REVUE FRANÇAISE D'AUTOMATIQUE INFORMATIQUE
RECHERCHE OPÉRATIONNELLE. INFORMATIQUE THÉORIQUE

R. CASTANET

Sémantique formelle des opérateurs d'un langage de listes

Revue française d'automatique informatique recherche opérationnelle. Informatique théorique, tome 8, n° R3 (1974), p. 19-36

<http://www.numdam.org/item?id=ITA_1974__8_3_19_0>

R.A.I.R.O.
(8e année, R-3, 1974, p. 19 à 36)

SEMANTIQUE FORMELLE DES OPERATEURS D'UN LANGAGE DE LISTES

par R. CASTANET [1]

Communiqué par E. ENGELER

Résumé. — *Le Lambda K Calculus de Church est un système formel susceptible de servir de langage d'expression de la sémantique d'opérateurs de manipulation de listes et de parcours de ces listes.*

On définira les primitives de ces deux types d'opérateurs : extraction, affectation de parties d'éléments composant une liste et on exprimera la sémantique de ces primitives dans le système formel choisi. A partir de là, les opérateurs de parcours de liste et de manipulation (ajout et retrait d'un élément) pourront être de même exprimés.

INTRODUCTION

Formaliser la sémantique d'un opérateur ou d'un objet quelconque d'un langage consiste à définir un système formel dans lequel il est possible d'exprimer cet opérateur ou cet élément. Mais donner une expression dans un certain système formel consiste finalement à paraphraser dans un système dont la sémantique des opérateurs et des objets élémentaires est connue.

La formalisation de la sémantique de plusieurs langages a été réalisée par le calcul des prédicats (Paduceva [5]), par des sous-ensembles de langages (APL pour APL, Lathwell [7], LISP pour LISP, Mac Carthy [9]). Ces études restent attachées à des langages particuliers, sans qu'il soit possible de généraliser ces méthodes à un langage quelconque.

D'après les travaux de Curry [3], L. Nolin [4] et B. Robinet [6], la logique combinatoire représente un domaine où il est possible de définir des systèmes formels adaptés à l'expression de sémantique.

(1) U.E.R. Mathématique et Informatique, Université de Bordeaux I à Talence.

Ce type de systèmes ont une syntaxe et une sémantique propres très restreintes ; ainsi la paraphrase conduit à de très longues expressions pour la sémantique d'un élément d'un langage quelconque.

Curry introduit un combinateur F (fonctionality) spécialisé pour l'expression de la fonction propre d'un opérateur, c'est-à-dire de sa sémantique. B. Robinet a appliqué cette théorie pour les opérateurs du langage APL agissant sur les tableaux.

Cette formalisation nécessite l'introduction d'un combinateur spécifique, alors que le système formel choisi pour cette étude, le Lambda-K-Calculus de Church, n'est absolument pas lié à l'idée de sémantique. Ce système formel permet de proposer une paraphrase dont il est possible de s'inspirer dans une réalisation de micro-programmes pour une machine qui serait conçue pour le traitement de listes. On trouvera en annexe la définition du Lambda-K-Calculus. Ce système formel permet d'exprimer tous les opérateurs du langage étudié.

Il est évident que tout système formel ne peut pas toujours exprimer les éléments et les opérateurs d'un langage ; dans ce cas, ce système formel serait inadéquat pour exprimer la sémantique du langage total.

Par exemple, les systèmes combinatoires, et, en particulier, le Lambda-K-Calculus ne peuvent exprimer l'opération de rupture de séquence ($G\emptyset\, T\emptyset$). Par contre, ils sont adaptés à l'expression des traitements conditionnels (IF... THEN... ELSE, WHILE... DO), ainsi qu'au traitement de blocs.

Le langage de listes qui sert de base à cette formalisation est un langage spécialisé dans la manipulation de listes composant un arbre syntaxique associé à une phrase d'un langage naturel. Mais la structure proposée est suffisamment générale pour pouvoir s'adapter à toute autre structure de liste comme celle de LISP [9], SLIP [8], L 6 [10]. On s'intéressera à des opérateurs du langage de base identiques à ceux que l'on peut trouver aussi, explicitement ou implicitement dans tous les langages de listes.

On définira la sémantique des opérateurs du langage par étapes :

- formalisation des éléments simples du langage (paragraphes I, II, III);
- formalisation des primitives des opérateurs du langage (paragraphe IV);
- formalisation des opérateurs eux-mêmes, d'une part de parcours des listes, d'autre part de modification des listes (ajout ou retrait d'un élément) (paragraphe V).

I. DEFINITION D'UNE LISTE

Soit $\mathfrak{L}$ une liste définie comme un doublet (S, H) tel que S représente l'état statique de la liste $\mathfrak{L}$ et M l'ensemble des modificateurs de S.

A la liste $\mathfrak{L}$ est associé un caractère essentiellement dynamique par opposition au caractère statique de S.

S est composé d'un ensemble E d'éléments $\{ e_i \}$, $i = [0, n]$ et d'un ensemble A d'accès sur E.

Ces accès sont de la forme successeur s ou prédécesseur p. Ils font intervenir des informations contenues dans les éléments $\{ e_i \}$ de la liste.

L'ensemble M des modificateurs de S est tel que $M = M_A \cup M_R$ où M_A est l'ensemble des modificateurs de type « ajout » et M_R l'ensemble des modificateurs de type « retrait ».

Pour les ensembles M et A on définit les primitives suivantes :

a : opérateur d'adressage
c : opérateur de contenu
ρ : opérateur de positionnement
$\leftarrow$: opérateur d'affectation.

Ces primitives ont pu être précisées à la suite de la démarche formalisatrice dans le langage étudié.

On définit $\mathfrak{C}$ comme l'ensemble des valeurs pouvant être contenues dans un élément $e_i \in E$, et $\mathcal{A}$ comme l'ensemble des valeurs de $\mathfrak{C}$ pouvant être des adresses.

Ainsi :

a : est un opérateur de $\mathcal{A}$ dans E
c : est un opérateur de E dans $\mathfrak{C}$
ρ : est un opérateur de E dans E.

II. SEMANTIQUE D'UN ELEMENT $e_i \in E$

Dans toute formalisation d'un langage, il est indispensable de considérer, d'une part, les primitives des opérateurs de ce langage, et, d'autre part, les éléments primitifs sur lesquels agissent les opérateurs.

Un élément e_i avait été présenté dans un souci de clarté comme le plus petit élément accessible de la liste.

En réalité dans toute liste, ces éléments nommés cellules de liste, doivent être décomposés.

On propose une décomposition très générale qui pourrait être simplifiée pour certains langages de listes comme LISP, c'est-à-dire que certains niveaux de la décomposition ne seront pas présents.

Définition 1. Un élément $e_i \in E$ est un n-uple ayant une structure hiérarchisée telle que :

$e_i = \{ \text{mot} \}^m$: ensemble de m mots concaténés
mot = reliquat $\cup$ partie information
partie information (p.i.) = $\{ \text{caractère} \}^p$
reliquat = $\{ \text{composante élémentaire (c.e.)} \}^r$
caractère = $\{ \text{composante élémentaire (c.e.)} \}^c$.

Les éléments primitifs de $\mathfrak{L}$ sont les éléments « mot », « reliquat », « partie information », « caractère », « composante élémentaire ».

III. EXPRESSION D'UN ÉLÉMENT e_i DANS LE SYSTÈME FORMEL

On représentera le n-uple e_i par la Lambda K expression :

$$\lambda x(xx_1x_2 \ldots x_{n-1}x_n)$$

c'est un vecteur qui représente la structure physique d'un e_i, alors que la structure logique d'un e_i est une arborescence avec les niveaux successifs mot, p.i., caractère, c.e.

IV. EXPRESSION DES PRIMITIVES DES OPERATEURS DU LANGAGE DE LISTE

Définition 2. Soit $M, A, F, B, C \in V$ ensemble des vecteurs et E_x ensemble des éléments du vecteur x tel que $E_M, E_A, E_B, E_C \in E_{e_i}$. M, A, F, B, C sont des i-uples $1 \leqslant i < n$.

Soit $\mathfrak{J}$ l'ensemble des entiers positifs ou nuls : $N, D, E, P \in \mathfrak{J}$ et sont inférieurs ou égaux à n.

On définit des opérateurs *contenu* ou opérateurs d'extraction :

$\mathcal{M}$ opérateur d'extraction d'un mot
$\mathcal{A}$ opérateur d'extraction d'une p.i.
$\mathcal{R}$ opérateur d'extraction d'un reliquat
β opérateur d'extraction d'une c.e.
$\mathcal{C}$ opérateur d'extraction d'un caractère.

$M \leftarrow \mathcal{M}Ne_i$	$e_i \in V, N \in \ , M \in V$	M est un mot
$A \leftarrow \mathcal{A}\mathcal{M}Ne_i$		A est une p.i.
$F \leftarrow \mathcal{R}\mathcal{M}Ne_i$		F est un reliquat
$B \leftarrow \beta D\mathcal{R}\mathcal{M}Ne_i$		B est une c.e.
$C \leftarrow \mathcal{C}E\mathcal{A}\mathcal{M}Ne_i$		C est un caractère
$B \leftarrow \beta P\mathcal{C}E\mathcal{A}\mathcal{M}Ne_i$		B est une c.e.

Proposition 1. La sémantique de l'opérateur *contenu* sur les composantes d'un élément d'une liste peut être représenté par un combinateur $\mathfrak{S}$ 2-paramétré exprimable dans le système formel du Lambda K-Calculus par :

$$\mathfrak{S}_{x,y} = \lambda u_1 \lambda u_2 (\lambda v (v \mathfrak{X}(x(Pu_1)) W((Ku_2)\mathfrak{E})(Py) W((KI(J)K))$$

x, y sont les paramètres du combinateur $\mathfrak{S}$.

Les combinateurs $\mathfrak{X}$, P, W, K, $\mathfrak{E}$, I, J sont λ-K exprimables

$\mathfrak{X} = \lambda m_1 \lambda m_2 \lambda a (m_1 (m_2 a))$ multiplication dans l'ensemble des entiers $\mathfrak{Z}$
P = prédécesseur dans $\mathfrak{Z}$
$W = \lambda r_1 r_2 (r_1 r_2 r_2)$
$K = \lambda xy(x)$
$\mathfrak{E} = \lambda xy(y)$
$I = \lambda x(x)$ identité
$J = \lambda x(\lambda yxy)$.

On a les équivalences suivantes d'après la définition II :

$\mathcal{M} \equiv \mathfrak{S}_{n,n}$
$\mathcal{C} \equiv \mathfrak{S}_{a,a}$
$\beta \equiv \mathfrak{S}_{1,1}$
$\mathcal{R} \equiv \mathfrak{S}_{0,P_r}$ (prédécesseur de r)
$\mathcal{A} \equiv \mathfrak{S}_{r,rP_n}$ ($n - r$: r fois le prédécesseur de n).

REMARQUE : n, a, r sont des entiers définissant la structure d'un e_i.

n : nombre de c.e. d'un mot
a : nombre de c.e. d'un caractère
r : nombre de c.e. d'un reliquat.

Ces entiers sont suffisants pour définir la structure d'un e_i. Il est possible de trouver une autre combinaison de 3 entiers pour définir un e_i (3 entiers : nombre de niveaux de la structure hiérarchisée d'un e_i moins une unité).

Démonstration de la proposition 1 pour $\mathfrak{S}_{n,n}$

Extraction du contenu du Nième mot de la cellule e_i

$$\mathfrak{S}_{n,n}Ne_i = \lambda u_1 \lambda u_2 (\lambda v (v \mathfrak{X}(n(Pu_1)) W((Ku_2)\mathfrak{E})(Pn) W((KI(J)K))Ne_i$$
$$= \lambda v (v \mathfrak{X}(n(PN)) W(Ke_i)\mathfrak{E})(Pn) W((KI(J)K)).$$

On décompose cette expression afin d'en rendre plus clair le calcul :

$$A = \mathfrak{X}(n(PN)) W(Ke_i)\mathfrak{E} \quad , \quad B = (Pn) W((KI(J)K)).$$

Dans ce système formel les entiers sont représentés par l'expression suivante :

$$\forall n \geqslant 0 \equiv \lambda n_1 n_2 \{ (n_1 \}^n n_2 \{) \}^n \quad \text{si} \quad \{ (n_1 \}^n$$

représente la concaténation de n fois « $(n_1$ » et $\{\ )\}^n$ la concaténation de n parenthèses fermantes.

$$A = \lambda n_1 n_2 \{\ (n_1\}^{n\times(N-1)} n_2 \{\ )\}^{n\times(N-1)} W((Ke_i)\mathfrak{E})$$
$$= \{\ (W\}^{n\ \ (N-1)}(Ke_i)\mathfrak{E}\{\ )\}^{n\times(N-1)}$$

L'expression de W étant $\lambda r_1 r_2 (r_1 r_2 r_2)$ on a : $W(Ke_i)\mathfrak{E} = Ke_i\mathfrak{E}\mathfrak{E}$.

Ainsi $A = Ke_i\{\ \mathfrak{E}\}^{n\times(N-1)+1} = \lambda xy(x)e_i\{\ \mathfrak{E}\}^{n\times(N-1)+1} = e_i\{\ \mathfrak{E}\}^{n\times(N-1)}$

L'expression de e_i étant $\lambda x(xx_1 \ldots x_p)$ on a $e_i\mathfrak{E} = \mathfrak{E}x_1 \ldots x_p = \lambda y(yx_2 \ldots x_p)$.

Ainsi on obtient l'élimination de $n \times (N-1)$ premiers éléments de e_i par application de $\mathfrak{E}n \times (N-1)$ fois.

Ainsi :
$$A = \lambda x(xx_{n\times(N-1)+1} \ldots x_p)$$
$$B = (Pn)W((KI)J)K$$

Par une suite de transformations analogues :

$$B = KI\{\ J\}^n K = \{\ J\}^{n-1}K$$

B appliqué sur A nous permet de sélectionner les n premiers éléments de A :

$$AB = \lambda x(xx_{n\times(N-1)+1} \ldots x_p)\{\ J\}^{n-1}K = Jx_{n\times(N-1)+1} \ldots x_p\{\ J\}^{n-2}K$$
$$= x_{n\times(N+1)+1} \ldots \lambda y(x_{n\times N-1} yx_{n+N} \ldots x_p)K = x_{n\times(N-1)+1} \ldots x_{n\times N}$$

Ainsi $\mathfrak{S}_{n,n}Ne_i = \lambda v(vx_{n\times(N-1)+1} \ldots x_{n\times N})$.

Définition 3. Soient $\mathfrak{F}_m$, $\mathfrak{F}_c$, $\mathfrak{F}_B$, $\mathfrak{F}_i$ *les opérateurs de* positionnement sur respectivement : un mot, un caractère, une composante élémentaire et une partie information, tels que :

$$\mathfrak{F}_m Ne_i \to V_1^* \quad , \quad \mathfrak{F}_i V_1^* \to V_2^* \quad , \quad \mathfrak{F}_c CV_2^* \to V_3^* \quad , \quad \mathfrak{F}_B BV_3^* \to V_4^*.$$

On a : N, C, $B \in \mathfrak{I}$ ensemble des entiers naturels. $V_i^* \notin V$ ensemble des vecteurs si N, C, $B \neq 0$; on a $V_i^* \in V^*$ ensemble de vecteurs pointés (un vecteur pointé possède des composantes comme un vecteur, mais l'une d'elles est repérée ou pointée. Exemple : $x_1x_2 \ldots x_k\lambda x(xx_{k+1} \ldots x_n)$ est un vecteur pointé avec la composante x_{k+1} pontée).

Proposition 3. La sémantique de l'opérateur de positionnement sur un vecteur ou un vecteur pointé peut être représentée par un combinateur π 1-paramétré exprimable dans le système formel du Lambda K-calculus :

$$\pi_x = \lambda u_1 \lambda u_2 (\mathfrak{X}(x(Pu_1))W((Ku_2)J)).$$

On a les équivalences :

$$\pi_n \equiv \mathfrak{F}_m \quad , \quad \pi_r \equiv \mathfrak{F}_i \quad , \quad \pi_e \equiv \mathfrak{F}_c \quad , \quad \pi_1 \equiv \mathfrak{F}_B.$$

La démonstration de cette proposition ne pose aucune difficulté.

Définition 4. Soient $\mathcal{R}_m$, $\mathcal{R}_r$, $\mathcal{R}_p$, $\mathcal{R}_c$, $\mathcal{R}_B$ *les opérateurs d'affectation*, respectivement d'un mot, d'un reliquat, d'une p.i., d'un caractère, d'une c.e. Ces opérateurs agissent sur le résultat d'un positionnement. Ils effectuent une élimination (remise à zéro) suivie d'une insertion.

Ces opérateurs envoient l'ensemble des vecteurs pointés V^* et des vecteurs V sur l'ensemble des vecteurs V.

Proposition 3. La sémantique de l'opérateur d'affectation sur un vecteur pointé peut être représenté par un combinateur $\mathcal{A}$ 1-paramétré exprimable dans le système formel du Lambda K calculus.

$$\mathcal{A}_x = \lambda u_1 \lambda u_2 (\lambda v (vxW((Ku_2)\mathcal{E})(Pr)W((Ku_1)(J_1J))(K_1J)I))$$
$$J_1 = \lambda xyz((xy)z) \quad \text{extension de } J \text{ à 3 variables}$$
$$K_1 = \lambda xy(\lambda z(xy)) \quad \text{extension de } K \text{ à 3 variables.}$$

On a les équivalences :

$$\mathcal{R}_m \equiv \mathcal{A}_n \quad , \quad \mathcal{R}_r \equiv \mathcal{A}_r \quad , \quad \mathcal{R}_p \equiv \mathcal{A}_{n-r} \quad , \quad \mathcal{R}_e \equiv \mathcal{A}_e \quad , \quad \mathcal{R}_B \equiv \mathcal{A}_1.$$

Démonstration pour $\mathcal{A}_n$ par exemple :

$$\mathcal{A}_n V_i^* V_j \to V_i \qquad V_i, V_j \in V \text{ vecteurs}$$
$$V_i^* \in V^* \text{ vecteurs pointés.}$$
$$\mathcal{A}_n V_i^* V_j = \lambda v(v(nW((KV_i^*)\mathcal{E})(Pn)W((KV_j^*)(J_1J))(K_1J)I))$$
$$= \lambda v(vV_i^* \{ \mathcal{E} \}^n V_j \{ (J_1J) \}^{n-1}(K_1J)I).$$

V_i^* est de la forme : $x_1x_2 \ldots x_k\lambda y(x_{k+1}y \ldots x_p)$.

$V_i^* \{ \mathcal{E} \}^n$ provoque l'élimination des n premières composantes de V_i^* à partir de l'élément pointé.

V_j est de la forme : $\lambda y(yx_1' \ldots x_n')$

$V_j \{ (J_1J) \}^{n-1}$ permet d'obtenir la chaîne : $(Jx_1')(Jx_2') \ldots \lambda z((Jx_{n-1}')zx_n')$.

On peut ainsi insérer dans le vecteur pointé V_i^* les composantes de V_j; on obtient finalement un vecteur V_i tel que :

$$V_i = \lambda v(vx_1 \ldots x_{n\times(N-1)-1}x_1'x_2' \ldots x_n'x_{n\times(N)+1} \ldots x_p)$$

Définition 5. On peut représenter *l'opérateur d'adressage* D dans une mémoire à bande par le combinateur d'extraction $\mathcal{G}_{l,l}$ dans lequel l est le nombre de composantes de cette mémoire.

$\mathcal{G}_{l,l}$ envoie l'ensemble $\mathfrak{I} \times \mathcal{M}$ sur l'ensemble E. ($\mathfrak{I}$ ensemble des entiers, $\mathcal{M}$ ensemble des mémoires, E ensemble des cellules.)

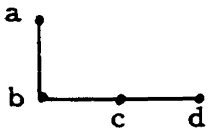
V. EXPRESSION DE LA SEMANTIQUE DES OPERATEURS D'UN LANGAGE DE LISTE

V.1. Opérateurs $\in A$ (accès sur E)

Pour définir précisément ces opérateurs, il est indispensable de se donner un type de liste et sa représentation.

On s'intéressera à une liste symétrique du type de celles du LISP [9].

On se limitera à l'étude de quelques opérateurs d'accès. Soit la structure de liste suivante :



telle que b, c, d sont les trois successeurs directs de a.

Sur cette structure on définit, à l'aide des primitives a, ρ, c, $\rightarrow$ (paragraphe I) et du pointeur de niveau π les opérateurs suivants :

DESCG (descente à gauche : $a \rightarrow b$) $\equiv c \cdot \rho^1 \cdot a \cdot c \cdot \pi$

DROITE (aller à droite : $b \rightarrow c, c \rightarrow d$) $\equiv c \cdot \rho^2 \cdot a \cdot c \cdot \pi$

GAUCHE (aller à gauche) : $(d \rightarrow c, c \rightarrow b) \equiv c \cdot \rho^3 \cdot a \cdot c \cdot \pi$

DESCD (descente à droite : $a \rightarrow d$) $\equiv \{ c \cdot \rho^2 \cdot a \cdot c \}^n \cdot c \cdot \rho^1 \cdot a \cdot c \cdot \pi$

HAUT (prédécesseur : $d \rightarrow a, c \rightarrow a, b \rightarrow a$) $\equiv c \cdot \rho^3 \cdot a \cdot \{ c \cdot \rho^3 \cdot a \cdot c \}^{p-1} \cdot \pi$

Ces différents opérateurs indiquent qu'un élément de liste e_i contient au moins quatre sous-éléments avec le second contenant une information pour DESCG (présence de ρ^1), le troisième une information pour DROITE (présence de ρ^2), le quatrième une information pour GAUCHE ou HAUT (présence de ρ^3).

Sémantiquement, on peut définir deux classes d'opérateurs :

mouvement simple : { DESCG, GAUCHE, DROITE }
mouvement multiple : { HAUT, DESCD }.

Expression des opérateurs de mouvement simple :

Proposition 4. La sémantique des opérateurs de mouvement peut s'exprimer sous la forme d'un couple de combinateurs (M^1, M^2) 1-paramétré par u affectant le pointeur de liste π tels que :

$$M^1_{p_1} \equiv \mathcal{R}_p V'_p \mathcal{I}_i 1 \qquad \text{et} \qquad M^2 \equiv \mathcal{R}_B i \mathcal{I}_B 4$$

avec :

$$\left|\begin{array}{l} V'_p \leftarrow \mathcal{A}i\pi V_p \\ V_p \leftarrow \mathcal{M}uD\mathcal{A}\pi \\ i \leftarrow \beta 2\mathcal{R}V_p \end{array}\right.$$

et :

$$\pi \leftarrow (M_u^1, M^2) \cdot \pi$$

On rappelle que les combinateurs élémentaires présents dans le couple (M_u^1, M^2) signifient :

$\mathcal{I}_B$: positionnement sur une composante élémentaire
$\mathcal{I}_i$: positionnement sur une partie information
$\mathcal{R}_B$: affectation d'une composante élémentaire
$\mathcal{R}_p$: affectation d'une partie information
$\mathcal{A}$: contenu d'une partie information
$\mathcal{M}$: contenu d'un mot
β : contenu d'une composante élémentaire
$\mathcal{R}$: contenu d'un reliquat
D : combinateur d'adressage.

Le combinateur M_u^1 permet d'extraire la partie information du $u$ième mot de la cellule pointée par π et d'affecter la p.i. de π avec cette information. Cette opération ne s'effectue que si la $2e$ c.e. du $u$ième reliquat de e_i contient $\lambda x(x0)$; sinon cette c.e. contient le combinateur $\lambda x(xK)$ et π reste inchangé (il n'est pas possible d'effectuer le mouvement simple demandé : par exemple pour DESCG, l'élément pointé par π n'a pas de successeur).

Le combinateur M^2 permet de stocker le résultat de l'opération exprimé sous forme : $\{ \lambda x(x0), \lambda x(xK) \}$ dans la $4e$ c.e. du reliquat de π.

Démonstration de la proposition 4 pour l'opération DESCG

Dans ce cas : $u = 2$.

$V_p \leftarrow \mathcal{M}2D\mathcal{A}\pi$, V_p : 2e mot de la cellule pointée par π
$i \leftarrow \beta 2\mathcal{R}V_p$, i : 2e c.e. du reliquat de V_p

1) Si $i \equiv \lambda x(x0)$ on a : $V'_p \leftarrow \mathcal{A}i\pi V_p$
ou $V'_p \leftarrow \mathcal{A}0\pi V_p$ ou : $V'_p \leftarrow \mathcal{A}V_p$, V'_p : p.i. de V_p

$\mathcal{I}_i 1\pi$ permet le positionnement sur la p.i. de π et $\mathcal{R}_p V'_p \mathcal{I}_i 1\pi$ affecte p.i. de π avec V'_p.

2) Si $i \equiv \lambda x(xK)$ on a : $V'_p \leftarrow \mathcal{A}K\pi V_p$ c'est-à-dire $V'_p \leftarrow \mathcal{A}\pi$. Cela revient à affecter la p.i. de π avec p.i. de π, donc à ne pas modifier π.

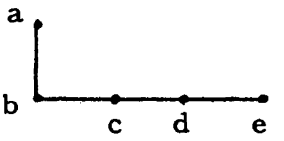
Le combinateur M^2 du couple est tel que : $\pi \leftarrow M^2\pi$, $\pi \leftarrow \mathcal{R}\, i\mathcal{I}\, 4\pi$.

$\left\{\begin{array}{l} \mathcal{I}_B 4\pi \text{ positionnement sur la } 4^e \text{ c.e. du reliquat de } \pi \\ \mathcal{R}_B i\mathcal{I}_B 4\pi \text{ remplacement par } i \text{ de cette c.e.} \end{array}\right.$

Expression des opérateurs de mouvement multiple

Pour ce type d'opérations, il faut introduire une récursivité afin de répéter autant de fois qu'il est nécessaire l'opération de mouvement simple associée.

Par exemple, dans la structure suivante :



l'opération HAUT à partir de *d* est équivalente à deux opérations GAUCHE, suivies d'une montée directe.

De même, pour cet exemple, l'opération DESCD (descente à droite) équivaut à DESCG suivie de trois opérations DROITE.

La répétition des mouvements est obtenue par la présence d'un combinateur dans la cellule pointée à chaque mouvement élémentaire suivant le même procédé que pour l'adressage indirect dans la technologie des ordinateurs (cet adressage se présente généralement sous la forme d'un astérisque. Dans l'exemple pris, on aura $\in \cdot b$ (contenu de b) $= * a$, $\in \cdot c = * b$, $\in \cdot d = * c$, $\in \cdot e = * d$. Ainsi, à partir de e, on accèdera à l'élément a par 3 indirections).

Expression de l'opérateur HAUT

Soit p l'opérateur prédécesseur et S^i l'opérateur donnant le $i$ème successeur. On suppose que l'on a c.e.[3] (reliquat (mot 4)) e_i égale à :

$$\begin{array}{lll} \lambda x(x0) & \text{si} & p(e_i) = \varnothing \\ \lambda x(xK) & \text{si} & e_i^* \ (e_i \text{ courant}) = S^1(e_i) \\ \lambda x(xR) & \text{si} & e_i^* = S^i(e_i) \qquad i > 1 \end{array}$$

avec : $R \equiv \beta 4 \mathcal{R} r \mathcal{M} 4 D \mathcal{A}$ et $r \equiv \lambda r_1(r_1 I r_1)$.

D'autre part, comme pour les autres opérateurs, on supposera que c.e[5]. (reliquat (mot 2)) e_i^* est égale à :

$$\begin{array}{lll} \lambda x(x0) & \text{si} & p(e_i^*) \neq \varnothing \\ \lambda x(xK) & \text{si} & p(e_i) = \varnothing \ \text{(tête de liste).} \end{array}$$

Proposition 5. La sémantique de l'opérateur HAUT s'exprime par un couple de combinateur (H^1, H^2) sur π tels que :

$$H_1 : \pi \leftarrow \mathcal{R}_p \mathcal{A}_i I V_4 \pi \mathcal{T}_i 1 \pi \qquad \begin{array}{l} V_4 \leftarrow \mathcal{M} 4 D \mathcal{A} \pi \\ i \leftarrow \beta 5 \mathcal{R} V_4 \end{array}$$

$$H_2 : \pi \leftarrow \mathcal{R}_B \beta 5 \mathcal{R} V_4 \mathcal{T}_B 4 \pi$$

Démonstration :

— si $i = \lambda x(x0)$ $\mathcal{R}_p\mathcal{A}_i IV_4\pi\mathcal{I}_i 1\pi \Rightarrow \mathcal{R}_p\mathcal{A}0V_4\pi\mathcal{I}_r 1\pi$

$\Rightarrow \mathcal{R}_p\mathcal{A}\pi\mathcal{I}_r 1\pi$: π inchangé.

— si $i = \lambda x(xK)$ $\mathcal{R}_p\mathcal{A}\lambda x(xK)IV_4\pi\mathcal{I}_i 1\pi$

$\Rightarrow \mathcal{R}_p\mathcal{A}KV_4\pi\mathcal{I}_r 1\pi$

$\Rightarrow \mathcal{R}_p\mathcal{A}V_4\mathcal{I}_r 1\pi$: p.i. $(V) \rightarrow$ p.i. (π)

— si $i = \lambda x(xR)$ $\mathcal{R}_p\mathcal{A}\lambda x(xR)IV_4\pi\mathcal{I}_i 1\pi$

$\Rightarrow \mathcal{R}_p\mathcal{A}RV_4\pi\mathcal{I}_i I\pi$

$\Rightarrow \mathcal{R}_p\mathcal{A}\beta 5\mathcal{R}r\mathcal{M}4D\mathcal{A}V_4\pi\mathcal{I}_i 1\pi$

$\Rightarrow \mathcal{R}_p\mathcal{A}\beta 5\mathcal{R}rV'_4\pi\mathcal{I}_i I\pi$

$\Rightarrow \mathcal{R}_p\mathcal{A}\underbrace{\beta 5\mathcal{R}V'_4}IV'_4\pi\mathcal{I}_i I\pi$

La partie en accolade permet d'atteindre une nouvelle composante i'

$$\Rightarrow \mathcal{R}_p\mathcal{A}i'IV'_4\pi\mathcal{I}_i 1\pi$$

Cette forme est équivalente à la forme primitive de l'opérateur HAUT i et V_4 ont été remplacés par i' et V'_4.

Pour continuer le processus itératif, il faut examiner la valeur de i'. La dernière itération se fera lorsqu'on arrivera en début de niveau, c'est-à-dire lorsque $i^{(x)} = \lambda x(xK)$.

Une condition nécessaire pour que l'opérateur HAUT soit applicable quel que soit e_i est que R ne contienne aucune variable liée telle que π ou une variable dépendant d'un e_i. La forme présentée satisfait cette condition.

Expression de l'opérateur DESCD (descente à droite)

Soit s l'opérateur donnant les successeurs de e_i.

Si card $(s(e_i)) = n$ DESCD $\equiv$ DESCG { DROITE }^{n}

Soit $S^i(e_i)$ le $i$ème successeur de e_i, on suppose que l'on a pour $S^1(e_i)$ et pour $S^i(e_i)$ avec $i \neq 1$, c.e[3]. (reliquat (mot[4])) égal à :

— $\lambda x(x0)$ si $S^1(e_i) = \varnothing$ (vide) ou DROITE $(e_i^*) = \varnothing$ (e_i^* étant l'élément e_i courant dans l'itération).

— $\lambda x(xR')$ si $S^1(e_i) \neq \varnothing$ et DROITE $(e_i^*) \neq \varnothing$

$R' \equiv \beta 3\mathcal{R}r\mathcal{M}3D\mathcal{A}K$ avec $K' = \lambda xy(x)$.

D'autre part, on a :

c.e[4]. (reliquat (mot[2]))e_i égale à :

— $\lambda x(x0)$ si $s(e_i) \neq \varnothing$

— $\lambda x(xK)$ sinon.

Ainsi, au premier pas de l'itération, on peut connaître le résultat de l'opérateur, si $s(e_i) \neq \varnothing$, résultat $\Leftrightarrow \lambda x(x0)$, sinon résultat $\Rightarrow \lambda x(xK)$.

Proposition 6. La sémantique de l'opérateur DESCG s'exprime par un couple de combinateurs sur $\pi(D_1, D_2)$ tels que :

$$D_1 : \pi \leftarrow \mathcal{R}_p\mathcal{A}iIV_2\pi\mathcal{T}_i1\pi \qquad \text{avec} \begin{cases} V_2 \leftarrow \mathcal{M}2D\mathcal{A}\pi \\ i \leftarrow \beta 2\mathcal{R}V_2 \end{cases}$$

$$D_2 : \pi \leftarrow \mathcal{R}\ \beta 4\mathcal{R}V_4\mathcal{T}\ 4\pi.$$

Démonstration :

Cas n° 1 : $s(e_i) = 0 \Rightarrow i = \lambda x(x0)$

$$\mathcal{R}_p\mathcal{A}\lambda x(x0)IV_2\pi\mathcal{T}_i1\pi$$
$$\Rightarrow \mathcal{R}_p\mathcal{A}0V_2\pi\mathcal{T}_i1\pi$$
$$\Rightarrow \mathcal{R}_p\mathcal{A}\pi\mathcal{T}_i1\pi : \text{p.i. } (\pi) \rightarrow \text{p.i. } (\pi) \quad , \quad \pi \text{ inchangé.}$$

Cas n° 2 : $|s(e_i)| = 1 \quad i = \lambda x(xR')$

$$\mathcal{R}_p\mathcal{A}R'V_2\pi\mathcal{T}_i1\pi \Rightarrow \mathcal{R}_p\mathcal{A}\beta 3\mathcal{R}r\mathcal{M}3D\mathcal{A}K'V_2\pi\mathcal{T}_i1\pi$$
$$\Rightarrow \mathcal{R}_p\mathcal{A}\beta 3\mathcal{R}r\underbrace{\mathcal{M}3D\mathcal{A}V_2}_{V'_3}V_2\mathcal{T}_i1\pi$$
$$\Rightarrow \mathcal{R}_p\mathcal{A}\underbrace{\beta 3\mathcal{R}V'_3}_{i'}IV'_3V_2\mathcal{T}_i1\pi$$

$$\text{Card } (s(e_i)) = 1 \qquad i' = \lambda x(x0)$$
$$\mathcal{R}_p\mathcal{A}R'V_2\pi\mathcal{T}_i1\pi \Rightarrow \mathcal{R}_p\mathcal{A}V'_3V_2\mathcal{T}_i1\pi \Rightarrow \mathcal{R}_p\mathcal{A}V_2\mathcal{T}_i1\pi$$

Ainsi $s(e_i) \rightarrow$ p.i. (π).

Cas n° 3 : Card $(s(e_i)) > 1$

$$i' = i = \lambda x(xR')$$

On reprend le calcul au niveau du cas 2 :

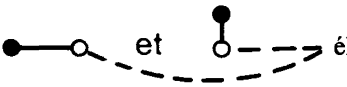
$$\mathcal{R}_p\mathcal{A}R'V_2\pi\mathcal{T}_i1\pi \Rightarrow \mathcal{R}_p\mathcal{A}R'V'_3V_2\mathcal{T}_i1\pi$$
$$\Rightarrow \mathcal{R}_p\mathcal{A}\beta 3\mathcal{R}r\underbrace{\mathcal{M}3D\mathcal{A}V'_3}_{V''_3}V'_3\mathcal{T}_i1\pi$$
$$\Rightarrow \mathcal{R}_p\mathcal{A}\underbrace{\beta 3\mathcal{R}V''_3}_{i''}IV''_3V'_3\mathcal{T}_i1\pi$$

Si $i'' \equiv \lambda x(x0)$ on exécute le cas 2, sinon on exécute de nouveau le cas 3.

VI.2. Opérateurs $\in M$ (modificateurs sur E.)

On se limitera à des opérateurs de retrait et d'ajout agissant sur des éléments e_i extrémités de liste.

Opérateur de retrait

On trouvera deux cas de figures  éléments

à retirer. Dans un but de simplification, on suppose que la réserve de cellules libres est infinie ou suffisante pour une suite d'opérations quelconques pour éviter le « repêchage » des cellules (garbage collector).

Proposition 7. La sémantique de l'opérateur de retrait RET dans une liste bidirectionnelle peut s'exprimer par un doublet de combinateurs (RET 1, RET 2) avec RET 1 sur π et RET 2 sur $D\mathcal{A}\pi$ tels que :

$$\text{RET 1} : \pi \leftarrow \mathcal{R}_p V_4' \mathcal{T}_i \pi \quad \text{avec} \begin{cases} V_4' \leftarrow \mathcal{A} i I \pi V_4 \\ V_4 \leftarrow \mathcal{M} 4 D \mathcal{A} \pi \\ i \leftarrow \beta 5 \mathcal{R} V_4 \end{cases}$$

$$\text{RET 2} : D\mathcal{A}\pi \leftarrow \mathcal{R}_B K_p \mathcal{T}_B 2 \mathcal{T}_m i' D\mathcal{A}\pi \quad \text{avec} \begin{cases} i' \leftarrow \beta 2 \mathcal{R} V_4 I 23 \\ K_p \equiv \lambda x(xK) \end{cases}$$

Démonstration :

$\pi \leftarrow$ RET 1. π V_4' est la p.i. $(\text{mot}^4)\, e_i$ donc l'adresse de la cellule $e_{i.1}$
$D\mathcal{A}\pi \leftarrow$ RET 2. $D\mathcal{A}\pi$ RET 2 affecte c.e^2. $(\text{reliquat } (\text{mot}^{i'}))\, e_i$
à $\lambda x(x0)$ si GAUCHE $(e_i) \neq \varnothing$
à $\lambda x(xK)$ sinon.

$\beta 2 \mathcal{R} V_4$ sélectionne c.e^2. (reliquat (V_4)).

1) $i' = \lambda x(xK) I 23$
$= K 23$
$= 2$

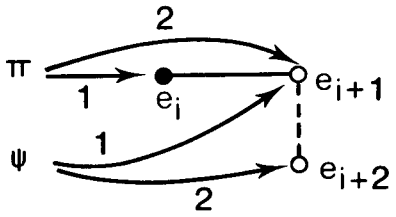
2) $i' = \lambda x(x0) I 23$
$= 0 23$
$= 3$

Opérateur d'ajout

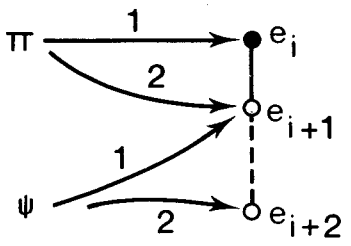
A ce niveau, il est indispensable d'introduire l'ensemble « réserve » composé de cellules disponibles liées. Soit ψ le pointeur sur la tête de la réserve.

On observera les deux cas possibles : (les liaisons 1 donnant la situation avant l'opération d'ajout et les liaisons 2 après cette opération).

Cas n° 1 :



Cas n° 2 :



Proposition 8. La sémantique de l'opérateur d'ajout s'exprime par un quadruplet de combinateurs 1-paramétré (u) $aj = (aj_1, aj_2, aj_3, aj_5)$. Ce quadruplet est ordonné :

$$aj_1 > aj_2 > aj_3 > aj_4.$$

Les quatre éléments intervenant dans cette opération d'ajout sont :

$$e_i(D\mathcal{A}\pi) \quad , \quad e_{i+1}(D\mathcal{A}\psi) \quad , \quad \pi \quad , \quad \psi$$

sur lesquels agissent respectivement : aj_1, aj_2, aj_3, aj_4.

$$\begin{aligned} aj_1 &\equiv \mathcal{R}_B O_p \mathcal{I}_B 2\mathcal{I}_m 2\mathcal{R}_p \mathcal{A}\psi\mathcal{I}_i\mathcal{I}_m u \\ aj_2 &\equiv \mathcal{R}_B K_p \mathcal{I}_B 2\mathcal{I}_m 2\mathcal{R}_B O_p \mathcal{I}_B i'\mathcal{I}_m 4\mathcal{R}_p \mathcal{A}\pi\mathcal{I}_i\mathcal{I}_m 4 \\ aj_3 &\equiv \mathcal{R}_p \mathcal{A}\psi\mathcal{I}_i \\ aj_4 &\equiv \mathcal{R}_p \mathcal{A}\mathcal{M}2D\mathcal{A}\psi\mathcal{I}_i \end{aligned}$$

avec :

$$\begin{aligned} O_p &= \lambda x(x0) \\ i' &\leftarrow \beta 2\mathcal{R}V_3 I25 \\ V_3 &\leftarrow \mathcal{M}3D\mathcal{A}\pi \end{aligned}$$

Seul le combinateur aj_1 est 1-paramétré.

Pour le cas n° 1, $u = 2$, pour le cas n° 2, $u = 3$.

Démonstration :

aj_1 : 1. Mise en place dans e_i de l'adresse du successeur e_{i+1}.

2. Indication que $s(e_i) \neq \varnothing$ ou DROITE $(e_i) \neq \varnothing$.

$$\underset{1.2.}{\mathcal{R}_p \mathcal{A}\psi}\ \underset{1.1.}{\mathfrak{I}_i \mathfrak{I}_m 2D\mathcal{A}\pi}$$

1.1. Positionnement sur p.i^2. de l'élément dont l'adresse est dans π, c'est-à-dire e_i.

1.2. Sélection de p.i. (ψ)

1.1. et 1.2. : Affectation de p.i^2. (e_i) par p.i. (ψ).

2 :
$$\underset{2.2.}{\mathcal{R}_B O_p}\ \underset{2.1.}{\mathfrak{I}_B 2\mathfrak{I}_m 3D\mathcal{A}\pi}$$

2.1. Positionnement sur la c.e^2. (reliquat (mot^3)) e_i

2.2. Affectation de cette c.e. par $O_p = \lambda x(x0)$.

aj_2 : 1. Mise en place dans e_{i+1} de l'adresse de $p(e_i)$ ou de GAUCHE (e_i).

2. Indication que $s(e_i) \neq \varnothing$ ou GAUCHE $(e_i) \neq \varnothing$.

3. Indication que $s(e_{i+1}) = \varnothing$.

Expression de l'action 1 : $\mathcal{R}_p \mathcal{A}\pi \mathfrak{I}_i \mathfrak{I}_m 4D\mathcal{A}\psi$

Expression de l'action 2 : $\mathcal{R}_B O_p \mathfrak{I}_B i' \mathfrak{I}_m 4D\mathcal{A}\psi$

$\mathfrak{I}_m 4D\mathcal{A}\psi$ positionne sur mot^4 (e_{i+1})

$i' \leftarrow \beta 2\mathcal{R} V_3 I25$

$V_3 \leftarrow \mathcal{M}3D\mathcal{A}\pi$: V_3 mot^3 (e_i).

$\beta 2\mathcal{R} V_3$: extraction de c.e^2. (reliquat (V_3)).

Deux cas sont possibles :

- Composante : $\lambda x(x0)$ positionné par aj_1
  cela implique : $aj_1 > aj_2$.
- Composante : $\lambda x(xK)$ (DROITE $(e_i) = \varnothing$)
  cas 1 : $\lambda x(xK) I25 \Rightarrow 2$
  cas 2 : $\lambda x(x0) I25 \Rightarrow 5$.

Cette sélection permet de se positionner sur la c.e. correspondante.

aj_3 : p.i. $(\pi) \leftarrow$ p.i. (ψ)
$\pi \leftarrow \mathcal{R}_p \mathcal{A} \psi \mathcal{S}_i \pi$

aj_4 : p.i. $(\psi) \leftarrow$ adresse de $s(e_{i+1})$
$\psi \leftarrow \mathcal{R}_p \mathcal{A} \mathcal{M} 2 D \mathcal{A} \psi \mathcal{S}_i \psi$
2 1

1 : positionnement sur p.i. de ψ
2 : extraction dans e_{i+1} de l'adresse de e_{i+2}.

Dans le même esprit, il est possible d'exprimer d'autres opérateurs d'ajout et de retrait. plus complexes (cas d'éléments qui ne sont pas des queues de listes).

BIBLIOGRAPHIE

[1] R. CASTANET, *Une formalisation de la sémantique des opérateurs d'extraction dans une arborescence*, C. R. Acad. Sc. Paris, série A, *275* (1972), 135-137.

[1A] R. CASTANET, *Sur la sémantique des opérateurs d'insertion dans une arborescence*, C. R. Acad. Sc. Paris, série A, *275* (1972), 209-212.

[2] A. CHURCH, *The Calculi of lambda conversion*, Princeton University Press (Princeton), 1941.

[3] H. B. CURRY et R. FEYS, *Combinatory logic*, 1, North Holland (Amsterdam), 1958.

[4] L. NOLIN, *Logique combinatoire et algorithmes*, C. R. Acad. Sc. Paris, série A, *272* (1972), 1435-1438 et 1485-1488.

[5] E. V. PADUCEVA, *Réduction de groupes de coordination contenant des éléments se répétant, in* La sémantique en U.R.S.S., Dunod, Paris (1971).

[6] B. ROBINET, *Sémantique des tableaux : Application au langage APL*. Thèse 3e cycle, Université Paris, 6 (1972).

[7] R. H. LETHWELL and J. E. MEZEI, *A formal description of APL*, Actes du congrès APL, IRIA, Rocquencourt (1971).

[8] M. WEIZENBAUM, *Symetric List Processor*, Comm. ACM, vol. 6, n° 9 (1963), 524-544.

[9] J. Mac CARTHY, *LISP 1.5 Programmer's Manual*, MIT Press, Cambridge Mass. (1962).

[10] K. C. KNOWLTON, *A programmer's description of L6*. Comm. ACM, vol. 7 (1965), 623-625.

[11] C. BÖHM, *The CUCH as a formal and description language, in* Formal language, description language, STEEL (1964).

[12] P. WEGNER, *Programming, information structures and machine organisation*, McGraw Hill, New York (1968).

[13] M. VENTURINI-ZILLI, *Lambda K formulae for vectors operators*, Int. Comp. Center Bull., *4* (1965), 157-174.

ANNEXE 1

PRESENTATION DU LAMBDA-*K*-CALCULUS

1. Morphologie

– Les éléments de base sont les variables. Soit V le jeu de toutes les variables. Les autres symboles sont : λ, (,). Ces symboles primitifs permettent de former d'autres symboles par concaténation.

– Les instructions ou mots bien formés du système formel seront appelés lambda expressions.

Les règles qui permettent de définir une λ expression sont les suivantes :

– Une variable est une λ expression.

– Si M est une λ expression et x est une variable, λxM est une λ expression.

– Si A et B sont des λ expressions (AB) est une λ expression où A est la partie opérateur et B la partie opérande.

On définit trois classes de variables :

– Classe des variables de liaison qui suivent le caractère λ.

– Classe des variables liées, x est lié dans X si et seulement si x est l'argument lié dans X ou une composante de X.

– Classe des variables libres, x est libre dans X si x n'apparaît pas comme liée dans X.

REMARQUE : La morphologie du λ-*K*-Calculus accepte comme bien formée des expressions de la forme : λxM où x n'apparaît pas dans M.

2. Théorie propre du Lambda-*K*-Calculus

On se fixe deux règles qui permettent de déduire d'axiomes des théorèmes qui seront des λ expressions bien formées.

— *Règle de substitution ou de réduction*

Les expressions du type (AB) spécifient l'application de l'opérateur A sur l'opérande B.

Si A est de la forme λxM cette application est la substitution de toutes les occurrences de la variable libre x dans M par l'opérande B.

— *Règle de réappélation*

Si M est une λ expression et si x est une variable liée dans M, on peut lui substituer la variable y si M ne contient pas d'occurrences libres de x et si y n'apparaît pas dans M.