \ldots \lambda x_k\,(v[w_0/y][u_1/x_1, \ldots, u_n/x_n])\,w_1' \ldots w_m'$.

Or on a $v[w_0/y][u_1/x_1, \ldots, u_n/x_n] \equiv v[u_1/x_1, \ldots, u_n/x_n][w_0'/y]$, puisque la variable y n'est pas libre dans $u_1, \ldots, u_n$. Il en résulte que les termes (i) et (ii) sont identiques.

C.Q.F.D.

THÉORÈME 1 : *Soit t un terme résoluble, dont la forme normale de tête*

principale est $\lambda \mathbf{z}(a)\mathbf{b}$. *Alors la réduction de tête de* $(t)\mathbf{u}$ *aboutit au terme obtenu en effectuant d'abord la réduction de tête de t puis en réduisant le redex multiple* $\lambda \mathbf{z}$ *dans* $(\lambda \mathbf{z}(a)\mathbf{b})\mathbf{u}$; *et ce en le même nombre de pas.*

On montre le résultat par induction sur la longueur p de la réduction de tête de t. C'est évident si t est une f.n.t. Si t ne commence pas par λ, le premier pas de la réduction de tête de $(t)\mathbf{u}$ se fait dans t, d'où le résultat par hypothèse de récurrence. Sinon, on a $t = \lambda \mathbf{x}(\lambda y v)\mathbf{w}$ (puisque t commence par λ et n'est pas une f.n.t.). La réduction de tête de $(t)\mathbf{u} = (\lambda \mathbf{x}(\lambda y v)\mathbf{w})\mathbf{u}$ commence par la réduction successive des redex $\lambda \mathbf{x}$ puis λy. Mais, d'après le lemme, le résultat (et le nombre de β-réductions) est le même si on réduit d'abord le redex λy puis $\lambda \mathbf{x}$. Or la réduction de λy donne $(t_1)\mathbf{u}$, $t_1 = \lambda \mathbf{x} v'$ étant obtenu à partir de t par une β-réduction de tête. On a donc à considérer la réduction de tête de $(t_1)\mathbf{u}$, et la longueur de la réduction de tête de t_1 est $p-1$. D'où le résultat par hypothèse de récurrence.

C.Q.F.D.

MISE EN MÉMOIRE D'UN BOOLÉEN

Dans tout ce qui suit, on utilise systématiquement la stratégie de réduction par la gauche.

PROPOSITION 1 : *Soit* ψ *un terme de la forme* $\lambda x \lambda \mathbf{y}(x) u_1 \ldots u_k$, *la variable* x *n'étant pas libre dans* $u_1 \ldots u_k$; *si* b *est un booléen* ($b \simeq 0$ *ou* 1), *alors* b *est calculé une seule fois lors de l'évaluation paresseuse de* $(\psi)b$.

Autrement dit, le terme ψ, considéré comme une fonction sur les booléens, n'évalue son argument qu'une fois.

En effet, $(\psi)b$ se réduit par la gauche en $\lambda \mathbf{y}(b) u_1 \ldots u_k$. D'après le théorème 1, on peut considérer qu'on effectue ensuite la réduction de tête de b; or, cela revient à normaliser b : en effet, si par exemple $b \simeq \mathbf{1} = \lambda x \lambda z x$, toute forme normale de tête de b s'écrit $\lambda x \lambda z t$, avec $t \simeq x$; mais t est sous f.n.t., donc $t = x$. On trouve donc $\lambda \mathbf{y}(\lambda x \lambda z x) u_1 \ldots u_k$, puis $\lambda \mathbf{y}(u_1) u_3 \ldots u_k$, et b n'est évalué qu'une fois.

C.Q.F.D.

Soit alors φ un terme considéré comme fonction sur les booléens. On cherche un terme ψ définissant la même fonction sur les booléens, et qui n'évalue son argument qu'une seule fois. Il suffit pour cela que ψ soit de la forme indiquée dans la proposition 1. Une solution évidente est $\psi = \lambda x((x)(\varphi)\mathbf{1})(\varphi)\mathbf{0}$.

Désignons alors par $!B$ le terme $\lambda x \lambda f ((x)(f)\mathbf{1})(f)\mathbf{0}$. Pour tout booléen b ($b \simeq \mathbf{0}$ ou $\mathbf{1}$) et tout terme φ, on a immédiatement $(!B)\, b\, \varphi \simeq \varphi\, b$.

Considérons la réduction par la gauche de $(!B)\, b\, \varphi$. On obtient d'abord $((b)(\varphi)\mathbf{1})(\varphi)\mathbf{0}$. Ensuite, d'après le théorème 1, on peut continuer en faisant la réduction de tête de b, ce qui donne un booléen réduit ε ($\varepsilon \equiv \mathbf{0}$ ou $\mathbf{1}$). On réduit ensuite $((\varepsilon)(\varphi)\mathbf{1})(\varphi)\mathbf{0}$, et on aboutit à $\varphi\varepsilon$. On est alors ramené à évaluer $\varphi\varepsilon$, ε étant un booléen réduit.

On peut donc naturellement considérer que $(!B)\, b$ est « l'opération de mise en mémoire du booléen b ». $!B$ (et tout autre terme ayant les mêmes propriétés) sera appelé *opérateur de mise en mémoire* d'un booléen.

Par exemple, le terme $!B_1 = \lambda x ((x) \lambda f f \mathbf{1}) \lambda f f \mathbf{0}$ est également un opérateur de mise en mémoire d'un booléen (même démonstration).

DÉFINITION : Un terme T du λ-calcul sera appelé *opérateur de mise en mémoire d'un booléen* si, pour tout booléen, b ($b \simeq \varepsilon$, $\varepsilon \equiv \mathbf{0}$ ou $\mathbf{1}$) le terme Tb se réduit à $\lambda f (f)\varepsilon$ *par réduction de tête.*

D'après le théorème 1, on voit que, pour tout terme φ, l'évaluation paresseuse de $(Tb)\varphi$ revient à faire d'abord la réduction de tête de Tb, ce qui donne $\lambda f (f)\varepsilon$, puis à évaluer $(\lambda f (f)\varepsilon)\varphi$, c'est-à-dire $(\varphi)\varepsilon$. Le booléen b a donc été calculé d'abord.

MISE EN MÉMOIRE D'UN ENTIER DE CHURCH

Soit ψ un terme λ-calcul, considéré comme une fonction sur les entiers de Church.

Maintenant, il ne suffit plus que ψ ait la forme $\lambda x (x)\mathbf{u}$, avec x non dans $\mathbf{u}$, pour que ψ n'évalue qu'une fois son argument.

Par exemple, soient $\psi = \lambda x (x) \lambda y (y) y$, et ν un entier de Church (non réduit), $\nu \simeq \lambda f \lambda x (f)^n x$. Alors $(\psi)\nu$ s'évalue d'abord en $(\nu) \lambda y (y) y$. D'après le théorème 1, on peut considérer que la suite de l'évaluation consiste à faire la réduction de tête de ν, ce qui donne $\lambda f \lambda x (f)\mu$, avec $\mu \simeq (f)^{n-1} x$. On obtient ensuite $\lambda x (\lambda y (y) y)\mu'$, avec $\mu' = \mu[\lambda y (y) y/f]$, puis $\lambda x (\mu')\mu'$. Il est alors clair que ψ va évaluer plusieurs fois son argument.

La proposition suivante donne une condition suffisante pour que ψ n'évalue son argument entier qu'une seule fois :

PROPOSITION 2 : *Soit ψ un terme de la forme $\lambda x \lambda \vec{\mathbf{y}} (x) u u_0 \ldots u_k$, la variable x n'étant pas libre dans u, $u_0, \ldots, u_k$; u est supposé de la forme $\lambda g \lambda \mathbf{z} (g) v_1 \ldots v_m$, g n'étant pas libre dans $v_1, \ldots, v_m$. Si ν est un entier de*

Church [$\nu \simeq \lambda f\, \lambda x\, (f)^n x$ *pour un* $n \in \mathbb{N}$], *alors la f.n.t. principale de* $(\psi)\nu$ *ne dépend que de n, et* ν *n'est calculé qu'une fois lors de l'évaluation paresseuse de* $(\psi)\nu$.

On raisonne par récurrence sur n (l'entier de Church normal équivalent à ν). La réduction de $(\psi)\nu$ donne d'abord le terme $\tau = \lambda\, \mathbf{y}\, (\nu)\, uu_0 \ldots u_k$. D'après le théorème 1, tout se passe ensuite comme si on faisait la réduction de tête de ν. Si $n=0$, on aboutit alors à $\lambda\, \mathbf{y}\, (\lambda f\, \lambda x x)\, uu_0 \ldots u_k$, puis à $\lambda\, \mathbf{y}\, (u_0)\, u_1 \ldots u_k$, et ν n'a été évalué qu'une seule fois.

Si $n>0$, on aboutit à $\lambda\, \mathbf{y}\, (\lambda f\, \lambda x\, (f)\, \mu)\, uu_0 \ldots u_k$. On pose $\nu' = \lambda f\, \lambda x\, \mu$; ν' est alors un entier de Church, β-équivalent à $n-1$. Le terme τ s'est donc réduit en $\lambda\, \mathbf{y}\, (u)\, \mu^* u_1 \ldots u_k$, avec $\mu^* = \mu\, [u/f,\, u_0/x]$. Or on a $u = \lambda g\, \lambda z_1 \ldots \lambda z_p\, (g)\, v_1 \ldots v_m$, g n'étant pas libre dans $v_1, \ldots, v_m$.

La réduction de $\lambda\, \mathbf{y}\, (u)\, \mu^* u_1 \ldots u_k$ donne alors le terme

$$\tau^* = \lambda\, \mathbf{y}\, \lambda\, \mathbf{z}'\, (\mu^*)\, v'_1 \ldots v'_m\, \mathbf{w};$$

la suite $\mathbf{z}'$ est vide si $p \leqq k$, la suite $\mathbf{w}$ est vide si $p \geqq k$; $v'_1, \ldots, v'_m$ et $\mathbf{w}$ ne dépendent pas de ν.

Posons alors $\tau' = \lambda\, \mathbf{y}\, z'\, (\nu')\, uu_0 v'_1 \ldots v'_m\, \mathbf{w}$. Comme $\nu' = \lambda f\, \lambda x\, \mu$, τ' se réduit en τ^* par deux pas de β-réduction de tête. Tout se passe donc comme si on effectuait maintenant la réduction de tête de τ'. Or τ' a la même forme que τ, mais l'entier de Church ν a été remplacé par $\nu' \simeq n-1$. D'où le résultat, par hypothèse de récurrence.

C.Q.F.D.

Donnons-nous un terme φ, considéré comme fonction sur $\mathbb{N}$ (ensemble des entiers de Church). On cherche un terme ψ de la forme indiquée dans la proposition 2, tel que $(\psi)\, n \simeq (\varphi)\, n$, pour tout $n \in \mathbb{N}$. Soit s un terme pour le successeur (par exemple, $s = \lambda k\, \lambda f\, \lambda x\, (f)\, (k) f x$).

1^{er} *exemple :* On cherche Ψ telle que $(\Psi)\, n \simeq \varphi \circ s^n$ c'est-à-dire $\lambda\, x\, (\varphi)\, (s)^n\, x)$. On définit Ψ par récurrence :

$$(\Psi)\, 0 = \varphi; \qquad (\Psi)\, (s)\, n = (F)\, (\Psi)\, n \qquad \text{avec} \quad F = \lambda g\; g \circ s = \lambda g\, \lambda x\, (g)\, (s)\, x.$$

On trouve donc $(\Psi)\, n = (n)\, F \varphi$. On peut alors définir le terme ψ cherché en posant $(\Psi)\, x = (\Psi)\, x\, \mathbf{0} = (x)\, F \varphi\, \mathbf{0}$.

Un candidat pour un opérateur $!\mathbb{N}$ de mise en mémoire d'un entier de Church est donc donné par : $!\mathbb{N} = \lambda x \lambda f\, (x)\, F f\, \mathbf{0}$, avec $F = \lambda g \lambda x\, (g)\, (s)\, x$. Pour tout terme φ, et tout $n \in \mathbb{N}$, on a $(!\mathbb{N})\, n\, \varphi \simeq (\varphi)\, n$.

2^e *exemple :* On cherche Ψ_1 telle que $(\Psi_n)\,n = \lambda f\,(f)\,(s)^n\,\mathbf{0}$, et on définira ψ en posant $(\psi)\,x = (\Psi_1)\,x\,\varphi$.

On définit Ψ_1 par récurrence :

$$(\Psi_1)\,0 = \lambda g\,(g)\,\mathbf{0} = \delta_0; \qquad (\psi_1)\,(s)\,n = (G)\,(\Psi_1)\,n \qquad \text{avec} \quad (G)\,h = \lambda f\,(h)\,f \circ s.$$

On a donc $(\Psi_1)\,x = (x)\,G\,\delta_0$.

Un autre candidat $!\,\mathbb{N}_1$ pour un opérateur de mise en mémoire d'un entier de Church est donc défini par :

$$!\,\mathbb{N}_1 = \lambda x \lambda f\,(x)\,G\,\delta_0 f \qquad \text{avec} \quad G = \lambda h \lambda f\,(h)\,\lambda x\,(f)\,(s)\,x \quad \text{et} \quad \delta_0 = \lambda g\,(g)\,\mathbf{0}.$$

On a encore $!\,\mathbb{N}_1\,n\,\varphi \simeq (\varphi)\,n$, pour tout terme φ et tout entier n.

DÉFINITION : Un terme T du λ-calcul sera appelé *opérateur de mise en mémoire d'un entier de Church* si, pour tout entier $\nu\,(\nu \simeq \lambda f \lambda x\,(f)^n\,x,\ n \in \mathbb{N})$ le terme $T\nu$ se réduit à $\lambda f\,(f)\,(s)^n\,\mathbf{0}$ *par réduction de tête.*

D'après le théorème 1, on voit que, pour tout terme φ, l'évaluation paresseuse de $(T\nu)\,\varphi$ revient à faire d'abord la réduction de tête de $T\nu$, ce qui donne $\lambda f\,(\,f)\,(s)^n\,\mathbf{0}$, puis à évaluer $(\lambda f\,(\,f)\,(s)^n\,\mathbf{0})\,\varphi$, c'est-à-dire $(\varphi)\,(s)^n\,\mathbf{0}$. L'entier ν a donc été calculé d'abord (sous la forme $(s)^n\,\mathbf{0}$).

Montrons maintenant que les deux termes $!\,\mathbb{N}$ et $!\,\mathbb{N}_1$ définis plus haut sont des opérateurs de mise en mémoire d'un entier. Soit ν un entier de Church (non réduit), $\nu \simeq \lambda f \lambda x\,(\,f)^n\,x$. Par réduction de tête, $!\,\mathbb{N}\,\nu$ donne d'abord $\lambda f\,(\nu)\,Ff\,\mathbf{0}$; ensuite, en appliquant le théorème 1, on effectue d'abord la réduction de tête de ν. Si $\nu \simeq \mathbf{0}$, on aboutit à $\mathbf{0}$, donc la réduction de tête de $\lambda f\,(\nu)\,Ff\,\mathbf{0}$ donne $\lambda f\,(\,f)\,\mathbf{0}$, ce qui est le résultat cherché. Sinon, ν est ramené à sa f. n. t. principale, qui est de la forme $\lambda g \lambda x\,(g)\,t$, avec $t \simeq (g)^{n-1}\,x$. La réduction de tête de $\lambda f\,(\nu)\,Ff\,\mathbf{0}$ donne alors $\lambda f\,(F)\,t'\,\mathbf{0}$, puis $\lambda f\,(t')\,(s)\,\mathbf{0}$, avec $t' = t\,[F/g, f/x]$.

Posons $\nu' = \lambda g \lambda x t$; alors ν' est un entier de Church, $\nu' \simeq \lambda g \lambda x\,(g)^{n-1}\,x$.

Or, la réduction de tête de $\lambda f\,(\nu')\,Ff\,(s)\,\mathbf{0}$ donne, en deux pas, le terme $\lambda f\,(t')\,(s)\,\mathbf{0}$. Tout se passe donc comme si on repartait du terme $\lambda f\,(\nu')\,Ff\,(s)\,\mathbf{0}$. On va donc obtenir successivement les termes $\lambda f\,(\nu)\,Ff\,\mathbf{0}$, $\lambda f\,(\nu')\,Ff\,(s)\,\mathbf{0}$, $\lambda f\,(\nu'')\,Ff\,(s)^2\,\mathbf{0}$, . . . On aboutit finalement à $\lambda f\,(\zeta)\,Ff\,(s)^n\,\mathbf{0}$, avec $\zeta \simeq \mathbf{0}$, puis, en appliquant le théorème 1, à $\lambda f\,(\mathbf{0})\,Ff\,(s)^n\,\mathbf{0}$, et finalement à $\lambda f\,(\,f)\,(s)^n\,\mathbf{0}$, ce qui est le résultat voulu.

$!\,\mathbb{N}$ est donc bien un opérateur de mise en mémoire pour les entiers de Church.

La méthode est la même pour l'évaluation de $!\mathbb{N}_1\,\nu$. On trouve d'abord $\lambda f\,(\nu)\,G\,\delta_0 f$. Si $\nu \simeq \mathbf{0}$, on obtient δ_0 c'est-à-dire $\lambda f\,(f)\,\mathbf{0}$. Sinon, ν est ramené à sa f. n. t. principale $\lambda g\,\lambda x\,(g)\,t$, avec $t \simeq (g)^{n-1}\,x$. La réduction de tête de $\lambda f\,(\nu)\,G\,\delta_0 f$ donne alors $\lambda f(G)\,t'f$, puis $\lambda f\,(t')f \circ s$, avec $t' = t[G/g, f/x]$.

Posons encore $\nu' = \lambda g\,\lambda x\,t$; alors $\nu' \simeq \lambda g\,\lambda x\,(g)^{n-1}\,x$. La réduction de tête de $\lambda f\,(\nu')\,G\,\delta_0 f \circ s$ donne, en deux pas, le terme $\lambda f\,(t')f \circ s$. Tout se passe donc comme si on repartait du terme $\lambda f\,(\nu')\,G\,\delta_0 f \circ s$. On va donc obtenir successivement les termes $\lambda f\,(\nu)\,G\,\delta_0 f$, $\lambda f\,(\nu')\,G\,\delta_0 f \circ s$, $\lambda f\,(\nu'')\,G\,\delta_0 f \circ s^2$, ... On aboutit finalement à $\lambda f\,(\zeta)\,G\,\delta_0 f \circ s^n$, avec $\zeta \simeq \mathbf{0}$; puis, en appliquant le théorème 1, à $\lambda f\,(\mathbf{0})\,G\,\delta_0 f \circ s^n$, et finalement à $\lambda f\,(f)\,(s)^n\,\mathbf{0}$.

Remarque : On voit que, par réduction de tête, $(!\,\mathbb{N})\,\nu\varphi$ et $(!\,\mathbb{N}_1)\,\nu\varphi$ se ramènent à $(\varphi)\,(s)^n\,\mathbf{0}$ pour tout terme, φ. Toutefois, cette réduction ne s'est pas faite exclusivement dans $!\,\mathbb{N}\,\nu$ ou $!\,\mathbb{N}_1\,\nu$. Il est facile d'écrire un opérateur de mise en mémoire pour lequel ceci se produira : il suffit de considérer le cas particulier $\varphi = \lambda x\,\lambda f f x$. Posons, par exemple, $!\,\mathbb{N}' = \lambda x\,(!\,\mathbb{N}\,x)\,\lambda x\,\lambda f f x$.

D'après ce qu'on vient de montrer, le terme $!\,\mathbb{N}'\,\nu$ se ramène, par réduction de tête, à $(\lambda x\,\lambda f f x)\,(s)^n\,\mathbf{0}$. Comme ce terme ne commence pas par λ, il en est de même de tous les termes intermédiaires dans la réduction. Par suite, la réduction de tête de $(!\,\mathbb{N}'\,\nu)\,\varphi$ se fait seulement dans $!\,\mathbb{N}'\,\nu$, jusqu'à aboutir à $(\lambda x\,\lambda f f x)\,(s)^n\,\mathbf{O})\,\varphi$; ce qui donne ensuite $(\varphi)\,(s)^n\,\mathbf{0}$.

OPÉRATEUR DE MISE EN MÉMOIRE POUR UN ENSEMBLE DE TERMES

Soit E un ensemble de termes clos du λ-calcul, saturé pour la β-équivalence ($t \in E$, $t' \simeq t \Rightarrow t' \in E$). Un terme T sera appelé *opérateur de mise en mémoire pour les termes de E*, si, pout tout $\nu \in E$, le terme $T\nu$ se réduit, *par réduction de tête*, à $\lambda f\,(f)\,\nu^*$, avec $\nu^* \simeq \nu$, ν^* ne dépendant que de la classe d'équivalence de ν (si μ, $\nu \in E$, et $\mu \simeq \nu$, alors $\mu^* = \nu^*$).

On a déjà considéré deux exemples : le cas où E est l'ensemble des (termes équivalents à des) booléens, ou l'ensemble des entiers de Church. On va considérer maintenant le produit et la somme de deux ensembles de termes.

Soient U, V deux ensembles des termes clos du λ-calcul, saturés pour la β-équivalence, pour lesquels il existe des opérateurs de mise en mémoire, notés $!\,U$ et $!\,V$.

Le produit $U \times V = W$ est, par définition, l'ensemble des termes équivalents à $\lambda f\,(f)\,uv$ pour $u \in U$ et $v \in V$.

Il existe alors un opérateur de mise en mémoire pour W; on peut prendre, par exemple : $!\,W = \lambda z\,(z)\,\lambda u\,\lambda v\,\lambda f\,(!\,U u)\,\lambda x\,(!\,V v)\,\lambda y\,(f)\,\lambda g\,(g)\,xy$.

Soit en effet $\nu = \lambda f(f)uv$, avec $u \in U$, $v \in V$. Alors $(!\,W)\nu$ donne, par réduction de tête, $\lambda f(!\,U u)\lambda x(!\,V v)\lambda y(f)\lambda g(g)xy$. Comme $(!\,U)u$ donne par réduction de tête $\lambda g(g)u^*$ et $(!\,V)v$ donne $\lambda g(g)v^*$, on voit en appliquant le théorème 1, que la réduction de tête de $(!\,W)\nu$ donne le même résultat que celles des termes suivants : $\lambda f(\lambda g(g)u^*)\lambda x(!\,V v)\lambda y(f)\lambda g(g)xy$, puis $\lambda f(!\,V v)\lambda y(f)\lambda g(g)u^* y$, puis $\lambda f(\lambda g(g)v^*)\lambda y(f)\lambda g(g)u^* y$, et enfin $\lambda f(f)\lambda g(g)u^* v^*$, c'est-à-dire $\lambda f(f)\nu^*$, où $\nu^* = \lambda g(g)u^* v^* \simeq \nu$ ne dépend que de la classe d'équivalence de ν.

Notons deux autres exemples d'opérateurs de mise en mémoire pour $U \times V$:

$$!\,W' = \lambda z \lambda f(z)\lambda u \lambda v(!\,U u)\lambda x(!\,V v)\lambda y(f)\lambda g(g)xy,$$

et

$$!\,W'' = \lambda z \lambda f((!\,U)(z)\mathbf{1})\lambda x((!\,V)(z)\mathbf{0})\lambda y(f)\lambda g(g)xy.$$

Désignons maintenant par W l'ensemble somme $U + V$ qui est l'ensemble des termes équivalents à $\lambda g \lambda h(g)u$ ou à $\lambda g \lambda h(h)v$ avec $u \in U$ et $v \in V$.

Il existe alors un opérateur de mise en mémoire pour W; on peut prendre, par exemple :

$$!\,W = \lambda z[(z)\lambda u \lambda f(!\,U u)\lambda x(f)\lambda g \lambda h(g)x]\lambda v \lambda f(!\,V v)\lambda y(f)\lambda g \lambda h(h)y.$$

Soit en effet $\nu = \lambda g \lambda h(g)u$, avec $u \in U$. Alors $(!\,W)\nu$ donne, par réduction de tête,

$$[(\lambda g \lambda h(g)u)\lambda u \lambda f(!\,U u)\lambda x(f)\lambda g \lambda h(g)x]\lambda v \lambda f(!\,V v)\lambda y(f)\lambda g \lambda h(h)y,$$

puis

$$\lambda f(!\,U u)\lambda x(f)\lambda g \lambda h(g)x.$$

Comme $!\,U$ est un opérateur de mise en mémoire, $!\,U u$ donne, par réduction de tête, $\lambda g(g)u^*$. D'après le théorème 1, la réduction de tête de $!\,W\nu$ donne donc le même résultat que celle de $\lambda f(\lambda g(g)u^*)\lambda x(f)\lambda g \lambda h(g)x$, soit $\lambda f(f)\lambda g \lambda h(g)u^*$, ou encore $\lambda f(f)\nu^*$; et $\nu^* = \lambda g \lambda h(g)u^* \simeq \nu$ ne dépend que de la classe d'équivalence de ν.

Notons deux autres exemples d'opérateurs de mise en mémoire pour $U + V$:

$$!\,W' = \lambda z \lambda f[(z)\lambda u(!\,U u)\lambda x(f)\lambda g \lambda h(g)x]\lambda v(!\,V v)\lambda y(f)\lambda g \lambda h(h)y.$$

$$!\,W'' = \lambda z \lambda f[(((z)\lambda x\mathbf{1})\lambda x\mathbf{0})((!\,U)(z)\mathrm{II})\lambda x(f)\lambda g \lambda h(g)x]$$
$$((!\,V)(z)\mathrm{II})\lambda x(f)\lambda g \lambda h(h)y, \qquad \mathrm{I} = \lambda xx.$$

On se propose maintenant de donner une méthode pour construire des termes de mise en mémoire.

LA LOGIQUE INTUITIONNISTE DU SECOND ORDRE

Nous utiliserons un système de λ-calcul typé, basé sur la logique intuitionniste du deuxième ordre. Ce système est étudié dans [3 à 6]. Donnons brièvement ici la description de sa syntaxe, ce qui suffira pour l'utilisation que nous avons en vue.

Dans ce système, les types sont les formules d'un langage $\mathscr{L}$ du calcul des prédicats du second ordre, constitué des symboles suivants :

– les symboles logiques qui sont le connecteur $\rightarrow$ et le quantificateur $\forall$;

– des variables d'individu : $x, y, z, \ldots$

– des variables de relation : $X, Y, Z, \ldots$ A chacune de ces variables est attaché un entier $\geqq 0$, qui est son arité. Les variables d'arité 0 sont appelées variables propositionnelles;

– des symboles de fonction (sur les individus) : $f, h, \ldots$ A chacun de ces symboles est attaché un entier $\geqq 0$ qui est son arité. Les symboles de fonction d'arité 0 sont appelés symboles de constante.

Les termes de $\mathscr{L}$ sont construits par les règles suivantes :

– une variable d'individu ou un symbole de constante de $\mathscr{L}$ est un terme de $\mathscr{L}$;

– si f est un symbole de fonction de $\mathscr{L}$, d'arité n, et si $t_1, \ldots, t_n$ sont des termes de $\mathscr{L}$, alors $f(t_1, \ldots, t_n)$ est un terme de $\mathscr{L}$.

Les types (ou formules de $\mathscr{L}$ du second ordre) sont construits par les règles suivantes :

– si X est une variable de relation d'arité n, et si $t_1, \ldots, t_n$ sont des termes de $\mathscr{L}$, alors $X(t_1, \ldots, t_n)$ est un type (appelé formule ou type atomique); en particulier, toute variable propositionnelle est un type;

– si A, B sont deux types, $A \rightarrow B$ est un type;

– si A est un type, et x (resp. X) est une variable d'individus (resp. de relation), alors $\forall x\, A$ (resp. $\forall X A$) est un type.

La formule $\perp$ est, par définition $\forall XX$, où X est une variable propositionnelle; si A est une formule, la formule $\neg A$ est, par définition, $A \rightarrow \perp$.

Nous utiliserons la notation $A, B, C \rightarrow D$ pour $A \rightarrow (B \rightarrow (C \rightarrow D))$.

Un contexte Γ est, par définition, une fonction dont le domaine est un ensemble fini de variables du λ-calcul, à valeurs dans les types; on le note

sous la forme $x_1 : A_1, \ldots, x_k : A_k$, où $x_1, \ldots, x_k$ sont des variables du λ-calcul, deux à deux distinctes. On dira que x_i est déclarée de type A_i dans le contexte Γ. L'entier k peut être nul, auquel cas on a le « contexte vide ».

Remarque : Ne pas confondre les variables d'individu du langage $\mathscr{L}$, et les variables du λ-calcul, bien qu'elles soient écrites de la même façon. De même, ne pas confondre les termes de $\mathscr{L}$, et les termes du λ-calcul.

On désignera par $\Gamma, x : A$ le contexte obtenu en ajoutant la déclaration $x : A$ au contexte Γ. Cette notation suppose que x n'est pas déjà déclarée dans Γ.

Étant donnés un terme t du λ-calcul, un type A, et un contexte Γ, on définit, au moyen des « règles de typage » suivantes [(T1) à (T7)], la notion : « t est de type A dans le contexte Γ », ce que l'on note $\Gamma \vdash t : A$.

(T1) Si x est une variable du λ-calcul, alors $\Gamma, x : A \vdash x : A$.

(T2) Si $\Gamma, x : A \vdash t : B$, alors $\Gamma \vdash \lambda x\, t : A \to B$.

(T3) Si $\Gamma \vdash t : A$ et $\Gamma \vdash u : A \to B$, alors $\Gamma \vdash (u)\, t : B$.

(T4) Si $x_1 : A_1, \ldots, x_k : A_k \vdash t : A$ et si la variable d'individu x n'est pas libre dans $A_1, \ldots, A_k$, alors $x_1 : A_1, \ldots, x_k : A_k \vdash t : \forall x\, A$.

(T5) Si $\Gamma \vdash t : \forall x\, A\,[x]$, et si u est un terme du langage $\mathscr{L}$, alors $\Gamma \vdash t : A\,[u]$.

(T6) Si $x_1 : A_1, \ldots, x_k : A_k \vdash t : A$ et si la variable de relation X n'est pas libre dans $A_1, \ldots, A_k$, alors $x_1 : A_1, \ldots, x_k : A_k \vdash t : \forall X A$.

(T7) Si $\Gamma \vdash t : \forall X A\,[X]$, X étant une variable de relation n-aire, et si $F[x_1, \ldots, x_n]$ est une formule de $\mathscr{L}$ ($x_1, \ldots, x_n$ étant des variables d'individu), alors $\Gamma \vdash t : A\,[F]$.

$A\,[F]$ désigne la formule obtenue en remplaçant dans A, chaque formule atomique $X(t_1, \ldots, t_n)$ commençant par la variable X, par la formule $F[t_1, \ldots, t_n]$.

Remarque : Ce système de λ-calcul typé est une extension du « système $\mathscr{F}$ » défini dans [2]. On vérifie aisément qu'il permet de typer exactement les mêmes termes du λ-calcul que le système $\mathscr{F}$. Il vérifie donc le théorème de normalisation : tout terme du λ-calcul typable dans ce système est fortement normalisable (*cf.* [2]).

Soient A une formule, et $\mathscr{A} = \{A_1, \ldots, A_k\}$ un ensemble fini de formules de $\mathscr{L}$. On définit l'expression « *A est conséquence de $\mathscr{A}$ en logique classique du second ordre* », ce qu'on note $\mathscr{A} \vdash A$, au moyen des « règles de

démonstration » suivantes :

(D0) On a $\mathscr{A} \vdash \neg\neg A \to A$, pour toute formule A.

(D1) Si $A \in \mathscr{A}$, alors on a $\mathscr{A} \vdash A$.

(D2) Si $\mathscr{A}, A \vdash B$, alors $\mathscr{A} \vdash A \to B$.

(D3) Si $\mathscr{A} \vdash A$ et $\mathscr{A} \vdash A \to B$, alors $\mathscr{A} \vdash B$.

(D4) Si $\mathscr{A} \vdash A$ et si la variable d'individu x n'est libre dans aucune formule de $\mathscr{A}$, alors $\mathscr{A} \vdash \forall x A$.

(D5) Si $\mathscr{A} \vdash \forall x A[x]$, alors $\mathscr{A} \vdash A[u]$ pour tout terme u de $\mathscr{L}$.

(D6) Si $\mathscr{A} \vdash A$ et si la variable X de relation n-aire n'est libre dans aucune formule de $\mathscr{A}$, alors $\mathscr{A} \vdash \forall X A$.

(D7) Si $\mathscr{A} \vdash \forall X A[X]$, X étant une variable de relation n-aire, et si $F[x_1, \ldots, x_n]$ est une formule de $\mathscr{L}$ ($x_1, \ldots, x_n$ étant des variables d'individu), alors $\mathscr{A} \vdash A[F]$.

On définit l'expression « *A est conséquence de $\mathscr{A}$ en logique intuitionniste du second ordre* », ce qu'on notera $\mathscr{A} \vdash^i A$, au moyen des règles de démonstration (D1) à (D7) ci-dessus.

L'énoncé suivant, qui est une forme de ce qu'on appelle « l'isomorphisme de Curry-Howard », est alors évident, d'après ces définitions :

Pour qu'il existe un terme t du λ-calcul tel que $x_1 : A_1, \ldots, x_k : A_k \vdash t : A$, il faut et il suffit que A soit conséquence intuitionniste de $\{A_1, \ldots, A_k\}$.

De fait, les constructions de termes typés correspondent, de façon canonique et évidente, aux démonstrations intuitionnistes à l'aide des règles (D1) à (D7).

Il est clair que, si A est conséquence intuitionniste de $\mathscr{A}$, A est aussi conséquence classique de $\mathscr{A}$. La traduction de Gödel (ou $\neg\neg$-traduction) permet d'énoncer une sorte de réciproque.

On se donne un nouveau symbole O de prédicat 0-aire (variable propositionnelle) qui n'est pas dans $\mathscr{L}$. Si F est une formule, $\neg_0 F$ est, par définition, la formule $F \to O$.

A chaque formule F de $\mathscr{L}$, on associe la « traduction de Gödel de F », que nous noterons F^*, définie par les règles suivantes :

- si F est atomique, F^* est $\neg_0 \neg_0 F$;
- $(F \to G)^*$ est $F^* \to G^*$;
- $(\forall x F)^*$ est $\forall x F^*$; $(\forall X F)^*$ est $\forall X F^*$.

La traduction de Gödel de F est donc la formule F^* obtenue en remplaçant dans F chaque formule atomique A par $\neg_0 \neg_0 A$.

Le théorème suivant est un résultat standard :

THÉORÈME : *Si* $A_1, \ldots, A_k \vdash A$ *(en logique classique), alors* $A_1^*, \ldots, A_k^* \vdash^i A^*$ *(en logique intuitionniste).*

LES TYPES DE DONNÉES

Un type de données est une formule $D[x]$, à une variable libre d'individu, obtenue de la façon suivante : considérons un langage $\mathscr{L}$, et ajoutons-lui de nouveaux symboles de fonction : $f_1, \ldots, f_n$, d'arités respectives $k_1+m_1, \ldots, k_n+m_n$, ce qui donne un langage $\mathscr{L}'$. Soit X une variable de relation unaire. A chacun des nouveaux symboles f_j, on associe une formule :

$$\Delta_j[X] \equiv \forall x_1 \ldots \forall x_{k_j} \forall y_1 \ldots \forall y_{m_j}$$
$$\{ D_1^j[x_1], \ldots, D_{k_j}^j[x_{k_j}], Xy_1, \ldots, Xy_{m_j} \to Xf_j(x_1, \ldots, x_{k_j}, y_1, \ldots, y_{m_j}) \}$$

où chaque $D_i^j[x_i]$ est un type de données déjà construit dans le langage $\mathscr{L}$.

$\Delta_j[X]$ exprime que « l'ensemble X est clos par la fonction f_j ».

La formule $D[x]$ pour le nouveau type de données le définit comme le plus petit ensemble clos par les fonctions f_j. Cette formule s'écrit donc :

$$D[x] \equiv \forall X \{ \Delta_1[X], \ldots, \Delta_n[X] \to Xx \}.$$

Les nouveaux symboles de fonction f_j représentent les constructeurs du type $D[x]$ au-dessus des types de données $D_i^j[x]$ déjà construits.

Considérons un type de données $D[x]$; on lui associe l'ensemble $\mathscr{T}(D)$, défini inductivement comme l'ensemble des termes clos de $\mathscr{L}'$ de la forme $f_j(t_1, \ldots, t_{k_j}, u_1, \ldots, u_{m_j})$, avec $t_i \in \mathscr{T}(D_i^j)$ et $u_i \in \mathscr{T}(D)$. C'est l'ensemble des termes τ de $\mathscr{L}'$ tels que $\vdash D[\tau]$ (« algèbre de termes » associée au type $D[x]$).

On associe également au type de données $D[x]$, un ensemble $\Lambda(D)$ de termes du λ-calcul, saturé par β-équivalence, défini par : $t \in \Lambda(D) \Leftrightarrow$ il existe un terme u du λ-calcul, $u \simeq t$, et $\tau \in \mathscr{T}(D)$ tel que $\vdash u : D[\tau]$.

$\Lambda(D)$ sera appelé, par abus de langage, « l'ensemble des termes de type D ».

Exemples : booléens, entiers, produit, somme, listes.

Le type booléen est la formule :

$$\mathrm{Bool}[x] \equiv \forall X \{ X1, X0 \to Xx \};$$

les symboles de fonction f_i sont les symboles de constante 0, 1 (pour les deux booléens).

$\mathscr{T}(D) = \{0, 1\}$ et $\Lambda(D)$ est l'ensemble des termes du λ-calcul β-équivalents à **0**, **1** (*cf.* [4, 5]).

Le type entier est la formule :

$$\mathrm{Ent}[x] \equiv \forall X \{ \forall y (Xy \to Xsy), X0 \to Xx \};$$

les symboles de fonction sont s (fonction unaire, pour le successeur) et 0 (constante). $\mathscr{T}(D)$ est l'ensemble des $s^n 0$, et $\Lambda(D)$ est l'ensemble des entiers de Church [4, 5].

Soient $U[x]$, $V[x]$ deux types de données; le type produit de U, V est la formule :

$$PUV[x] \equiv \forall X \{ \forall y \forall z (U[y], V[z] \to Xc(y, z)) \to Xx \};$$

on a ajouté le symbole de fonction binaire c, pour le couple ordonné.

$\mathscr{T}(PUV)$ est l'ensemble des termes $c(u, v)$ avec $u \in \mathscr{T}(U)$ et $v \in \mathscr{T}(V)$; $\Lambda(PUV)$ est $\Lambda(U) \times \Lambda(V)$.

Le type somme de U, V est la formule :

$$SUV[x] \equiv \forall X \{ \forall y (U[y] \to Xiy), \forall z (V[z] \to Xjz) \to Xx \};$$

on a ajouté les symboles de fonction unaires i, j (injections canoniques de U, V dans la somme). $\mathscr{T}(SUV)$ est l'ensemble des termes $i(u), j(v)$ avec $u \in \mathscr{T}(V)$ et $v \in \mathscr{T}(V)$; $\Lambda(PUV)$ est $\Lambda(U) + \Lambda(V)$.

Le type des listes (suites finies) d'éléments de type U est la formule :

$$LU[x] \equiv \forall X \{ \forall y \forall z (U[y], Xz \to X\mathrm{cons}(y, z)), X0 \to Xx \};$$

on a ajouté le symbole de constante 0 (pour la liste vide) et le symbole de fonction binaire cons. $\mathscr{T}(D)$ est l'ensemble des termes $\mathrm{cons}(u_1, \mathrm{cons}(u_2, \ldots, \mathrm{cons}(u_n, 0)\ldots))$, avec $u_i \in \mathscr{T}(U)$. $\Lambda(D)$ est formé des termes β-équivalents à $\lambda f \lambda x (\ldots ((f) u_1)(f) u_2 \ldots (f) u_n) x$ avec $u_i \in \Lambda(U)$.

LES TERMES DE MISE EN MÉMOIRE ET LA TRADUCTION DE GÖDEL

On va donner maintenant une méthode pour construire les opérateurs de mise en mémoire, au moyen de la traduction de Gödel des formules.

LEMME : *Si* $\vdash^i A_1, \ldots, A_k \to A$, *alors*

$$\vdash^i \neg_0 \neg_0 A_i, \ldots, \neg_0 \neg_0 A_k \to \neg_0 \neg_0 A.$$

On a les hypothèses $(A_j \to O) \to O\,(1 \leqq j \leqq k)$, $A \to O$ et on veut démontrer O. On a $A_1, \ldots, A_k \to A$, et $A \to O$, donc $A_1, \ldots, A_k \to O$; on raisonne alors par induction sur k; on a $A_1, \ldots, A_{k-1} \to (A_k \to O)$; on a aussi $(A_k \to O) \to O$, d'où l'on déduit $A_1, \ldots, A_{k-1} \to O$. D'où le résultat par hypothèse d'induction.

C.Q.F.D.

PROPOSITION 3 : *Si* $D[x]$ *est un type de données, alors*

$$\vdash^i \forall x \{ D^*[x] \to \neg_0 \neg_0 D[x] \}.$$

Le type $D[x]$ s'écrit $\forall X \{ \Delta_1[X], \ldots, \Delta_n[X] \to X x \}$; donc :

$$D^*[x] \equiv \forall X \{ \Delta_1^*[X], \ldots, \Delta_n^*[X] \to \neg_0 \neg_0 X x \}.$$

En substituant à Xx la formule $D[x]$ [règle (D7)], on voit que $D^*[x] \vdash^i \Delta_1^*[D], \ldots, \Delta_n^*[D] \to \neg_0 \neg_0 D[x]$. On obtiendra donc le résultat cherché en montrant $\vdash^i \Delta_j^*[D]$ pour $1 \leqq j \leqq n$. Or on a :

$$\Delta_j[X] \equiv \forall x_1 \ldots \forall x_{k_j} \forall y_1 \ldots \forall y_{m_j}$$
$$\{ D_1^j[x_1], \ldots, D_{k_j}^j[x_{k_j}], Xy_1, \ldots, Xy_{m_j} \to X f_j(x_1, \ldots, x_{k_j}, y_1, \ldots, y_{m_j}) \}$$

où chaque $D_i^j[x_i]$ est un type de données déjà construit dans le langage $\mathscr{L}$.

Donc $\Delta_j^*[D]$ s'écrit :

$$\forall x_1 \ldots \forall x_{k_j} \forall y_1 \ldots \forall y_{m_j}$$
$$\{ D_1^{j*}[x_1], \ldots, D_{k_j}^{j*}[x_{k_j}], \neg_0 \neg_0 D[y_1], \ldots, \neg_0 \neg_0 D[y_{m_j}]$$
$$\to \neg_0 \neg_0 D[f_j(x_1, \ldots, x_{k_j}, y_1, \ldots, y_{m_j})] \}.$$

Comme $D_i^j[x]$ est un type de données, on a (hypothèse d'induction) $D_i^{j*}[x] \vdash^i \neg_0 \neg_0 D_i^j[x]$.

Pour montrer $\vdash^i \Delta_j^*[D]$, il suffit donc de montrer :

$$\vdash^i \neg_0 \neg_0 D_1^j[x_1], \ldots, \neg_0 \neg_0 D_{k_j}^j[x_{k_j}], \neg_0 \neg_0 D[y_1], \ldots, \neg_0 \neg_0 D[y_{m_j}]$$
$$\to \neg_0 \neg_0 D[f_j(x_1, \ldots, x_{k_j}, y_1, \ldots, y_{m_j})].$$

D'après le lemme précédent, on est ramené à démontrer :

$$\vdash^i D_1^j[x_1], \ldots, D_{k_j}^j[x_{k_j}], D[y_1], \ldots, D[y_{m_j}] \rightarrow D[f_j(x_1, \ldots, x_{k_j}, y_1, \ldots, y_{m_j})];$$

ou encore, par définition de $D[f_j(x_1, \ldots, x_{k_j}, y_1, \ldots, y_{m_j})]$:

$$\vdash^i D_1^j[x_1], \ldots, D_{k_j}^j[x_{k_j}], D[y_1], \ldots, D[y_{m_j}], \Delta_1[X], \ldots, \Delta_n[X] \rightarrow X f_j(x_1, \ldots, x_{k_j}, y_1, \ldots, y_{m_j}).$$

Or, on a $D[y_i], \Delta_1[X], \ldots, \Delta_n[X] \vdash X y_i$, par définition de $D[y_i]\ (1 \leqq i \leqq m_j)$.

Il reste donc à montrer :

$$D_1^j[x_1], \ldots, D_{k_j}^j[x_{k_j}], X y_1, \ldots, X y_{m_j}, \Delta_1[X], \ldots, \Delta_n[X] \vdash^i X f_j(x_1, \ldots, x_{k_j}, y_1, \ldots, y_{m_j}).$$

Or cette dernière assertion est évidente, car on a, en fait, par définition de $\Delta_j[X]$:

$$D_1^j[x_1], \ldots, D_{k_j}^j[x_{k_j}], X y_1, \ldots, X y_{m_j}, \Delta_j[X] \vdash^i X f_j(x_1, \ldots, x_{k_j}, y_1, \ldots, y_{m_j}).$$

C.Q.F.D.

Cette démonstration intuitionniste de $\vdash^i \forall x \{ D^*[x] \rightarrow \neg_0 \neg_0 D[x] \}$ donne, par l'ismorphisme de Curry-Howard décrit ci-dessus, un terme clos $!D$ du λ-calcul, de type $\forall x \{ D^*[x] \rightarrow \neg_0 \neg_0 D[x] \}$.

On vérifie que *ce terme est un opérateur de mise en mémoire pour l'ensemble* $\Lambda(D)$ *des termes du* λ*-calcul « de type* D *»*. Nous ferons cette vérification, par exemple, pour les types $\mathrm{Bool}[x]$, $\mathrm{Ent}[x]$, et $PUV[x]$:

On a $\mathrm{Bool}^*[x] \equiv \forall X \{ \neg_0 \neg_0 X 1, \neg_0 \neg_0 X 0 \rightarrow \neg_0 \neg_0 X x \}$; on a donc

$$z : \mathrm{Bool}^*[x] \vdash z : \neg_0 \neg_0 \mathrm{Bool}[1], \neg_0 \neg_0 \mathrm{Bool}[0] \rightarrow \neg_0 \neg_0 \mathrm{Bool}[x] \text{ (règle T7).}$$

Or $\vdash \mathbf{1} : \mathrm{Bool}[1]$, et donc $\vdash \lambda f f) \mathbf{1} : (\mathrm{Bool}[1] \rightarrow O) \rightarrow O$; de même $\vdash \lambda f (f) \mathbf{0} : (\mathrm{Bool}[0] \rightarrow O) \rightarrow O$; il en résulte que :

$$\lambda z ((z) \lambda f (f) \mathbf{1}) \lambda f (f) \mathbf{0} \vdash \mathrm{Bool}^*[x] \rightarrow \neg_0 \neg_0 \mathrm{Bool}[x].$$

On retrouve le deuxième terme de mise en mémoire des booléens considéré plus haut.

On a

$$\mathrm{Ent}^*[x] \equiv \forall X [\forall y \neg_0 \neg_0 X y \rightarrow \neg_0 \neg_0 X s y), \neg_0 \neg_0 X 0, \rightarrow \neg_0 \neg_0 X x].$$

Par suite, en substituant la formule $\mathrm{Ent}[x]$ à Xx on trouve (règle T7) :

$$z:\mathrm{Ent}^*[x]\vdash z:\forall y(\neg_0\neg_0\mathrm{Ent}[y]\to\neg_0\neg_0\mathrm{Ent}[sy]),$$
$$\neg_0\neg_0\mathrm{Ent}[0]\to\neg_0\neg_0\mathrm{Ent}[x].$$

Or $\delta_0=\lambda f(f)\mathbf{0}$ est du type $\neg_0\neg_0\mathrm{Ent}[0]$. Par ailleurs, le terme $s=\lambda n\lambda f\lambda x(f)(n)fx$ est du type $\mathrm{Ent}[y]\to\mathrm{Ent}[sy]$. Donc $g:\mathrm{Ent}[sy]\to O\vdash g\circ s:\mathrm{Ent}[y]\to O$, et, par suite :

$$h:(\mathrm{Ent}[y]\to O)\to O, g:\mathrm{Ent}[sy]\to O\vdash(h)g\circ s:O.$$

Par suite $G=\lambda h\lambda g(h)g\circ s$ est du type $\neg_0\neg_0\mathrm{Ent}[y]\to\neg_0\neg_0\mathrm{Ent}[sy]$, et le terme $\lambda z(z)G\delta_0$ est du type $\mathrm{Ent}^*[x]\to\neg_0\neg_0\mathrm{Ent}[x]$. On retrouve, là encore, le deuxième exemple d'opérateur de mise en mémoire des entiers de Church.

Soient $U[x]$, $V[x]$ deux types de données; le type produit est :

$$PUV[x]\equiv\forall X\{\forall y\forall z(U[y],V[z]\to Xc(y,z))\to Xx\}.$$

Donc :

$$PUV^*[X]\equiv\forall X\{\forall y\forall z(U^*[y],V^*[z]\to\neg_0\neg_0 Xc(y,z))\to\neg_0\neg_0 Xx\}.$$

En substituant $PUV[x]$ à Xx, on a donc :

$$z:PUV^*[X]\vdash z:\forall y\forall z(U^*[y],V^*[z]$$
$$\to\neg_0\neg_0 PUV[c(y,z)])\to\neg_0\neg_0 PUV[x].$$

On cherche un terme F de type $U^*[y],V^*[z]\to\neg_0\neg_0 PUV[c(y,z)]$; le terme cherché sera alors $!PUV=\lambda z(z)F$.

On a $y:U[y],z:V[z]\vdash\lambda g(g)yz:PUV[c(y,z)]$; donc

$$y:U[y],z:V[z],f:PUV[c(y,z)]\to O\vdash(f)\lambda g(g)yz:O,$$

et

$$y:U[y],f:PUV[c(y,z)]\to O\vdash\lambda z(f)\lambda g(g)yz:V[z]\to O.$$

On a, par hypothèse,

$$\vdash !U:U^*[y]\to\neg_0\neg_0 U[y]\qquad\text{et}\qquad\vdash !V:V^*[z]\to\neg_0\neg_0 V[z].$$

Donc :

$$u:U^*[y],v:V^*[z]\vdash !Uu:\neg_0\neg_0 U[y]\qquad et\qquad !Vv:\neg_0\neg_0 Vz$$

Donc :

$$u : U^*[y], v : V^*[z], y : U[y], f : PUV[c(y, z)] \to O \vdash (!\, V v)\, \lambda z\, (f)\, \lambda g\, (g)\, yz : O;$$

donc :

$$u : U^*[y], v : V^*[z], f : PUV[c(y, z)] \to O \vdash \lambda y\, (!\, V v)\, \lambda z\, (f)\, \lambda g\, (g)\, yz : U[y] \to O;$$

puis

$$u : U^*[y], v : V^*[z], f : PUV[c(y, z)] \to O \vdash (!\, U u)\, \lambda y\, (!\, V v)\, \lambda z\, (f)\, \lambda g\, (g)\, yz : O;$$

d'où le terme cherché $F = \lambda u \lambda v \lambda f\, (!\, U u)\, \lambda y\, (!\, V v)\, \lambda z\, (f)\, \lambda g\, (g)\, yz$. On retrouve le premier terme de mise en mémoire trouvé plus haut pour l'ensemble produit $\Lambda(U) \times \Lambda(V)$.

Il est alors naturel d'énoncer la conjecture suivante : toute démonstration intuitionniste de $\vdash^i \forall x \{ D^*[x] \to \neg_0 \neg_0 D[x] \}$ correspond à un terme de mise en mémoire pour l'ensemble $\Lambda(D)$; ou encore

CONJECTURE : *Si $D[x]$ est un type de données, tout terme du λ-calcul, de type $\forall x \{ D^*[x] \to \neg_0 \neg_0 D[x] \}$, est un terme de mise en mémoire pour l'ensemble des termes du λ-calcul « de type D ».*

Remarque (ajoutée à l'envoi de l'article à l'impression) : l'auteur vient de démontrer une forme légèrement modifiée de cette conjecture. Ce résultat fera l'objet d'un autre article [7].

BIBLIOGRAPHIE

1. H. P. BARENDREGT, The lambda calculus, North Holland, 1984.
2. J. Y. GIRARD, Une extension de l'interprétation de Gödel à l'analyse, et son application à l'élimination des coupures, *Proc. 2nd Scand. Logic Symp.*, 1970, North Holland, p. 63-92.
3. J. L. KRIVINE, Lambda-calcul typé, *Masson*, 1990.
4. J. L. KRIVINE et M. PARIGOT, Programming with proofs, *6th Symp.. Comp. Theory* 87, J. Inf. Process. Cybern., 3, 1990, p. 149-157.
5. D. LEIVANT, Reasoning about Functional Programs and Complexity Classes Associated with Type Disciplines, *24th Symp. on Found. of Comp. Sci.*, 1983, p. 460-469.
6. M. PARIGOT, Programming with Proofs: a Second Order Type Theory, *Proc. ESOP'88, Lect. Notes in Comp. Sci., 300*, p. 145-159.
7. J. L. KRIVINE, Opérateurs de mise en mémoire et traduction de Gödel, *Archiv for math. Logic* (à paraître).