

B. ROBINET

Un modèle fonctionnel des structures de contrôle

RAIRO. Informatique théorique, tome 11, n° 3 (1977), p. 213-236

<http://www.numdam.org/item?id=ITA_1977__11_3_213_0>

© AFCET, 1977, tous droits réservés.

L'accès aux archives de la revue « RAIRO. Informatique théorique » implique l'accord avec les conditions générales d'utilisation (<http://www.numdam.org/conditions>). Toute utilisation commerciale ou impression systématique est constitutive d'une infraction pénale. Toute copie ou impression de ce fichier doit contenir la présente mention de copyright.

NUMDAM

Article numérisé dans le cadre du programme
Numérisation de documents anciens mathématiques

<http://www.numdam.org/>

UN MODÈLE FONCTIONNEL DES STRUCTURES DE CONTRÔLE (*)

par B. ROBINET (1)

Communiqué par J.-F. Perrot

Résumé. — *Nous prouvons la validité des transformations syntaxiques de programme en accord avec la sémantique des structures de contrôle d'EXEL induite par un modèle en λ -calcul. Pour définir ce modèle, nous donnons d'abord une sémantique en λ -calcul des diagrammes de Böhm et Jacopini utilisant une formalisation de la notion de pile, puis nous la composons avec la traduction d'EXEL en termes de diagrammes présentée dans un précédent article.*

INTRODUCTION

L'idée d'une théorie du calcul (computation) est fort ancienne. Elle revêt aujourd'hui une importance nouvelle, en raison des difficultés très réelles que rencontre l'analyse et la mise au point de programmes complexes.

Anticipant sur ces vicissitudes, quelques précurseurs ont très tôt cherché à adapter les diverses théories de la calculabilité, édifiées par les logiciens dans les années 30, aux contingences du calcul sur ordinateur. Parmi les différents formalismes connus, c'est celui du lambda-calcul de Church (ou la logique combinatoire de Curry, plus compacte) qui s'est montré le mieux adapté : il se trouve à la source des principales approches de la sémantique des langages de programmation, aussi bien dénotationnelles (Landin, Strachey, Scott) qu'opérationnelles (McCarthy). Dans ces systèmes formels, que ce soit le lambda-calcul ou la logique combinatoire, une fonction est représentée par une expression qui, concaténée à une liste d'arguments, est soumise à un algorithme universel de réduction qui la transforme en la valeur de la fonction au point considéré.

Partant de l'a-priori qu'un programme n'est autre que l'expression contrainte du calcul d'une fonction, Landin [15, 16] vit dès, 1965, l'intérêt de traduire tout programme sous forme de lambda-expressions (mots du lambda-calcul); à la même époque, Böhm [5, 6] et McCarthy [18] proposèrent même de

(*) Reçu en mars 1977.

(1) Institut de Programmation, Université Pierre-et-Marie-Curie et L.A. 248 Informatique Théorique et Programmation, Paris.

programmer directement en diverses variantes du lambda-calcul (CUCH, LISP pur). Ils ouvraient la voie à tout un courant de recherche actuel : on sait maintenant traduire tout programme ou toute structure de données en une lambda-expression. Ceci permet de définir dans un univers conceptuel cohérent les notions usuelles de procédures [13, 23, 24, 28], de structures [9, 23] et, surtout, celle de types : dénnotations des ensembles auxquels appartiennent les objets informatiques [26]. Toutes ces constructions sont telles que, disposant d'un programme qui sait interpréter quelques objets primitifs et qui sait faire les constructions du lambda-calcul, on peut manipuler n'importe lequel des objets construits ; en particulier on peut démontrer automatiquement diverses propriétés d'objets informatiques : citons les travaux de Berry [4], Böhm et Dezani [8] sur les stratégies d'implantation de la règle de recopie d'Algol, et ceux de Michel [19] sur les extensions d'opérateurs en APL. Ces démonstrations ne sont bien sûr possibles que si l'on dispose de la définition précise de la signification (sémantique) des langages en question ; ainsi dès la parution officielle du surlangage EXEL [1], Nolin et Ruggiu ont défini, en termes de lambda-calcul, la sémantique mathématique de ce surlangage [21].

Des diverses propriétés d'EXEL, celle de permettre l'expression aisée de transformations de programmes est sans doute une des plus fortes. Ces transformations que l'on qualifie de syntaxiques, car préservant la sémantique, permettent de fonder la nouvelle conception de la programmation proposée par Arzac dans ses « Nouvelles Leçons de Programmation » [2]. Appliquées par Arzac [3] puis étudiées algébriquement par Cousineau [10, 11], elles donnent pour tout programme de nombreux programmes équivalents, en éliminant entièrement certaines notions qualifiées d'invariants comme le nombre de boucles, leurs degrés d'emboîtement, etc... Mais la validité de la plupart de ces transformations est, le plus souvent, simplement admise, c'est-à-dire qu'on les considère comme formant un système d'axiomes. Le but de cet article est de les démontrer en prouvant leur validité dans le cadre de la sémantique d'EXEL que nous proposons ; ces démonstrations permettent d'envisager en toute sécurité, sans risques d'inconsistance génératrice de contradictions, la construction de divers logiciels de transformation automatique de programmes. Pour ce faire nous utilisons la sémantique opérationnelle d'EXEL que nous avons définie dans un article précédent [27] en termes de diagrammes de Böhm et Jacopini [7] ; reprenant cette définition, nous donnons, dans un système formel dérivé du lambda-calcul [26], un modèle fonctionnel des structures de contrôle d'EXEL qui nous permet de fonder les axiomes d'Arzac et d'en prouver la consistance.

Dans une première partie, et faute d'un traité aisément accessible sur le lambda-calcul, nous donnons les rappels nécessaires à la lecture de l'article et nous présentons une version légèrement modifiée du langage CUCH [5, 6], synthèse du lambda-calcul et de la logique combinatoire. La deuxième partie est consacrée à la définition de la traduction σ_B des diagrammes de Böhm et Jacopini en mots du langage CUCH, à la preuve de sa correction (théorème 3) et

à la définition et l'étude de l'équivalence de diagrammes. Reprenant la sémantique opérationnelle σ_E du surlangage EXEL, nous consacrons la troisième partie de cet article à la sémantique mathématique Σ_E d'EXEL : cette dernière est définie comme la composition de σ_B et σ_E , c'est-à-dire que la sémantique d'un programme EXEL est celle du mot CUCH obtenu par application de Σ_E au programme ; enfin nous démontrons la validité des transformations syntaxiques de programme que sont la répétition, la fausse itération, la factorisation, l'inversion, l'absorption (Théorèmes 5 à 9) et nous généralisons quelques propriétés classiques des structures de contrôle, un exemple venant illustrer leur emploi.

I. PROLÉGOMÈNES. LOGICO-COMBINATOIRES

Dans cette première partie nous présentons le langage CUCH [5, 6] augmenté de différentes primitives utiles au dessein poursuivi ici. Afin de ne pas alourdir le propos, la plupart des démonstrations sont rejetées en Annexe. Pour de plus amples développements sur le lambda-calcul et les systèmes applicatifs avec combinateurs le lecteur pourra consulter Curry [12] ou Hindley [14].

1.1. Syntaxe du langage CUCH

1.1.1. Grammaire

Soient :

$V_T = \{ S, K, C, \dots, Y, o, \text{if}, \text{si}, \dots, \Omega, a, b, \dots, \alpha, \beta, \dots, x, y, \dots, \lambda, (,) \}$
le vocabulaire terminal,

$V_A = \{ \mathcal{C}, \mathcal{V}, \mathcal{E}_S, \mathcal{E}_C, \mathcal{L} \}$ le vocabulaire auxiliaire

et \mathcal{L} l'axiome.

Le langage CUCH est engendré par la grammaire suivante :

1. $\mathcal{C} \rightarrow S|K|C| \dots |Y|o| \text{if} | \text{si} | \dots |\Omega|a|b| \dots |\alpha|\beta| \dots$
2. $\mathcal{V} \rightarrow x|y| \dots$
3. $\mathcal{E}_S \rightarrow \mathcal{C}|\mathcal{V}|\lambda\mathcal{V}\mathcal{E}_S|\lambda\mathcal{V}(\mathcal{E}_C)$
4. $\mathcal{E}_C \rightarrow \mathcal{E}_S\mathcal{E}_S|\mathcal{E}_C\mathcal{E}_S|\mathcal{E}_S(\mathcal{E}_C)|\mathcal{E}_C(\mathcal{E}_C)$
5. $\mathcal{L} \rightarrow \mathcal{E}_S|\mathcal{E}_C$

Cette grammaire n'est autre que la formalisation des abus de notation dans l'écriture usuelle du lambda-calcul.

Tout mot du langage CUCH sera nommé λ -expression ; $\lambda x(\lambda y \lambda t K)(S(Kx))$ en est un exemple.

L'identité syntaxique sera notée =.

1.1.2. Variables et contextes

Rappelons sommairement quelques définitions classiques.

Soit E une λ -expression ; toute occurrence de x dans E sera dite liée dans $\lambda x E$ et x sera qualifiée de libre dans E si elle n'est pas liée.

On appelle contexte toute λ -expression dans laquelle une sous-expression a été effacée ; $E[\]$ désignera un contexte de la λ -expression E et $E[A]$ la λ -expression obtenue en insérant le mot A dans $E[\]$.

1.2. Sémantique du langage CUCH

D'une façon classique, nous définissons pour ce langage un certain nombre de règles de calcul appelées souvent règles de réduction ; M, M' et N désignent des mots du langage CUCH.

1.2.1. α -règle

A toute λ -expression $E[\lambda x M]$, on peut substituer $E[\lambda y M']$ si y n'est pas libre dans M et si M' résulte de M en remplaçant toute occurrence libre de x par y .

1.2.2. β -règle

A toute λ -expression $E[\lambda x MN]$, on peut substituer $E[M']$ si M' est le résultat du remplacement de toutes les occurrences libres de x dans M par N et si aucune variable à occurrences libres dans N est liée dans M ; si cette dernière contrainte n'est pas vérifiée, une application préalable de la α -règle à M est effectuée.

1.2.3. η -règle

A toute λ -expression $E[\lambda x (M x)]$, on peut substituer $E[M]$ si x n'est pas libre dans M .

1.2.4. C -règles

Certaines constantes du langage, telles S, K, C, \dots , sont en fait des abréviations pour désigner des λ -expressions remarquables ; ces primitives, que nous nommerons combinateurs par analogie avec les constantes de Curry, représentent en fait des opérateurs dont l'usage sera fréquent.

Nous donnons ci-contre la table des combinateurs que nous utiliserons par la suite, les combinateurs ajoutés au langage CUCH original étant if, si et o . La troisième colonne donne leurs règles de calculs, appelées C -règles, induites par leurs définitions présentes dans la deuxième colonne. Toutes ces constantes ne sont pas indépendantes et l'on montre qu'elles peuvent toutes s'exprimer en fonction des combinateurs S, K et I . On trouvera en Annexe leurs expressions et une illustration sur la façon de les calculer.

noms	λ-expressions	C-règles
S	$\lambda x \lambda y \lambda z (xz(yz))$	$Sabc \equiv ac(bc)$
K	$\lambda x \lambda y x$	$Kab \equiv a$
I	$\lambda x x$	$Ia \equiv a$
B	$\lambda x \lambda y \lambda z (x(yz))$	$Babc \equiv a(bc)$
C	$\lambda x \lambda y \lambda z (xzy)$	$Cabc \equiv acb$
W	$\lambda x \lambda y (xyy)$	$Wab \equiv abb$
Y	$\lambda f(\lambda h(f(hh)))(\lambda h(f(hh)))$	$Ya \equiv a(Ya)$
o	$\lambda x \lambda y \lambda z (y(zx))$	$oabc \equiv b(ac)$
if	$\lambda t \lambda x \lambda y \lambda u (tu(\lambda zy)xu)$	$ifabcd \equiv ad(\lambda zc)bd$
si	$\lambda t \lambda x \lambda u (tu(\lambda y \lambda z z)xu)$	$siabd \equiv ad(\lambda y \lambda z z)bd$
0	$\lambda y \lambda z z$	$0ab \equiv b$
I	$\lambda y \lambda z (yz)$	$Iab \equiv ab$
n	$\lambda y \lambda z \underbrace{y(\dots(yz)\dots)}_{n \text{ fois}}$	$nab \equiv a(\underbrace{n-1}ab), \forall n > 0$

Table des combinateurs

1.2.5. *Forme normale ; interprétation de CUCH*

E, F, G_i désignant des λ-expressions nous poserons $E \rightarrow F$ si F est obtenue à partir de E par application d'une des règles décrites ci-dessus. Nous écrivons $E \Rightarrow F$ s'il existe une séquence finie de λ-expressions G_i

$$E = G_1, \dots, G_i, \dots, G_n = F$$

telle que, $\forall i \in [1 : n - 1], G_i \rightarrow G_{i+1}$; on dit en pareil cas que E est réductible en F .

Si l'on a, $\forall i \in [1 : n - 1], G_i \rightarrow G_{i+1}$ ou $G_{i+1} \rightarrow G_i$ nous écrivons $E \equiv F$ et nous dirons que E (resp. F) est convertible en F (resp. E) ; il est aisé de voir que \equiv est une relation d'équivalence.

Une λ-expression est dite en forme *normale* si les β-η-C-règles ne lui sont applicables et une λ-expression E a une forme normale s'il existe F en forme normale telle que $E \Rightarrow F$.

Toute λ-expression ayant une forme normale peut être considérée comme une description de sa forme normale au même titre que le couple programme-données est une description du résultat d'un calcul.

Nous pouvons donc définir la sémantique du langage CUCH comme l'assignation d'une forme normale à toute λ-expression donnée.

Formellement, l'interprétation de CUCH dans lui-même est définie par la fonction partielle VALEUR telle que :

$$VALEUR(E) = \begin{cases} F, & \text{s'il existe } F \text{ en forme normale telle que } E \Rightarrow F \\ \text{indéfini} & \text{sinon} \end{cases}$$

Cette façon de définir la sémantique de CUCH peut être qualifiée de « syntaxique » : nous ne sommes pas préoccupés par des problèmes d'interprétation externes au langage CUCH et le processus de calcul permettant

d'obtenir VALEUR (E), pour tout mot E de CUCH, est mécanique car fondé sur la seule opération de substitution.

Par ailleurs le fait que VALEUR ait été définie comme fonction partielle repose sur les théorèmes suivants :

THÉORÈME DE CHURCH-ROSSER ET COROLLAIRES :

Si $E \Rightarrow F$ et $E \Rightarrow C$ alors il existe H unique telle que $F \Rightarrow H$ et $C \Rightarrow H$.

Si $E \equiv F$ alors il existe G telle que $E \Rightarrow G$ et $F \Rightarrow G$.

Si $E \equiv F$ alors ou bien E et F ont la même forme normale ou elles n'en ont pas.

Une démonstration de ces théorèmes figure dans [14] ; H. B. Curry les démontre sous des hypothèses plus générales [12].

Une conséquence importante de ce théorème est que si E admet une forme normale, $i - e$ si VALEUR (E) est définie, alors il existe un algorithme permettant de la calculer.

Remarquons enfin que si $E \equiv F$ alors VALEUR (E) \equiv VALEUR (F)

1.2.6. Extensionnalité

Étant donnée la définition de la convertibilité, il est clair que $E' \equiv F'$ entraîne $\lambda x E' \equiv \lambda x F'$; ainsi $Ex \equiv Fx$ entraîne $\lambda x (Ex) \equiv \lambda x (Fx)$. Si x n'est pas libre dans E et dans F , par application de la η -règle on a : $Ex \equiv Fx$ entraîne $E \equiv F$.

On dit que le langage CUCH possède la propriété d'extensionnalité. En particulier, soit x une variable non-libre dans E et dans F ; alors

$$Ex \equiv Fx \text{ entraîne VALEUR } (E) \equiv \text{VALEUR } (F)$$

1.2.7. Remarques sur quelques combinateurs

. Le combinateur Y , appelé combinateur de paradoxe par Russell ou opérateur de point-fixe par Rosser, permet de trouver la solution d'une équation récursive.

En effet, soit à trouver a tel que, pour F donnée, $Fa \equiv a$; alors YF est solution car $YF \equiv F(YF)$; le lecteur intéressé par l'étude des propriétés de ce point fixe pourra consulter Morris [20].

. Les combinateurs *if* et *si* sont introduits afin de modéliser les alternatives usuelles IF... THEN... ELSE et IF... THEN, dans la mesure où 0 et 1 sont pris comme modèle des valeurs de vérité FAUX et VRAI. Ainsi :

$$\begin{aligned} \text{if } abcd &\equiv \underline{0} (\lambda zc) bd \equiv bd \text{ si } ad \equiv \underline{0} \\ &\underline{1} (\lambda zc) bd \equiv \lambda zcbd \equiv cd \text{ si } ad \equiv \underline{1} \end{aligned}$$

et

$$\begin{aligned} \text{si } abd &\equiv bd \text{ si } ad \equiv \underline{0} \\ &d \text{ si } ad \equiv \underline{1} \end{aligned}$$

Nous nous servons de ces propriétés pour définir la sémantique des diagrammes de Böhm et Jacopini (§ 2.2)

. Le combinateur \circ est tel que $\circ abc \equiv b(ac)$; il sera utilisé pour modéliser la composition des fonctions, au sens usuel (§ 2.2).

. Les constantes $\Omega, a, b, \dots, \alpha, \beta, \dots$ ne possèdent pas de C -règles associées $i - e$ ne sont pas des combinateurs; Ω sera prise pour modèle de l'objet surdéfini ou erreur. Quant aux constantes $a, b, \dots, \alpha, \beta$, elles représentent les instructions de base et données des organigrammes : ce choix est justifié par le fait que nous sommes uniquement préoccupés par la sémantique induite par les structures de contrôle des langages; ceci aura comme vertu de simplifier mainte définition (§ 2.2, § 3.2).

1.3. Un modèle de la pile

La pile est une structure de données habituellement présentée comme une liste fonctionnant en "LIFO", autrement dit une pile est une liste, dont l'usage est limité aux trois opérations d'accès à l'élément sommet, d'ajout d'un élément sur le sommet et de retrait de l'élément sommet et vérifiant un certain nombre d'axiomes [17].

Le but du modèle présenté ici est de montrer d'une part que ces axiomes ne sont que des propriétés déduites du modèle et d'autre part de démontrer que les opérations de pile sont indépendantes de sa longueur.

Pour ce faire, nous définissons une pile de longueur n comme un couple constitué de son élément sommet et de la pile de longueur $n - 1$, le modèle du couple $\langle a, b \rangle$ étant la λ -expression $\lambda z(zab)$ (cf. Annexe § 2). Nous introduisons trois opérateurs *empile*, *top*, *pop* dont nous montrons qu'ils sont bien modèles des opérateurs de pile (théorème 1) et qu'ils jouissent des propriétés qui leur sont attachées (théorème 2).

Formellement, nous posons :

$$P_0 = K\Omega$$

$$\forall_{n \in \mathbf{N}}, P_n = \lambda a_1 \dots \lambda a_{n+1} \lambda z(z a_{n+1} (P_n a_1 \dots a_n))$$

$$\textit{empile} = BC(CI)$$

$$\textit{top} = CIK$$

$$\textit{pop} = CI(KI)$$

THÉORÈME 1 :

$$\forall a_i, i = [1, n], \forall n, \textit{empile} a_{n+1} (P_n a_1 \dots a_n) \equiv P_{n+1} a_1 \dots a_{n+1}$$

$$\textit{top} (P_{n+1} a_1 \dots a_{n+1}) \equiv a_{n+1} \textit{ et } \textit{top} P_0 \equiv \Omega$$

$$\textit{pop} (P_{n+1} a_1 \dots a_{n+1}) \equiv P_n a_1 \dots a_n \textit{ et } \textit{pop} P_0 \equiv \Omega$$

Preuve : voir Annexe

THÉORÈME 2 :

Soit $\Pi_1 = P_n a_1 \dots a_n$ et $\Pi_0 = P_0$; alors, quel que soit le mot x , on a :

top (*empile* $x\Pi_i$) $\equiv x$ $i = 0,1$

pop (*empile* $x\Pi_i$) $\equiv \Pi_i$ $i = 0,1$

empile (*top* Π_1)(*pop* Π_1) $\equiv \Pi_1$

Preuve : Les deux premières propriétés sont évidentes car elles découlent directement du théorème précédent; pour la troisième, voir Annexe.

Nous sommes donc parfaitement en droit d'affirmer que cette construction est un modèle de la pile, l'interprétation de l'erreur en Ω justifiant P_0 comme modèle de la pile vide; pour une étude plus complète de ce modèle le lecteur pourra consulter Robinet [26].

Notation : $P_n a_1 \dots a_n$ sera noté par la suite

$$\langle a_n, \langle a_{n-1}, \langle \dots, \langle a_1 \rangle \dots \rangle \rangle \rangle.$$

II. TRADUCTION DES DIAGRAMMES DE BÖHM ET JACOPINI

Nous donnons ici une traduction des diagrammes de Böhm et Jacopini dans le langage CUCH, c'est-à-dire une interprétation σ_B du langage des diagrammes dans l'ensemble des mots de CUCH. Nous prouvons sa correction et nous en démontrons quelques propriétés fondamentales. Mais auparavant rappelons quelques notations et résultats essentiels [7, 27].

2.1. Définitions et théorème de Böhm et Jacopini

Soient :

. X un ensemble d'objets x , nommés contextes, complété éventuellement avec les valeurs booléennes t et f ;

. Ψ un ensemble de prédicats unaires α, β, \dots sur X ;

. 0 un ensemble d'applications a, b, \dots de X dans X contenant Λ l'application identité;

. $\mathbf{T}, \mathbf{F}, \mathbf{K}$ trois opérations telles que, $v \in \{t, f\}$,

$$\mathbf{T}(x) = \langle t, x \rangle, \quad \mathbf{F}(x) = \langle f, x \rangle, \quad \mathbf{K} \langle v, x \rangle = x$$

. ω un prédicat tel que $\omega \langle v, x \rangle = v$

. Π, Δ, Φ trois diagrammes de base tels que

$$\begin{aligned} \Pi(a, b)(x) &= b(a(x)) \\ \Delta(\alpha, a, b)(x) &= a(x) \text{ si } \alpha(x) \text{ est vrai} \\ &= b(x) \text{ si } \alpha(x) \text{ est faux} \end{aligned}$$

$$\Phi(\alpha, a)(x) = a^n(x) \text{ avec } n, \text{ le plus petit entier,} \\ \text{s'il existe, tel que } \alpha(a^n(x)) \text{ soit faux.}$$

Soit $\mathcal{D}(\Psi, 0)$ la classe des applications de X dans X , représentables au moyen d'organigrammes construits sur $\Psi \cup 0$.

Alors, tout élément de $\mathcal{D}(\Psi, 0)$ est représentable par un élément de $\mathcal{D}'(\Psi, 0)$ où $\mathcal{D}'(\Psi, 0)$ est l'ensemble des organigrammes composés à partir des diagrammes Π , Δ et Φ , construits sur $\Psi \cup 0 \cup \{ \mathbf{K}, \mathbf{T}, \mathbf{F}, \omega \}$ et opérant sur des piles de contexte.

Rappelons que les contextes ou environnements d'un calcul sont structurés en pile afin de garder trace des chemins suivis dans le calcul [27].

2.2. Définition de σ_B

Soient :

- . $\mathcal{A}, \mathcal{B} \in \mathcal{D}'(\Psi, 0)$
- . $a \in 0$
- . $\alpha \in \Psi$.

Alors l'application σ_B de $\mathcal{D}'(\Psi, 0)$ dans CUCH est définie inductivement comme suit :

1. $\sigma_B(a) = a$ $\sigma_B(\Lambda) = I$
2. $\sigma_B(\alpha) = \alpha$
3. $\sigma_B(\Pi(\mathcal{A}, \mathcal{B})) = \circ \sigma_B(\mathcal{A})\sigma_B(\mathcal{B})$
4. $\sigma_B(\Delta(\alpha, \mathcal{A}, \mathcal{B})) = \text{if } \alpha \sigma_B(\mathcal{A})\sigma_B(\mathcal{B})$
- 4'. $\sigma_B(\Delta(\alpha, \mathcal{A}, \Lambda)) = \text{si } \alpha \sigma_B(\mathcal{A})$
5. $\sigma_B(\Phi(\alpha, \mathcal{A})) = Y\lambda x(\text{si } \alpha(\circ \sigma_B(\mathcal{A})x))$
6. $\sigma_B(\mathbf{K}) = \text{pop}$
7. $\sigma_B(\mathbf{T}) = \text{empile } \underline{0}$ et $\sigma_B(t) = \underline{0}$
8. $\sigma_B(\mathbf{F}) = \text{empile } \underline{1}$ et $\sigma_B(f) = \underline{1}$
9. $\sigma_B(\omega) = \text{top}$
10. Si $\langle v, x \rangle$ est une pile de contexte alors

$$\sigma_B(\langle v, x \rangle) = \text{empile } \sigma_B(v)\sigma_B(x) \text{ avec } v \in \{ t, f \}$$

11. Si x est une donnée alors $\sigma_B(x) = x$.

REMARQUE 1 : N'étant pas préoccupés par la sémantique des instructions de base et des données des organigrammes, nous les avons supposées constantes du langage CUCH, sans C-règles, au même titre que Ω (§ 1.2.10); ceci allège la définition de σ_B et explique le pourquoi des définitions 1, 2 et 11.

REMARQUE 2 : Si les définitions 3, 4, 4' n'appellent pas de commentaire (cf. § 1.2.10), la définition 5 peut être vue informellement comme l'expression de la solution d'une équation récursive, c'est-à-dire, l'expression d'un point fixe de l'équation $\Phi(\alpha, \mathcal{A}) = \text{si } \alpha \text{ alors } \mathcal{A}; \Phi(\alpha, \mathcal{A})$ sinon Λ .

REMARQUE 3 : Les définitions 6, 7, 8, 9 proviennent trivialement des définitions des opérateurs de pile (§ 1.3) et des opérateurs de contexte (§ 2.1).

2.3. Correction de σ_B

Reste à montrer que σ_B respecte la sémantique des diagrammes, c'est-à-dire montrer que σ_B est correcte [27].

THÉORÈME 3 : *L'application σ_B est correcte.*

Preuve : Elle se fait par induction sur la structure des éléments de $\mathcal{D}(\Psi, 0)$; afin d'alléger les écritures, on notera $\forall \mathcal{A}, \sigma_B(\mathcal{A}) = \mathcal{A}'$ et sauf avis contraire $\forall x, \sigma_B(x) = x'$.

1. $\sigma_B(a)\sigma_B(x) \equiv ax'$ et $\sigma_B(\Lambda)\sigma_B(x) \equiv Ix' \equiv x'$.
3. $\sigma_B(\Pi(\mathcal{A}, \mathcal{B}))\sigma_B(x) = \circ \mathcal{A}'\mathcal{B}'x' = \mathcal{B}'(\mathcal{A}'x')$.
4. $\sigma_B(\Delta(\alpha, \mathcal{A}, \mathcal{B}))\sigma_B(x) = \text{if } \alpha\mathcal{A}'\mathcal{B}'x' \equiv \mathcal{A}'x' \text{ si } \alpha x' \equiv \underline{0},$
- 4'. $\sigma_B(\Delta(\alpha, \mathcal{A}, \Lambda))\sigma_B(x) \equiv \mathcal{A}'x' \text{ si } \alpha x' \equiv \underline{0},$
 $x' \text{ si } \alpha x' \equiv \underline{1}.$
5. $\sigma_B(\Phi(\alpha, \mathcal{A}))\sigma_B(x) = Y\lambda x(\text{si } \alpha(\circ \mathcal{A}'x'))x'$

Posons $\theta = \lambda x(\text{si } \alpha(\circ \mathcal{A}'x))$; alors, comme $Y\theta \equiv \theta(Y\theta)$,

$$\begin{aligned} \sigma_B(\Phi(\alpha, \mathcal{A}))\sigma_B(x) &= Y\theta x' \equiv \theta(Y\theta)x' \\ &\equiv \lambda x(\text{si } \alpha(\circ \mathcal{A}'x))(Y\theta)x' \end{aligned}$$

Par application de la β -règle, il vient :

$$\begin{aligned} \sigma_B(\Phi(\alpha, \mathcal{A}))\sigma_B(x) &\equiv \text{si } \alpha(\circ \mathcal{A}'(Y\theta))x' \equiv x' \text{ si } \alpha x' \equiv \underline{1}, \\ &\equiv Y\theta(\mathcal{A}'x') \text{ si } \alpha x' \equiv \underline{0}; \end{aligned}$$

de façon générale,

$$\sigma_B(\Phi(\alpha, \mathcal{A}))x' \equiv \sigma_B(\Phi(\alpha, \mathcal{A}))(\mathcal{A}'(\mathcal{A}'(\dots (\mathcal{A}'x') \dots)))$$

c'est-à-dire

$$\begin{aligned} \sigma_B(\Phi(\alpha, \mathcal{A}))x' &\equiv n\mathcal{A}'x' \text{ avec } n, \text{ le plus petit entier, s'il existe, tel que} \\ &\alpha(n\mathcal{A}'x') \equiv \underline{1}. \end{aligned}$$

6.7.8.9. Soit $\langle v, x \rangle$ un contexte de Böhm et Jacopini;

Comme $\sigma_B(\langle v, x \rangle) = \text{empile } \sigma_B(v)\sigma_B(x)$ avec $v \in \{t, f\}$, on a successivement, en appliquant les propriétés de la pile,

$$\begin{aligned} \sigma_B(\mathbf{K})\sigma_B(\langle v, x \rangle) &= \text{pop}(\text{empile } \sigma_B(v)\sigma_B(x)) \\ &\equiv \sigma_B(x) \equiv \sigma_B(\mathbf{K} \langle v, x \rangle) \end{aligned}$$

$$\begin{aligned} \text{puis } \sigma_B(\mathbf{T})\sigma_B(\langle v, x \rangle) &= \text{empile } \underline{0}(\text{empile } \sigma_B(v)\sigma_B(x)) \\ &\equiv \sigma_B(\langle t, \langle v, x \rangle \rangle) \equiv \sigma_B(\mathbf{T} \langle v, x \rangle) \end{aligned}$$

$$\text{puis } \sigma_B(\mathbf{F})\sigma_B(\langle v, x \rangle) \equiv \sigma_B(\mathbf{F} \langle v, x \rangle) \text{ en changeant } \underline{0} \text{ en } \underline{1} \text{ et enfin}$$

$$\begin{aligned} \sigma_B(\omega)\sigma_B(\langle v, x \rangle) &= \text{top}(\text{empile } \sigma_B(v)\sigma_B(x)) \equiv \sigma_B(v) \\ &\equiv \sigma_B(\omega \langle v, x \rangle) \end{aligned}$$

Ainsi, dans tous les cas, $\sigma_B(\mathcal{A}x) \equiv \sigma_B(\mathcal{A})\sigma_B(x)$.

2.4. Sémantique des diagrammes

L'application σ_B ainsi définie associe à tout élément $\mathcal{A} \in \mathcal{D}'(\Psi, 0)$ un mot $\sigma_B(\mathcal{A})$ du langage CUCH dont la valeur est obtenue par application des diverses règles de réduction; la sémantique $\sigma_{B,J}$ de $\mathcal{D}'(\Psi, 0)$ est définie comme VALEUR $\circ \sigma_B$, c'est-à-dire

$$\forall \mathcal{A} \in \mathcal{D}'(\Psi, 0), \quad \sigma_{B,J}(\mathcal{A}) = \text{VALEUR}(\sigma_B(\mathcal{A})).$$

THÉOREME 4 : *Si $\sigma_{B,J}(\mathcal{A})$ est définie, $i - e$ si $\sigma_B(\mathcal{A})$ a une forme normale, alors le calcul de l'organigramme \mathcal{A} s'arrête.*

Preuve : Toutes les sous-expressions de $\sigma_B(\mathcal{A})$ ont une forme normale sauf, peut-être, les sous-expressions $\sigma_B(\Phi_i(\alpha_i, \mathcal{A}_i))$, car elles sont construites à partir des combinateurs *if*, *si* et \circ (cf. Table des combinateurs, § 2.3); si,

$$\forall i, \sigma_B(\Phi_i(\alpha_i, \mathcal{A}_i)) \equiv \underline{n}_i \sigma_B(\mathcal{A}_i),$$

avec $\sigma_B(\mathcal{A}_i)$ en forme normale, alors $\sigma_B(\mathcal{A})$ est en forme normale; si $\exists i$ tel que $\sigma_B(\Phi_i(\alpha_i, \mathcal{A}_i))$ n'a pas de forme normale, alors $\nexists n$ tel que $\forall x, \Phi_i(\alpha_i, \mathcal{A}_i) = \mathcal{A}_i^n(x)$, c'est-à-dire que le calcul de l'organigramme boucle.

2.5. Exemple

Soit l'organigramme $\mathcal{A} = \Phi(\omega, \Pi(\Pi(\mathbf{K}, a), \mathbf{T}))$ et montrons qu'il boucle.

$$\sigma_B(\mathcal{A}) = Y\lambda x \text{ (si top } (\circ(\circ(\circ \text{ pop } a)(\text{empile } \underline{0})))x)$$

Posons θ égal au facteur droit de Y ; pour $v \in \{t, f\}$,

$$\begin{aligned} \sigma_B(\mathcal{A})\sigma_B(\langle v, x \rangle) &= Y\theta\sigma_B(\langle v, x \rangle) \equiv \theta(Y\theta)\sigma_B(\langle v, x \rangle) \\ &\equiv \text{si top } (\circ(\circ(\circ \text{ pop } a)(\text{empile } \underline{0}))) (Y\theta)\sigma_B(\langle v, x \rangle) \end{aligned}$$

Si *top* $\sigma_B(\langle v, x \rangle) \equiv \underline{1}i - e v = f$ alors $\sigma_B(\mathcal{A}) \equiv I$;
 mais si *top* $\sigma_B(\langle v, x \rangle) \equiv \underline{0}i - e v = t$

$$\begin{aligned} \sigma_B(\mathcal{A})\sigma_B(\langle v, x \rangle) &\equiv \circ(\circ(\circ \text{ pop } a)(\text{empile } \underline{0}))(Y\theta)\sigma_B(\langle t, x \rangle) \\ &\equiv Y\theta(\circ(\circ \text{ pop } a)(\text{empile } \underline{0})\sigma_B(\langle t, x \rangle)) \\ &\equiv Y\theta(\text{empile } \underline{0}(\circ \text{ pop } a\sigma_B(\langle t, x \rangle))) \\ &\equiv Y\theta(\text{empile } \underline{0}(a(\text{pop } \sigma_B(\langle t, x \rangle)))) \\ &\equiv Y\theta(\text{empile } \underline{0}(a\sigma_B(x))) \\ &\equiv Y\theta\sigma_B(\langle t, ax \rangle) \end{aligned}$$

Ainsi $\sigma_B(\mathcal{A})\sigma_B(\langle t, x \rangle) \equiv \sigma_B(\mathcal{A})\sigma_B(\langle t, ax \rangle) \equiv \dots$,
 $i - e \sigma_{B,J}(\mathcal{A} \langle t, x \rangle)$ est indéfini, et le calcul se poursuit indéfiniment.

2.6. Propriétés d'équivalence

2.6.1. Définition

Soient $\mathcal{A}, \mathcal{B} \in \mathcal{D}'(\Psi, 0)$, nous dirons que \mathcal{A} et \mathcal{B} sont équivalents et nous noterons

$$\mathcal{A} = \mathcal{B}, \quad \text{ssi}, \quad \forall x \in X, \quad \sigma_B(\mathcal{A})\sigma_B(x) \equiv \sigma_B(\mathcal{B})\sigma_B(x).$$

REMARQUE 1 : Cette définition est plus générale que celle donnée dans [25] et [29]; toutefois les propriétés qui vont suivre se démontrent de la même façon.

REMARQUE 2 : En vertu du deuxième corollaire du théorème de Church-Rosser (§ 1.2.7) et du théorème 3 (§2.5) on déduit de cette définition que \mathcal{A} s'arrête ssi \mathcal{B} s'arrête.

2.6.2. Propriétés

Soient $\mathcal{A}, \mathcal{B}, \mathcal{C}$ des organigrammes quelconques, éléments de $\mathcal{D}(\Psi, 0)$. Alors nous avons les propriétés suivantes :

- . $\Pi(\mathcal{A}, \Pi(\mathcal{B}, \mathcal{C})) = \Pi(\Pi(\mathcal{A}, \mathcal{B}), \mathcal{C})$ (associativité de la composition)
- . $\Pi(\mathcal{A}, \Lambda) = \Pi(\Lambda, \mathcal{A}) = \mathcal{A}$ (élément neutre pour la composition)
- . $\Pi(\Delta(\alpha, \mathcal{B}, \mathcal{C}), \mathcal{A}) = \Delta(\alpha, \Pi(\mathcal{B}, \mathcal{A}), \Pi(\mathcal{C}, \mathcal{A}))$ (distributivité à droite de la composition par rapport à l'alternative)
- . $\Pi(\mathcal{A}, \Delta(\alpha, \mathcal{B}, \mathcal{C})) = \Delta(\alpha, \Pi(\mathcal{A}, \mathcal{B}), \Pi(\mathcal{A}, \mathcal{C}))$ ssi \mathcal{A} n'a pas d'effet de bord sur α (distributivité à gauche).

Preuve : Pour les deux premières on se reportera à [25]; pour la troisième posons $\forall \mathcal{A}, \sigma_B(\mathcal{A}) = \mathcal{A}'$; alors

$$\begin{aligned} \sigma_B(\Pi(\Delta(\alpha, \mathcal{B}, \mathcal{C}), \mathcal{A}))\sigma_B(x) &\equiv \circ \text{ (if } \alpha \mathcal{B}'\mathcal{C}'\text{) } \mathcal{A}'x' \\ &\equiv \mathcal{A}' \text{ (if } \alpha \mathcal{B}'\mathcal{C}'x'\text{)} \\ &\equiv \mathcal{A}'(\mathcal{B}'x') \quad \text{si } \alpha x' \equiv \underline{0} \\ \text{et} &\equiv \mathcal{A}'(\mathcal{C}'x') \quad \text{si } \alpha x' \equiv \underline{1} \end{aligned}$$

$$\begin{aligned} \sigma_B(\Delta(\alpha, \Pi(\mathcal{B}, \mathcal{A}), \Pi(\mathcal{C}, \mathcal{A})))\sigma_B(x) &\equiv iF\alpha(\circ \mathcal{B}'\mathcal{A}')(\circ \mathcal{C}'\mathcal{A}')x' \\ &\equiv \circ \mathcal{B}'\mathcal{A}'x' \equiv \mathcal{A}'(\mathcal{B}'x') \quad \text{si } \alpha x' \equiv \underline{0} \\ &\equiv \circ \mathcal{C}'\mathcal{A}'x' \equiv \mathcal{A}'(\mathcal{C}'x') \quad \text{si } \alpha x' \equiv \underline{1}. \end{aligned}$$

Pour la distributivité à gauche, les calculs sont similaires; elle est vérifiée ssi

$$\forall x, \alpha x' \equiv \alpha(\mathcal{A}'x') \text{ i.e. } B\alpha\mathcal{A}' \equiv \alpha$$

ou, en d'autres termes, ssi \mathcal{A} n'a pas d'effets de bord sur α .

III. SÉMANTIQUE MATHÉMATIQUE DE EXEL

Comme à tout mot de $\mathcal{D}'(\Phi, 0)$ nous savons associer par σ_B un mot du langage CUCH, la sémantique mathématique de EXEL sera définie comme la composition des deux interprétations σ_B et σ_E , cette dernière étant la sémantique du surlangage EXEL dans $\mathcal{D}'(\Psi, 0)$ [27]. Mais auparavant nous allons procéder à quelques rappels.

3.1. Le surlangage EXEL

3.1.1. Grammaire

Soient :

$$V_T = (a, b, \dots, \Lambda, \alpha, \beta, \dots, \text{SI, ALORS, SINON, IS, ;, \{, \}, !0, !1, !2, \dots)$$

le vocabulaire terminal, $V_A = (E, \Psi)$ le vocabulaire auxiliaire et E l'axiome;

EXEL est engendré par la grammaire suivante :

1. $E \rightarrow a|b| \dots | \Lambda|!0|!1|!2| \dots$
2. $\Psi \rightarrow \alpha|\beta| \dots$
3. $E \rightarrow E; E|SI \Psi \text{ ALORS } E \text{ SINON } E \text{ IS} | \{ E \}$

3.1.2. Sémantique

L'interprétation σ_E de EXEL dans $\mathcal{D}'(\Psi, 0)$ est définie comme suit, $\mathcal{A}_1, \mathcal{A}_2, \mathcal{B}$ désignant des mots EXEL quelconques :

$$\begin{aligned} \sigma_E(\Lambda) &= \Lambda \quad \text{et} \quad \sigma_E(a) = \Delta(\omega, \Pi(\Pi(\mathbf{K}, a), \mathbf{T}), \Lambda) \\ \sigma_E(SI \alpha \text{ ALORS } \mathcal{A}_1 \text{ SINON } \mathcal{A}_2 \text{ IS}) &= \Delta(\omega, \Pi(\Pi(\mathbf{K}, \Delta(\alpha, \sigma_E(\mathcal{A}_1), \sigma_E(\mathcal{A}_2))), \mathbf{T}), \Lambda) \\ \sigma_E(\Lambda) &= \Lambda \quad \text{et} \quad \sigma_E(a) = \Delta(\omega, \Pi(\Pi(\mathbf{K}, a), \mathbf{T}), \Lambda) \\ \sigma_E(SI \alpha \text{ ALORS } \mathcal{A}_1 \text{ SINON } \mathcal{A}_2 \text{ IS}) &= \Delta(\omega, \Pi(\Pi(\mathbf{K}, \Delta(\alpha, \sigma_E(\mathcal{A}_1), \sigma_E(\mathcal{A}_2))), \mathbf{T}), \Lambda) \end{aligned}$$

où F figure n fois et $\Pi n - 1$ fois.
 $\sigma_E(\mathcal{A}_1; \mathcal{A}_2) = \Pi(\sigma_E(\mathcal{A}_1), \sigma_E(\mathcal{A}_2))$

3.2. L'interprétation Σ_E

3.2.1. Définition

L'interprétation Σ_E du surlangage EXEL dans le langage CUCH est égale à la composition des interprétations σ_{BL} et $\sigma_E, i - e.$

$$\forall \mathcal{A} \in \text{EXEL}, \quad \Sigma_E(\mathcal{A}) = \sigma_{BL}(\sigma_E(\mathcal{A})) = \text{VALEUR}(\sigma_B(\sigma_E(\mathcal{A})))$$

3.2.2. Formulation de Σ_E

La formulation de Σ_E est faite cas par cas suivant la nature des différents mots de EXEL; il est important de noter qu'au vu de la définition de σ_E et σ_B , cette formulation se fait par simple *substitution*.

Soient :

- . $\mathcal{A}_1, \mathcal{A}_2, \mathcal{B}$ des mots EXEL quelconques;
- . a une instruction de base quelconque;
- . α un prédicat quelconque.

Alors, par définition

$$\begin{aligned} \Sigma_E(\Lambda) &= I \quad \text{et} \quad \Sigma_E(a) = si \ top \ (\circ(\circ \ pop \ a)(empile \ \underline{\quad})) \\ \Sigma_E(si \ \alpha \ \text{ALORS } \mathcal{A}_1 \ \text{SINON } \mathcal{A}_2 \ \text{is}) &= si \ top \ (\circ(\circ \ \overline{pop} \ (if \ \alpha \ \Sigma_E(\mathcal{A}_1) \ \Sigma_E(\mathcal{A}_2))))(empile \ \underline{\quad}) \\ \Sigma_E(\{ \mathcal{B} \}) &= si \ top \ (\circ(Y \ \lambda x (si \ top \ (\circ \ \Sigma_E(\mathcal{B})x))) \ pop) \\ \Sigma_E(!n) &\equiv si \ top \ (\underline{n}(empile \ \underline{\quad})), \forall n \geq 0 \\ \Sigma_E(\mathcal{A}_1; \mathcal{A}_2) &= \circ \ \Sigma_E(\mathcal{A}_1) \ \Sigma_E(\mathcal{A}_2) \end{aligned}$$

REMARQUE 1 :

$$\begin{aligned} \Sigma_E(!n) &= \text{VALEUR} \ (\sigma_B(\Delta(\omega, \Pi(\Pi(\dots, \Pi(\Pi(\mathbf{F}, \mathbf{F}), \mathbf{F}), \dots, \mathbf{F}), \mathbf{F}), \Lambda))) \\ &= \text{VALEUR} \ (si \ top \ (\circ(\dots \ \circ(\circ(empile \ \underline{\quad}))(empile \ \underline{\quad})) \ \dots)(empile \ \underline{\quad}))) \end{aligned}$$

Ce résultat est obtenu par simple substitution; mais

$$\Sigma_E(!n) = \text{VALEUR} \ (\sigma_B(\Delta(\omega, \Pi(\mathbf{F}, \Pi(\dots, \Pi(\mathbf{F}, \mathbf{F}) \dots)), \Lambda)))$$

en vertu de l'associativité de l'opération Π (§ 2.7.2);

$$\begin{aligned}\Sigma_E(!n) &= \text{VALEUR (si top } (\circ(\text{empile } \underline{1})(\dots(\circ(\text{empile } \underline{1})(\text{empile } \underline{1})).\dots))) \\ &\equiv \text{VALEUR (si top } (\underline{n-1}(\circ(\text{empile } \underline{1}))(\text{empile } \underline{1})))\end{aligned}$$

Comme $\underline{n-1}(\circ E)E \equiv \underline{n}E$, $\forall E \in \text{CUCH}$, $\forall n \geq 1$ (Annexe, § 5), on a donc

$$\Sigma_E(!n) = \text{VALEUR (si top } (\underline{n}(\text{empile } \underline{0}))) = \text{si top}(\underline{n}(\text{empile } \underline{0}))$$

REMARQUE 2 :

$$\begin{aligned}\text{si top}(\underline{0}(\text{empile } \underline{1}))x &\equiv x \text{ si top } x \equiv \underline{1} \\ &\equiv \underline{0}(\text{empile } \underline{1})x \text{ si top } x \equiv \underline{0} \\ &\equiv x\end{aligned}$$

On a donc, dans tous les cas, $\Sigma_E(!0) = I$, ce qui *justifie* la convention que $!0$ et Λ sont équivalents, posée comme axiome par Arsac [2], et indispensable à l'établissement des propriétés qui vont suivre.

3.3. Transformations syntaxiques du surlangage EXEL

Nous démontrons quelques propriétés du surlangage EXEL qui fondent des transformations syntaxiques de programme très étudiées présentement [2, 10, 11]; elles découlent directement du modèle en CUCH construit par Σ_E et justifient pleinement les axiomes d'Arsac [3], ce qui élimine totalement les risques d'inconsistance inhérents à tout système purement axiomatique. Les transformations étudiées ici sont la répétition, la fausse itération, la factorisation, l'inversion, l'absorption, un exemple venant illustrer leur emploi.

Dans ce qui suit $\mathcal{A}, \mathcal{B}, \mathcal{C}$ désignent des mots EXEL quelconques et $\mathcal{A}', \mathcal{B}', \mathcal{C}'$ leurs images respectives par Σ_E .

3.3.1. Répétition

$$\text{THEOREME 5 : } \forall \mathcal{A}, \{ \mathcal{A}; \dots; \mathcal{A} \} = \{ \mathcal{A} \}$$

Preuve : D'après la définition de Σ_E , et posant $x' = \Sigma_E(x)$

$$\Sigma_E(\{ \mathcal{A}; \dots; \mathcal{A} \})x' = \text{si top } (\circ(Y\lambda x(\text{si top } (\circ(\underline{n}\mathcal{A}')x))) \text{ pop } x')$$

Si $\text{top } x' = \underline{1}$ alors $\Sigma_E(\{ \mathcal{A}; \dots; \mathcal{A} \}) \equiv I$;
sinon $\text{top } x' \equiv \underline{0}$ et

$$\Sigma_E(\{ \mathcal{A}; \dots; \mathcal{A} \})x' = \circ(Y\lambda x(\text{si top } (\circ(\underline{n}\mathcal{A}')x))) \text{ pop } x';$$

posant $\theta_1 = \lambda x(\text{si top } (\circ(\underline{n}\mathcal{A}')x))$ il vient :

$$\begin{aligned}\Sigma_E(\{ \mathcal{A}; \dots; \mathcal{A} \})x' &\equiv \circ(Y\theta_1) \text{ pop } x' \equiv \text{pop } (Y\theta_1 x') \\ &\equiv \text{pop } (\theta_1(Y\theta_1)x') \\ &\equiv \text{pop } (\text{si top } (\circ(\underline{n}\mathcal{A}'))(Y\theta_1)x');\end{aligned}$$

comme $\text{top } x' \equiv \underline{0}$

$$\begin{aligned}\Sigma_E(\{ \mathcal{A}; \dots; \mathcal{A} \})x' &\equiv \text{pop } (\circ(\underline{n}\mathcal{A}'))(Y\theta_1)x' \equiv \text{pop } (Y\theta_1(\underline{n}\mathcal{A}'x')) \\ &\equiv \text{pop } (Y\theta_1(\underline{n}\mathcal{A}'(\dots(\underline{n}\mathcal{A}'x')\dots))) \\ &\equiv \text{pop } (Y\theta_1(\underline{m} \times \underline{n}\mathcal{A}'x')) \\ &\text{avec } m \text{ le plus petit entier, s'il existe, tel que} \\ &\text{top } (\underline{m} \times \underline{n}\mathcal{A}'x') \equiv \underline{1}.\end{aligned}$$

De la même façon, on obtient pour $\{ \mathcal{A} \}, i - e n = 1,$

$$\Sigma_E(\{ \mathcal{A} \})x' = \text{pop}(Y\theta_2(\underline{p}\mathcal{A}'x')), \quad \text{où } \theta_2 = \lambda x(\text{si top}(\circ\mathcal{A}'x')).$$

avec p le plus petit entier, s'il existe, tel que $\text{top}(\underline{p}\mathcal{A}'x') \equiv \underline{1}$

On a donc $p = m \times n$ et

$$\Sigma_E(\{ \mathcal{A} \})x' = \Sigma_E(\{ \mathcal{A}; \dots; \mathcal{A} \})x' = \text{pop}(\underline{m \times n}\mathcal{A}'x') \text{ si } m \text{ existe} \\ = \text{indéfini sinon}$$

Notons, pour l'utiliser au paragraphe 3.3.3, que :

$$\Sigma_E(\{ \mathcal{A}; \dots; \mathcal{A} \}) = B \text{ pop}(\underline{m \times n} \Sigma_E(\mathcal{A}))$$

3.3.2. Définitions

Afin de définir en EXEL la notion de "boîte noire", construction à une seule entrée et un seul successeur pris séquentiellement, nous poserons avec Arzac les définitions qui suivent [3].

On appellera *instruction terminale*, toute instruction de base ou !n exécutée en dernier dans un mot EXEL.

Comme $\Sigma_E(\mathcal{A}) \equiv \Sigma_E(\mathcal{A}; \Lambda) \equiv \Sigma_E(\mathcal{A}; !0)$, on ne change pas la signification d'un mot EXEL en rajoutant derrière toute instruction terminale le mot ";!0".

Nous noterons $\mathcal{A} + p$ le mot EXEL obtenu en augmentant de p l'indice n de toute instruction terminale de \mathcal{A} , éventuellement complétée par ";!0" si elle est différente d'une sortie indicée.

Ainsi soit $\mathcal{F} = \{ \text{SI } \alpha \text{ ALORS !} / \text{ SINON } a \text{ IS} \}$.

Alors $\mathcal{F} + p = \{ \text{SI } \alpha \text{ ALORS !} p + 1 \text{ SINON } a \text{ IS} \}$.

Soit $!n_i$ une sous-expression de \mathcal{A} ; soit p_i la profondeur de $!n_i$ dans \mathcal{A} , $i - e$ la différence entre les nombres d'accolades ouvrantes et d'accolades fermantes à gauche de $!n_i$ dans [27].

On dit que \mathcal{A} est une *expression propre* si toutes ses instructions terminales ($i - e$ les $!n_i$) sont telles que $p_i = n_i = 0$ ou bien $p_i = n_i$ et il n'existe pas dans \mathcal{A} de sous-expression à droite de l'accolade fermante de l'itération de profondeur 0 dans \mathcal{A} .

Ainsi $a; \{ \text{SI } \alpha \text{ ALORS } b \text{ SINON !}1 \text{ IS}; c \}$ et $\{ \text{SI } \alpha \text{ ALORS } a \text{ SINON } \{ b; \{ \text{SI } \beta \text{ ALORS } c \text{ SINON !}3 \text{ IS}; !2 \} \}$ sont deux mots propres.

Les expressions propres d'EXEL sont donc les seules expressions telles qu'en vertu de la sémantique σ_E , si $\langle t, x_1 \rangle$ était le contexte avant leurs exécutions, il est $\langle t, x_2 \rangle$ après [27]; il n'y a pas de valeur f empilée dans le contexte et x_2 ne dépend que de x_1 et des instructions et prédicats de base.

3.3.3. Fausse itération

THÉORÈME 6 : $\forall \mathcal{A}, \{ \mathcal{A} + 1 \} = \mathcal{A}$

Preuve : Par application de la remarque du § 3.3.1 on a :

$\Sigma_E(\{ \mathcal{A} + 1 \})x' \equiv B \text{ pop } (p \Sigma_E(\mathcal{A} + 1))x'$, avec $\Sigma_E(x) = x'$
où p est le plus petit entier s'il existe tel que $\text{top}(p \Sigma_E(\mathcal{A} + 1)x') \equiv \underline{1}$.

Comme $\mathcal{A} + 1$ ne peut être une expression propre,

$$\Sigma_E(\mathcal{A} + 1)x' \equiv \langle \underline{1}, \Sigma_E(\mathcal{A})x' \rangle; \text{top}(p \Sigma_E(\mathcal{A} + 1)x') \equiv \underline{1}$$

est donc vérifié pour $p = 1$ et l'on a :

$$\begin{aligned} \Sigma_E(\{ \mathcal{A} + 1 \})x' &\equiv B \text{ pop } (\underline{1} \Sigma_E(\mathcal{A} + 1))x' \equiv \text{pop } (\Sigma_E(\mathcal{A} + 1)x') \\ &\equiv \text{pop } \langle \underline{1}, \Sigma_E(\mathcal{A})x' \rangle \equiv \Sigma_E(\mathcal{A})x' \end{aligned}$$

3.3.4. Factorisation

$\forall \mathcal{A}, \{ \mathcal{B}; \mathcal{A} \} = \mathcal{B}; \{ \mathcal{A}; \mathcal{B} \}$ si \mathcal{B} est une expression propre.

Preuve : Toujours $x' = \Sigma_E(x)$, on a, d'après le § 3.3.1,

$$\forall x', \Sigma_E(\{ \mathcal{B}; \mathcal{A} \})x' \equiv \text{pop } (\underline{n}(\circ \mathcal{B}' \mathcal{A}')x')$$

avec n le plus petit entier, s'il existe, tel que

$$\text{top } (\underline{n}(\circ \mathcal{B}' \mathcal{A}')x') \equiv \underline{1}$$

et

$$\begin{aligned} \forall x', \Sigma_E(\mathcal{B}; \{ \mathcal{A}; \mathcal{B} \})x' &\equiv \circ \mathcal{B}' \Sigma_E(\{ \mathcal{A}; \mathcal{B} \})x' \\ &\equiv \text{pop } (\underline{m}(\circ \mathcal{A}' \mathcal{B}')(\mathcal{B}'x')) \end{aligned}$$

avec m le plus petit entier, s'il existe, tel que

$$\text{top } (\underline{m}(\circ \mathcal{A}' \mathcal{B}')(\mathcal{B}'x')) \equiv \underline{1}.$$

Comme, $\forall \mathcal{A}', \mathcal{B}', \forall p \in \mathbb{N}$,

$$p(\circ \mathcal{A}' \mathcal{B}')(\mathcal{B}'x') \equiv \mathcal{B}'(p(\circ \mathcal{B}' \mathcal{A}')x'),$$

m est donc le plus petit entier, s'il existe, tel que

$$\text{top } (\mathcal{B}'(\underline{m}(\circ \mathcal{B}' \mathcal{A}')x')) \equiv \underline{1}$$

De deux choses l'une, ou bien $x' = \langle \underline{1}, x'' \rangle$, et d'après la définition de σ_B , donc de Σ_E ,

$$\forall \mathcal{A}', \mathcal{A}' \langle \underline{1}, x'' \rangle \equiv \langle \underline{1}, x'' \rangle,$$

on a donc $n = m = 0$ et $\Sigma_E(\{ \mathcal{B}; \mathcal{A} \})x' \equiv \Sigma_E(\mathcal{B}; \{ \mathcal{A}; \mathcal{B} \})x' \equiv x'$; ou bien $x' = \langle \underline{0}, x'' \rangle$ et, \mathcal{B} étant une expression propre, \mathcal{B}' ne peut empiler des valeurs $\underline{1}$ sur la pile de contexte : on a donc $\text{top } (\mathcal{B}' \langle v, x'' \rangle) \equiv \underline{1}$, avec $v \in \{ \underline{0}, \underline{1} \}$, ssi $v = \underline{1}$; comme $\underline{m}(\circ \mathcal{B}' \mathcal{A}')x' \equiv \langle v, x'' \rangle$, \underline{m} est le plus petit entier, s'il existe, tel que $\text{top } (\underline{m}(\circ \mathcal{B}' \mathcal{A}')x') \equiv \underline{1}$; on a $\underline{n} = \underline{m}$ et

$$\underline{m}(\circ \mathcal{B}' \mathcal{A}')x' \equiv \underline{n}(\circ \mathcal{B}' \mathcal{A}')x' \equiv \langle \underline{1}, x''' \rangle;$$

comme $\mathcal{B}' \langle \underline{1}, x''' \rangle \equiv \langle \underline{1}, x''' \rangle$, on a finalement

$$\underline{n}(\circ \mathcal{B}' \mathcal{A}')x' \equiv \underline{m}(\circ \mathcal{A}' \mathcal{B}')(\mathcal{B}'x')$$

et $\Sigma_E(\{ \mathcal{B}; \mathcal{A} \})x' \equiv \Sigma(\mathcal{B}; \{ \mathcal{A}; \mathcal{B} \})x'$.

THÉORÈME 7 : $\forall \mathcal{B}, \{ \mathcal{A}; \mathcal{B} \} = \{ \mathcal{A}; \{ \mathcal{B}; \mathcal{A} \} + 1 \}$ ssi \mathcal{A} est une expression propre.

Preuve : Montrons d'abord que si \mathcal{B} est une expression propre $(\mathcal{B}; \mathcal{C}) + p = \mathcal{B}; \mathcal{C} + p$. Les instructions terminales de $\mathcal{B}; \mathcal{C}$ sont celles de \mathcal{C} car \mathcal{B} est une expression propre.

Alors, posant $x' = \Sigma_E(x)$

$$\Sigma_E(\mathcal{B}; \mathcal{C})x' \equiv \mathcal{C}'(\mathcal{B}'x') \text{ et}$$

$$\Sigma_E((\mathcal{B}; \mathcal{C}) + p)x' \equiv \langle \underline{1}, \langle \underline{1}, \dots, \langle \underline{1}, \mathcal{C}'(\mathcal{B}'x') \rangle \dots \rangle \rangle \text{ où } \underline{1} \text{ figure } p \text{ fois;}$$

d'autre part,

$$\forall x'', \Sigma_E(\mathcal{C} + p)x'' = \langle \underline{1}, \langle \underline{1}, \dots, \langle \underline{1}, \mathcal{C}'x'' \rangle \dots \rangle \rangle$$

où $\underline{1}$ figure p fois, et

$$\Sigma_E(\mathcal{B}; \mathcal{C} + p)x' \equiv \circ \Sigma_E(\mathcal{B})\Sigma_E(\mathcal{C} + p)x' \equiv \Sigma_E(\mathcal{C} + p)(\mathcal{B}'x');$$

comme \mathcal{B} est une expression propre, cette expression n'empile pas de constante $\underline{1}$ et

$$\Sigma_E(\mathcal{B}; \mathcal{C} + p)x' \equiv \langle \underline{1}, \langle \underline{1}, \dots, \langle \underline{1}, \mathcal{C}'(\mathcal{B}'x') \rangle \dots \rangle \rangle$$

où $\underline{1}$ figure p fois.

De ce résultat nous pouvons en particulier déduire que si \mathcal{A} est une expression propre,

$$\mathcal{A}; \{ \mathcal{B}; \mathcal{A} \} + 1 = (\mathcal{A}; \{ \mathcal{B}; \mathcal{A} \}) + 1.$$

Par ailleurs $\mathcal{A} = \mathcal{B}$ implique $\{ \mathcal{A} \} = \{ \mathcal{B} \}$; ceci résulte trivialement de la définition de Σ_E .

On a donc

$$\{ \mathcal{A}; \{ \mathcal{B}; \mathcal{A} \} + 1 \} = \{ (\mathcal{A}; \{ \mathcal{B}; \mathcal{A} \}) + 1 \};$$

comme

$$\{ (\mathcal{A}; \{ \mathcal{B}; \mathcal{A} \}) + 1 \} = \mathcal{A}; \{ \mathcal{B}; \mathcal{A} \}$$

d'après la propriété de fausse itération (§ 3.3.3), on obtient finalement

$$\mathcal{A}; \{ \mathcal{B}; \mathcal{A} \} = \{ \mathcal{A}; \{ \mathcal{B}; \mathcal{A} \} + 1 \}.$$

D'après la propriété de factorisation (§ 3.3.4), \mathcal{A} étant une expression propre,

$$\mathcal{A}; \{ \mathcal{B}; \mathcal{A} \} = \{ \mathcal{A}; \mathcal{B} \} = \{ \mathcal{A}; \{ \mathcal{B}; \mathcal{A} \} + 1 \}.$$

3.3.6. Absorption

Notant $\mathcal{A}[\mathcal{B}/\mathcal{C} : \mathcal{R}]$ le résultat de la substitution de \mathcal{B} à \mathcal{C} dans \mathcal{A} si la relation \mathcal{R} est vérifiée, on a :

THÉORÈME 8 : $\forall \mathcal{A}, \mathcal{B}, \mathcal{A}; \mathcal{B} = \mathcal{A}[\mathcal{B} + n_i / !n_i : n_i = p_i \text{ avec } p_i \text{ profondeur de } !n_i \text{ dans } \mathcal{A}]$

Preuve : Avec $x' = \Sigma_E(x)$, on sait que, $\forall x', \Sigma_E(\mathcal{A}; \mathcal{B})x' \equiv \mathcal{B}'(\mathcal{A}'x')$. L'exécution de \mathcal{A} conduisant à une instruction terminale $!n_j$, deux cas peuvent se présenter : ou bien n_j est supérieur à p_j ou bien n_j lui est égal;

. si $n_j > p_j$ alors $\mathcal{A}'x' \equiv \langle \underline{1}, \underline{1}, \dots, \underline{1}, x'' \rangle \dots \rangle$ où $\underline{1}$ figure $n_j - p_j$ fois et, d'après la définition de Σ_E , $\mathcal{B}'(\mathcal{A}'x') \equiv \mathcal{A}'x'$; comme $\Sigma_E(\mathcal{A}[\mathcal{B} + n_i/!n_i : n_i = p_i]) \equiv \Sigma_E(\mathcal{A})$, on a l'égalité annoncée.

. si $n_j = p_j$ alors l'état de la pile de contexte avant exécution de $!n_j$ était

$$\langle \underline{1}, \underline{1}, \dots, \underline{1}, x'' \rangle \dots \rangle,$$

où $\underline{1}$ figure n_j fois et l'état après exécution est x'' ; ainsi $\mathcal{A}'x' \equiv x''$ et, de ce fait,

$$\mathcal{B}'(\mathcal{A}'x') \equiv \langle \underline{1}, \underline{1}, \dots, \underline{1}, x''' \rangle \dots \rangle$$

où $\underline{1}$ figure k fois, l'instruction terminale de \mathcal{B} qui a été exécutée, fixant la valeur de k . Comme on a substitué $\mathcal{B} + n_j$ à $!n_j$ dans \mathcal{A} , on a donc

$$\Sigma_E(\mathcal{B} + n_j)x' \equiv \langle \underline{1}, \underline{1}, \dots, \underline{1}, x'' \rangle \dots \rangle$$

où $\underline{1}$ figure $n_j + k$ fois, et

$$\Sigma_E(\mathcal{A}'[\mathcal{B} + n_i/!n_i : n_i = p_i])x' \equiv \langle \underline{1}, \underline{1}, \dots, \underline{1}, x \rangle \dots \rangle$$

où $\underline{1}$ figure k fois, les n_j valeurs $\underline{1}$ de tête de la pile de contexte ayant été supprimées par n_j applications de l'opérateur *pop* puisque $!n_j$ était terminale et l'on a, là encore, l'égalité.

3.3.7. Propriétés particulières

Les propriétés d'associativité et l'existence d'un élément neutre pour la composition, démontrée au paragraphe 2.7.2 pour les diagrammes de Böhm et Jacopini restent évidemment vérifiées pour le surlangage EXEL. De plus,

$$\forall \mathcal{A}, \forall n \in \mathbf{N}^+, \mathcal{A}; !0 = !0; \mathcal{A} = \mathcal{A} \text{ et } !n; \mathcal{A} = !n.$$

Preuve : La première propriété résulte directement de la définition de Σ_E (cf. § 3.2.2); pour la seconde, on a successivement :

$$\begin{aligned} \Sigma_E(!n; \mathcal{A})x' &= \circ (\text{si top } (\underline{n}(\text{empile } \underline{1})))\mathcal{A}'x' \\ &\equiv \mathcal{A}'(\text{si top } (\underline{n}(\text{empile } \underline{1})))x' \\ &\equiv \mathcal{A}'(\underline{n}(\text{empile } \underline{1})x') \text{ si } \text{top}x' \equiv \underline{0}. \end{aligned}$$

Or, d'après la définition de Σ_E , tout mot \mathcal{A}' est de la forme *si top* X , où X est mis pour un mot quelconque;

$$\begin{aligned} \Sigma_E(!n; \mathcal{A})x' &\equiv \text{si top } X(\underline{n}(\text{empile } \underline{1})x') \\ &\equiv \underline{n}(\text{empile } \underline{1})x' \end{aligned}$$

$$\text{car } \text{top}(\underline{n}(\text{empile } \underline{1})x') \equiv \underline{1}.$$

Donc

$$\begin{aligned} \text{si } \text{top } x' &\equiv \underline{0}, \Sigma_E(!n; \mathcal{A})x' \equiv \Sigma_E(!n)x'; \\ \text{si } \text{top } x' &\equiv \underline{1}, \Sigma_E(!n; \mathcal{A})x' \equiv \mathcal{A}'x' \equiv x' \equiv \Sigma_E(!n)x' \end{aligned}$$

par le même argument que précédemment.

Q.E.D.

Compte tenu de ces propriétés, la distributivité à droite de la composition par rapport à l'alternative se généralise :

$$\forall \mathcal{A}, \mathcal{B}, \mathcal{C}, \forall \alpha \quad \text{SI } \alpha \text{ ALORS } \mathcal{A} \text{ SINON } \mathcal{B} \text{ IS}; \mathcal{C} = \text{SI } \alpha \text{ ALORS } \mathcal{A}; \mathcal{C} \text{ SINON } \mathcal{B}; \mathcal{C} \text{ IS}$$

3.3.8. Exemple

L'exemple suivant va montrer tout l'intérêt de pareilles propriétés : elles définissent des transformations qui préservent la signification des programmes $i - e$ que toutes les écritures obtenues par application de ces transformations ne sont que des versions d'un même algorithme.

Soit \mathcal{P} le programme EXEL-Algol recherchant si la suite Y de longueur M est sous-suite de la suite X de longueur N .

$$\mathcal{P} = I := 1; \{ J := 1; \{ \text{SI } X[I + J - 1] = Y[J] \text{ ALORS } !1 \\ \text{SINON } J := J + 1; \text{SI } J \leq M \text{ ALORS } R := 1; !2 \text{ SINON } \wedge \text{ IS IS } \} \\ I := I + 1; \text{SI } I \leq N - M + 1 \text{ ALORS } R := 0; !1 \text{ SINON } \wedge \text{ IS } \}.$$

Afin d'alléger l'écriture et de dépasser l'interprétation fixée par les instructions élémentaires, nous réécrivons le programme comme suit :

$$\mathcal{P} = a; \{ b; \{ \text{SI } \alpha \text{ ALORS } !1 \text{ SINON } c; \text{SI } \beta \text{ ALORS } d; !2 \text{ SINON } \wedge \text{ IS IS } \}; \\ e; \text{SI } \gamma \text{ ALORS } f; !1 \text{ SINON } \wedge \text{ IS } \}.$$

Comme b est une expression propre puisque instruction élémentaire, on peut appliquer la factorisation (§ 3.3.4) et obtenir

$$\mathcal{P} = a; b; \{ \{ \text{SI } \alpha \text{ ALORS } !1 \text{ SINON } c; \text{SI } \beta \text{ ALORS } d; !2 \text{ SINON } \wedge \text{ IS IS } \}; \\ e; \text{SI } \gamma \text{ ALORS } f; !1 \text{ SINON } \wedge \text{ IS}; b \}.$$

Par application de la distributivité généralisée (§ 3.3.7), on a :

$$\mathcal{P} = a; b; \{ \{ \text{SI } \alpha \text{ ALORS } !1 \text{ SINON } c; \text{SI } \beta \text{ ALORS } d; !2 \text{ SINON } \wedge \text{ IS IS } \}; \\ e; \text{SI } \text{ALORS } f; !1 \text{ SINON } b \text{ IS } \}.$$

Comme le premier $!1$ est de profondeur 1 dans l'itération commençant à $\text{SI } \alpha \dots$, on peut appliquer l'absorption (§ 3.3.6) et obtenir :

$$\mathcal{P} = a; b; \{ \{ \text{SI } \alpha \text{ ALORS } e; \text{SI } \gamma \text{ ALORS } f; !1 \text{ SINON } b \text{ IS} \\ \text{SINON } c; \text{SI } \beta \text{ ALORS } d; !2 \text{ SINON } \wedge \text{ IS IS } \} \}.$$

Enfin, par fausse itération (§ 3.3.3), le programme devient

$$\mathcal{P} = a; b; \{ \text{SI } \alpha \text{ ALORS } e; \text{SI } \gamma \text{ ALORS } f; !1 \text{ SINON } b \text{ IS} \\ \text{SINON } c; \text{SI } \beta \text{ ALORS } d; !1 \text{ SINON } \wedge \text{ IS IS } \}$$

ou, en remplaçant par les instructions élémentaires,

$$\mathcal{P} = I := 1; J := 1; \{ \text{SI } X[I + J - 1] = Y[J] \text{ ALORS } I := I + 1; \\ \text{SI } I \leq N - M + 1 \text{ ALORS } R := 0; !1 \text{ SINON } J := 1 \text{ IS} \\ \text{SINON } J := J + 1; \text{SI } J \leq M \text{ ALORS } R := 1; !1 \text{ SINON } \wedge \text{ IS IS } \}$$

L'application de ces transformations montre en particulier deux choses; la première est que nous sommes partis d'un programme à deux boucles pour arriver à un programme équivalent à une boucle : il est important de constater que le nombre de boucles d'un programme n'en est pas une caractéristique; la seconde est que l'une des versions contient une sortie de niveau 2 et pas l'autre : ceci montre, entre autres, comment un programme peut-être transformé mécaniquement en des versions plus ou moins courtes, plus ou moins « lisibles » ou plus ou moins efficaces.

3.4. Équivalence des structures de contrôle

Les structures de contrôle usuelles que sont les boucles WHILE, REPEAT ou DOFOREVER [22] ne sont que des cas particuliers de celles d'EXEL : tout programme écrit avec ces structures-ci peut être traduit trivialement en EXEL. Toutefois diverses questions sont encore ouvertes : si l'on ne peut traduire tout programme EXEL avec des boucles usuelles [3, 11], peut-on caractériser syntaxiquement une classe de programme EXEL traduisibles?

Un problème important, abordé par Ruggiu [29], mais non totalement résolu, est la caractérisation de la classe de calculs exprimables en EXEL. Enfin, les transformations syntaxiques de programme justifiées dans le présent article sont-elles les seules? Réponse doit être apportée afin de construire des logiciels de transformation automatique pour lesquels les propriétés de consistance et de complétude sont indispensables.

ANNEXE

1. DÉFINITIONS DES COMBINAISONS

$$\begin{array}{ll}
 B \equiv S(KS) K & \circ \equiv CB \\
 C \equiv S(BBS)(KK) & \text{if} \equiv B(B(B(B(CBK)(BW))C))(C(BC)) \\
 W \equiv SS(SK) & \text{si} \equiv B(BW)(BC(CC(KI))) \\
 Y \equiv B(WI)(BWB) & \underline{0} \equiv KI \\
 & \underline{n} \equiv SB \underline{n - 1}, \forall n > 0
 \end{array}$$

Ces définitions ne sont pas uniques, toutefois elles sont équivalentes; ainsi $W \equiv SS(KI) \equiv SS(SK)$ car :

$$\begin{array}{l}
 1^\circ SS(KI) ab \equiv Sa(KIa) b \equiv Sa Ib \equiv ab(Ib) \equiv abb \\
 \text{et } SS(SK) ab \equiv Sa(SKa) b \equiv ab(SKab) \equiv ab(Kb(Ka)) \equiv abb
 \end{array}$$

2° $SS(KI) ab \equiv W ab$ et $SS(SK) ab \equiv W ab$; en appliquant deux fois l'extensionnalité on obtient le résultat cherché.

Montrons un calcul de combinateurs, par exemple si; on a successivement :

$$\begin{aligned}
 ad(\lambda y \lambda z z) bd &\equiv ad(KI) bd \text{ car } KI \equiv \lambda x \lambda y x \lambda z z \\
 &\equiv \lambda y \lambda z z \\
 ad(KI) bd &\equiv Ca(KI) dbd \equiv CC(KI) adbd \\
 &\equiv C(CC(KI)a) bdd \\
 &\equiv BC(CC(KI)) abdd \\
 &\equiv W(BC(CC(KI)) ab) d \\
 &\equiv BW(BC(CC(KI) a) bd \\
 &= B(BW)(BC(CC(KI))) abd;
 \end{aligned}$$

finalement si $\equiv B(BW)(BC(CC(KI)))$

2. JUSTIFICATION DU MODELE DE COUPLE

Avec Church, prenons pour modèle de $\langle a, b \rangle$ la λ -expression $\lambda z(zab)$; pour justifier ce choix, il suffit de prouver que $\lambda z(za_1a_2) \equiv \lambda z(zb_1b_2)$ ssi $a_1 \equiv b_1$ et $a_2 \equiv b_2$.

La première partie est évidente; réciproquement, posant $E = \lambda z(za_1a_2)$ et $F = \lambda z(zb_1b_2)$, de $E \equiv F$ on tire $EK = FK$ et $E(KI) \equiv F(KI)$, c'est-à-dire : $\lambda z(za_1a_2)K \equiv Ka_1a_2 \equiv a_1 \equiv b_1$ et $\lambda z(za_1a_2)(KI) \equiv KIa_1a_2 \equiv a_2 \equiv b_2$.

3. MODÈLE DE LA PILE

3.1. Théorème

$$\begin{aligned}
 \forall n \in \mathbf{N}, \text{ empile } a_{n+1}(P_n a_1 \dots a_n) &\equiv P_{n+1} a_1 \dots a_{n+1} \\
 \text{top } (P_{n+1} a_1 \dots a_{n+1}) &\equiv a_{n+1} \text{ et } \text{top } P_0 \equiv \Omega \\
 \text{pop } (P_{n+1} a_1 \dots a_{n+1}) &\equiv P_n a_1 \dots a_n \text{ et } \text{pop } P_0 \equiv \Omega.
 \end{aligned}$$

Preuve :

$$\begin{aligned}
 \forall x, BC(CI) a_{n+1}(P_n a_1 \dots a_n) &\equiv C(CI a_{n+1})(P_n a_1 \dots a_n) x \\
 &\equiv CI a_{n+1} x(P_n a_1 \dots a_n) \\
 &= x a_{n+1}(P_n a_1 \dots a_n) \\
 &\equiv \lambda z(z a_{n+1}(P_n a_1 \dots a_n)) x \\
 &\equiv P_{n+1} a_1 \dots a_{n+1} x
 \end{aligned}$$

En usant du principe d'extensionnalité, on aboutit au résultat cherché. Pour les deux autres, ils sont conséquences immédiates des calculs du § 2 de l'Annexe.

3.2. Propriété

$$\text{empile } (\text{top } \Pi_1)(\text{pop } \Pi_1) \equiv \Pi_1 \text{ avec } \Pi_1 = P_n a_1 \dots a_n, n > 0.$$

Preuve :

$$\forall x, \text{empile}(\text{top } \Pi_1)(\text{pop } \Pi_1)x \equiv BC(CI)a_n(P_{n-1}a_1 \dots a_{n-1})x$$

car $\text{top } \Pi_1 = a_n$ et $\text{pop } \Pi_1 \equiv P_{n-1}a_1 \dots a_{n-1}$ (cf. Annexe 3.1); alors

$$\begin{aligned} \forall x, \text{empile}(\text{top } \Pi_1)(\text{pop } \Pi_1)x &\equiv C(CIa_n)(P_{n-1}a_1 \dots a_{n-1})x \\ &\equiv CIa_nx(P_{n-1}a_1 \dots a_{n-1}) \\ &\equiv \lambda z(z a_n(P_{n-1}a_1 \dots a_{n-1}))x \\ &\equiv \Pi_1 x \end{aligned}$$

On a le résultat annoncé en appliquant la propriété d'extensionnalité.

4. PROPRIÉTÉ DE \underline{n}

$$\forall E \in \text{CUCH}, \quad \forall n \geq 1, \quad \underline{n-1}(\circ E)E \equiv \underline{n}E.$$

Preuve : Par induction sur n ; pour $n = 1, \forall x, \underline{0}(\circ E)Ex \equiv Ex \equiv \underline{1}Ex$; par extensionnalité $\underline{0}(\circ E)E \equiv \underline{1}E$.

Supposons pour $n = m, \underline{n-1}(\circ E)E \equiv \underline{n}E$

Alors pour $n = m + 1,$

$$\begin{aligned} \forall x, \underline{m}(\circ E)Ex &\equiv \circ E(\underline{m-1}(\circ E)E)x \\ &\equiv \underline{m-1}(\circ E)E(Ex) \\ &\equiv \underline{m}E(Ex) \text{ en vertu de l'hypothèse d'induction.} \end{aligned}$$

Par ailleurs, $\forall p, \forall E, \forall x,$

$$\underline{p}E(Ex) \equiv \underline{p+1}Ex;$$

c'est évident pour $p = 0$ et, l'admettant pour $p = q$, on a pour $p = q + 1,$

$$\begin{aligned} \underline{q+1}E(Ex) &\equiv SB\underline{q}E(Ex) \equiv BE(\underline{q}E)(Ex) \equiv E(\underline{q}E(Ex)) \\ &= E(\underline{q+1}Ex) \text{ d'après l'hypothèse d'induction} \\ &= \underline{q+2}Ex \text{ (Annexe, § 2).} \end{aligned}$$

Ainsi

$\forall x, \forall E, \underline{m}E(Ex) \equiv \underline{m+1}Ex$ et $\underline{m}(\circ E)E \equiv \underline{m+1}E$ ce qui achève la démonstration.

Les combinateurs \underline{n} sont souvent nommés "itérateurs" et l'on montre que $\underline{Bmn} \equiv \underline{m \times n}$ et $\underline{BS(BB)\underline{mn}} = \underline{m+n}$.

BIBLIOGRAPHIE

1. J. ARSAC, L. NOLIN, G. RUGGIU et J. P. VASSEUR, *Le système de Programmation Structurée EXEL* Revue Technique Thomson CSF, Vol. 6, N° 3, 1974, p. 715-736.
2. J. ARSAC, Emploi de méthodes constructives en programmation, *Un dossier de programmation : la fonction d'Ackermann*. R.A.I.R.O., *Informatique Théorique*, Vol. 11, N° 2, 1977, p. 91-112.
3. J. ARSAC, *La construction de programmes structurés*, Dunod, Paris, 1977.
4. D. BERRY, *Implementation of a Contour Model Lambda-Calculus Machine*, TR n° 71-44. Brown University 1971.
5. C. BÖHM and W. GROSS, *Introduction to the CUCH, Automata Theory*, E. R. Caianello, Ed. Academic Press, 1966, p. 35-65.
6. C. BÖHM, *The CUCH as a Formal and Description Language*, Formal languages, Description languages for Computer Programming, T. B. Steel, Ed. North-Holland Pub., 1966, p. 179-197.
7. C. BÖHM and G. JACOPINI, *Flow Diagrams, Turing Machines and Languages with Only Two Formation Rules*, Comm. ACM, Vol. 9, N° 5, 1966, p. 365-371.
8. C. BOHM et M. DEZANI, *A CUCH-Machine : the Automatic Treatment of Bound Variables*, Int. journal of Comp. and Inf. Sciences, Vol. 1, N° 2, 1972, p. 171-186.
9. R. CASTANET, *Sémantique formelle des opérateurs d'un langage de liste*, RAIRO, R-3, 1974, p. 19-36.
10. G. COUSINEAU, *Transformations de programmes itératifs*, in Programmation B. Robinet, Ed., Dunod, 1977, p. 53-74.
11. G. COUSINEAU, *Les arbres à feuilles indicées : un cadre algébrique pour l'étude des structures de contrôle*, Thèse de Doctorat. Université Paris 7, 1977.
12. H. B. CURRY, R. FEYS and W. CRAIG, *Combinatory Logic*, Vol. 1. North-Holland, 1958.
13. M. J. FISCHER, *Lambda-Calculus Schemata*, Proc. ACM Conference on Proving Assertions About Programs. Las Cruces, 1972, p. 104-109.
14. J. R. HINDLEY, B. LERCHER and J. P. SELDIN, *Introduction to Combinatory Logic*, London Mathematical Society. Lecture Notes Series 7. Cambridge University Press. 1972.
15. P. J. LANDIN, *A Lambda-Calculus Approach*, Advances in Programming and Non-Numerical Computation. Pergamon Press, 1966, p. 97-141.
16. P. J. LANDIN, *A Correspondance Between Algol 60 and Church's Lambda-Notation*, Comm. ACM, Vol. 8, N° 2, 3, 1965, p. 89-101 et 158-165.
17. P. LUCAS, *On the Semantics of Programming Languages and Software Devices*, Formal Semantics of Programming Languages. R. Rustin, Ed., Prentice-Hall Inc., 1972, p. 41-57.
18. J. MCCARTHY, *Recursive Functions of Symbolic Expressions and their Computation by Machine*, Part I. Comm. ACM, Vol. 3, N° 4, 1960, p. 184-195.
19. J. MICHEL, *Évaluation automatique de formules de la Logique Combinatoire, Application aux techniques de preuve*. Thèse de Spécialité. Université Pierre-et-Marie-Curie, 1976.
20. J. H. MORRIS, *Lambda-Calculus Models of Programming Languages*, Ph. D., MIT, 1968.
21. L. NOLIN and G. RUGGIU, *Formalization of EXEL* Conference Record of ACM Symposium on Principles of Programming Languages, Boston, 1973, p. 108-119.

22. W. W. PETERSON, T. KASAMI and N. TOKURA, *On the Capabilities of While, Repeat and Exit Statements*, Comm. ACM, Vol. 16, N° 8, 1973, p. 503-512.
23. J. C. REYNOLDS, *GEDANKEN. A Simple Typeless Language Based on the Principle of Completeness and the Reference Concept*, Comm. ACM, Vol. 13, N° 5, p. 308-319, 1970.
24. J. C. REYNOLDS, *Definitional Interpreters for Higher-Order. Programming Languages*. Proc. 25th ACM, National Conference, 1972, p. 717-740.
25. B. ROBINET, *Un modèle sémantique pour un langage simple de programmation*, 1. Fachtagung über Automatentheorie und Formale Sprachen. Lecture Notes in Computer Science, N° 2, Springer-Verlag, 1973, p. 301-310.
26. B. ROBINET, *Contributions à l'étude de réalités informatiques*, Thèse de Doctorat. Université Pierre-et-Marie-Curie. Paris, 1974.
27. B. ROBINET et F. NOZIK, *Sémantique des structures de contrôle*. R.A.I.R.O., Informatique théorique, Vol. 11, N° 1, 1977, p. 63-74.
28. G. RUGGIU, *Les types et les appels de procédure*, in Automata, Languages and Programming. M. Nivat, Ed., North-Holland, 1973, p. 319-330.
29. G. RUGGIU, *De l'organigramme à la formule*, Thèse de Doctorat. Université Pierre et Marie Curie, Paris, 1974.