RAIRO. INFORMATIQUE THÉORIQUE

LAURENT KOTT

Sémantique algébrique d'un langage de programmation type Algol

RAIRO. Informatique théorique, tome 11, n° 3 (1977), p. 237-263

<http://www.numdam.org/item?id=ITA_1977__11_3_237_0>

R.A.I.R.O. Informatique théorique/Theoretical Computer Science
(vol. 11, n° 3, 1977 p. 237 à 263).

SÉMANTIQUE ALGÉBRIQUE D'UN LANGAGE DE PROGRAMMATION TYPE ALGOL (*)

par Laurent KOTT (1)
Communiqué par M. Nivat

Résumé. — *Nous présentons une sémantique d'un langage de programmation évolué qui résout les problèmes posés par la notion d'affectation, de variables locales et d'appel par la valeur des paramètres d'une procédure.*

I. INTRODUCTION

La nécessité de construire un système formel permettant la définition d'une sémantique mathématique des langages de programmation a été mise en évidence par de nombreux auteurs, parmi lesquels nous citerons McCarthy, Morris, Nivat, Park, Scott et Strachey.

La définition d'une sémantique mathématique évite les omissions, les contradictions et les ambiguïtés inhérentes aux spécifications d'une sémantique informelle ainsi que l'a montré les études faites sur le rapport Algol 60.

D'autre part la programmation devient de moins en moins l'art de mettre au point, par tentatives successives, un programme écrit rapidement et, de plus, celui de construire un programme en fournissant simultanément la preuve de sa correction (i.e. qu'il se termine, qu'il calcule bien le résultat attendu, . . .). Le but ultime étant la mise au point de système interactif d'aide à la programmation (une esquisse d'un tel système est présentée par R. Burstall et J. Darlington [2]), il est nécessaire que le langage de programmation employé par l'utilisateur du système ait une sémantique rigoureusement définie.

La sémantique a fait un grand pas en avant lorsque l'on a envisagé un programme comme définissant une fonction, car on a pu utiliser tout l'arsenal

(*) Reçu en septembre 1976, révisé en avril 1977.
(1) U.E.R de Mathématiques, Université Paris VII et CNRS, L.A. 248 Informatique Théorique et Programmation

mathématique déjà connu (J. McCarthy [11], D. Park [16]). La première sémantique construite fut celle dite "du point fixe" ou "dénotationnelle" fondée par D. Scott et C. Strachey [17] et utilisée par Z. Manna et J. Vuillemin [12, 13] pour définir la sémantique d'un langage de programmation voisin d'Algol [13].

Une seconde sémantique dite "algébrique" a été définie par M. Nivat [14, 15]. Cette théorie, utilisée pour exprimer de façon élégante certains problèmes comme celui de l'équivalence de programme [3, 5], n'a jamais servi à la définition d'une sémantique d'un langage de programmation. D'où son emploi dans cet article.

De plus, ce point de vue nous semble le plus propre à résoudre un certain nombre de problèmes laissés en suspens dans l'article précité de Z. Manna et J. Vuillemin [13], concernant l'existence de variables locales (au sens d'Algol) et l'appel par la valeur d'une procédure.

Enfin une dernière raison, qui n'est pas la moindre, nous ramène au second point développé au début de cette introduction : la transformation de programmes exige une vision syntaxique des phénomènes et, par conséquent, une distinction aussi fine que possible entre syntaxe et sémantique que la théorie algébrique réalise excellemment (cf. infra). D'où l'intérêt d'une sémantique *algébrique* d'un langage de programmation d'autant plus que des travaux récents pemettant d'avancer dans la conception de système de transformations de programme ont été déjà faits dans ce cadre algébrique [4, 6, 10].

La sémantique algébrique

Cette théorie, rejoignant celle élaborée par Ianov [7], dissocie les deux éléments qui définissent un programme :

– le schéma de programme qui représente la structure du programme (branchements, conditionnels, ...) en "oubliant" la signification et la nature des fonctions de base et des variables;

– l'interprétation qui restitue la nature et le sens des symboles formels représentent les fonctions de base et les variables du programme originel.

Un exemple, dont il faut excuser la banalité car il recèle quelques difficultés que nous verrons plus loin, va illustrer ce point de vue. Soit le programme Algol suivant :

```
DEBUT ENTIER PROCEDURE fact(x); ENTIER x;
                fact := SI x = 0 ALORS 1 SINON x* fact (x − 1);
      ENTIER n;
      Lire (n); Ecrire (fact (n));
FIN
```

auquel nous associons le schéma de programme en laissant de côté les ordres d'entrée-sortie

$$\varphi(u) = f(g(u), a, h(u, \varphi(k(u))))$$

et l'interprétation I qui est définie par :

— u prend ses valeurs dans l'ensemble des entiers naturels

— m, n étant des entiers et t un élément de { vrai, faux }, les applications suivantes :

$$a_I = 1$$
$$h_I(n) = n - 1 \quad \text{si} \quad n > 0$$
$$h_I(m, n) = m \,.\, n$$
$$g_I(n) = \begin{cases} \text{vrai si } n = 0 \\ \text{faux sinon} \end{cases}$$
$$f_I(t, m, n) = \begin{cases} m \text{ si } t = \text{vrai} \\ n \text{ si } t = \text{faux} \end{cases}$$

On dit que l'interprétation I transforme le schéma de programme en le programme P qui calcule $n!$ pour tout entier naturel.

Esquissons la démarche de la théorie algébrique. Soient F un ensemble de symboles de fonction dont chaque élément f est dotée d'une arité notée $a(f)$ et V un ensemble de variables : nous appelons $M(F, V)$ le F-magma libre engendré par V qui est l'ensemble des termes bien formés (en respectant les arités) à l'aide des éléments de F et de V ou, autrement dit, l'ensemble défini par

— $V \subseteq M(F, V)$

— soit f un élément de F et $t, \ldots, t_{a(f)}$ des éléments de $M(F, V)$ alors $f(t, \ldots, t_{a(f)})$ est aussi un élément de $M(F, V)$.

Nous appellerons *schéma de programme* un système d'équations sur le magma $M(F, V)$ du type suivant :

$$\Sigma' \begin{cases} \varphi_i(v_1, \ldots, v_{n_i}) = t_i \\ 1 \leqslant i \leqslant N \end{cases}$$

avec les notations suivantes :

$\Phi = \{ \varphi_1, \ldots, \varphi_N \}$ est un ensemble de nouveaux symboles, dits symboles de fonction inconnue,

n_i est l'arité du symbole φ_i

t_i est un élément du magma $M(F \cup \Phi, \{ v_1, \ldots, v_{n_i} \})$

(Dans notre exemple, le système Σ est réduit à une seule équation :

$$\varphi(u) = f(g(u), a, h(u, \varphi(k(u))))$$

sur le magma $M(F, V)$, avec $F = \{ a, f, g, h, k \}$ et $V = \{ u \}$)

A tout système Σ de ce type on associe un système de réécriture $\overline{\Sigma}$ sur le magma $M(F, V)$, c'est-à-dire une grammaire algébrique où les symboles φ_i sont les non-terminaux, φ_1 l'axiome, et dont les productions sont

$$\overline{\Sigma} \begin{cases} \varphi_i(v_1, \ldots, v_{n_i}) \to t_i \\ \varphi_i(v_1, \ldots, v_{n_i}) \to \Omega \end{cases}$$

abrégé en $\varphi_i(v_1, \ldots, v_{n_i}) = t_i + \Omega$ $(1 \leqslant i \leqslant N)$ où Ω est un symbole de fonction distingué (d'arité 0).

Cette " grammaire sur le magma " engendre un langage (un sous-ensemble de $M(F, V)$) qui représente la suite des approximations finies du calcul des fonctions inconnues, noté $L(\Sigma, \varphi_1)$, qui est ainsi associé au schéma Σ.

(Dans notre exemple, $\overline{\Sigma}$ contient l'unique règle de réécriture

$$\varphi(u) = f(g(u), a, h(u, \varphi(k(u)))) + \Omega$$

et les premiers termes de $L(\Sigma, \varphi)$ sont :

Ω, $f(g(u), a, h(u, \Omega))$, $f(g(u), a, f(u, f(g(k(u)), a, h(k(u), \Omega))))$, ...etc...

Chacun de ces termes représente un nombre limité d'appels de procédure et, donc, un résultat défini que pour un nombre fini de données.)

Une interprétation discrète I est donnée par :

— un domaine D_I muni d'un ordre, noté $\subseteq$, tel que $x \subseteq y$ si et seulement si $x = \omega$ ou $x = y$ où ω est le plus petit élément de D_I;

— un ensemble de fonctions monotones $\{ f_I \mid f \in F$ et $f_I : D_I^{a(f)} > D_I \}$.

Une interprétation valuée est un couple (I, ν) où I est une interprétation discrète et ν une valuation c'est-à-dire une application de V dans D_I.

(Dans notre exemple l'interprétation I a pour domaine :

— $D_I = \mathbf{N}_I \cup \{ \omega \}$

— les applications définies plus haut sont bien monotones

— une valuation est la donnée d'un entier, disons n, et elle sera notée n.)

Nous avons les résultats suivants dus à M. Nivat [15] :

(i) ν s'étend de façon unique en une application de $M(F, V)$ dans D_I et, si t est un élément de $M(F, V)$, nous notons $(I, \nu)(t)$ le résultat de cette application;

(ii) nous définissons l'application calculée par le schéma Σ sous linterprétation I de la façon suivante : soit $\vec{d}$ un n-uplet la valeur de cette fonction, notée $\mathrm{Val}_I(\Sigma, \varphi_1)$, au point $\vec{d}$ est égal à

$$\mathrm{Val}_I(\Sigma, \varphi_1)(\vec{d}) = \begin{cases} d' \text{ si il existe un mot } t \text{ de } L(\Sigma, \varphi_1) \text{ tel que } (I, \nu)(t) = d' \\ \omega \text{ sinon} \end{cases}$$

où ν est la valuation définie en associant à v_i d_i pour i compris entre 1 et n;

(iii) la validité de cette dernière définition est assurée par le résultat suivant : soient t et t' deux mots de $L(\Sigma, \varphi_1)$ et (I, ν) une interprétation valuée, si $(I, \nu)(t)$ et $(I, \nu)(t')$, éléments de D_I, sont tous deux distincts de ω alors ils sont égaux.

Sur l'exemple déjà utilisé ces résultats s'expriment ainsi : soit n une valuation, considérons les deux mots $f(g(u), a, h(u, \Omega))$ et

$$f(g(u), a, h(u, f(g(k(u)), a, h(k(u), \Omega))))$$

alors

$$(I, n)(f\,(g(u), a, h(u, \Omega))) = f_I(g_I(n), a_I, h_I(n, \omega))$$
$$= \begin{cases} 1 & \text{si } n = 0 \\ \omega & \text{sinon} \end{cases}$$

$$(I, n)(f(g(u), a, h(u, f(g(k(u)), a, h(k(u), \Omega))))$$
$$f_I(g_I(n), a_I, h_I(n, f_I(g_I(k_I(n)), a_I, h_I(k_I(n), \omega)) = \begin{cases} 1 & \text{si } n = 0 \\ 1 & \text{si } n = 1 \\ \omega & \text{sinon} \end{cases}$$

Voici sommairement exposée cette théorie dont le lien avec la sémantique du point fixe et la sémantique dénotationnelle a été fait par M. Nivat [15].

Sémantique d'un langage. Exemple de problèmes.

Avant de définir cette sémantique il est bon de donner un exemple des phénomènes que nous voulons prendre en compte. Soit le programme Algol suivant :

```
(P')  DÉBUT ENTIER PROCÉDURE fact (x); ENTIER x;
                  fact := SI x = 0 ALORS 1 SINON x* fact (x − 1);
            ENTIER a, b;
            Lire (a); b := fact (a);
            a := SI a = (a ÷ 2)* 2 ALORS 1 SINON a ↑ b + b;
            Ecrire (a);
      FIN
```

qui ressemble fort à l'exemple donné plus haut.

Nous allons le transcrire en un schéma de programme suivant la méthode préconisée par Z. Manna et J. Vuillemin [13] qui semble la plus naturelle. D'où le schéma :

$$T \begin{cases} \varphi(u) = f(g(u), a, h(u, \varphi(k(u)))) \\ \psi(u) = f(q(u), a, s(r(u, \varphi(u)), \varphi(u))) \end{cases}$$

et l'interprétation J définie par

son domaine, D_J, qui est égal à $\overline{\mathbf{Z}} = \mathbf{Z} \cup \{\omega\}$

pour tout couple (m, n) de $\overline{\mathbf{Z}} \times \overline{\mathbf{Z}}$ et t de { vrai, faux }

$$a_J = 1$$
$$k_J(n) = n - 1$$
$$h_J(m, n) = m \,.\, n$$
$$s_J(m, n) = m + n$$
$$r_J(m, n) = m^n \quad \text{si} \quad n > 0$$
$$q_J(m, n) = \begin{cases} \text{vrai} & \text{si } n \text{ est pair} \\ \text{faux sinon} \end{cases}$$
$$g_J(n) = \begin{cases} \text{vrai} & \text{si } n = 0 \\ \text{faux sinon} \end{cases}$$

$$f_J(t, m, n) = \begin{cases} m & \text{si } t = \text{vrai} \\ n & \text{si } t = \text{faux} \end{cases}$$

Lors de cette transcription nous avons considéré la variable b comme un intermédiaire dans le calcul (en fait utilisé pour économiser un appel de procédure) et l'instruction d'affectation, $b :=$ fact (a), comme la substitution à la variable b de l'expression fact (a).

Calculons la fonction définie par T sous l'interprétation J. Pour ce faire nous étudions le langage, infini, $L(T, \psi)$ engendré par T considéré comme un système de réécriture; un des mots de ce langage s'écrit $f(q(u), a, s(r(u, \Omega), \Omega)$. Notons n la valuation qui à u associe l'entier n; il est clair que nous avons

$$(J, -2)(f(m(u), a, s(r(u, \Omega), \Omega)) = 1$$

et donc $\text{Val}_J(T, \psi)(-2) = 1$

Or le programme ne calcule rien au point -2 car la procédure fact (appelée par l'instruction $b :=$ fact (a)) boucle indéfiniment pour tout entier négatif. Le schéma T n'est pas celui qu'il faut associer au programme P' et cette erreur est due, nous semble-t-il, à l'existence de la variable b, simple intermédiaire, et au traitement de l'instruction d'affectation.

C'est donc ce type de phénomènes, qui se retrouve dans l'appel par la valeur, que nous souhaitons expliciter et qui exige une formulation légèrement différente de la théorie algébrique en introduisant un nouveau type de systèmes d'équations sur le magma $M(F, V)$ dits systèmes schématiques généralisés.

La définition de cette sémantique se fait ainsi :

(i) définition et étude des systèmes schématiques généralisés,

(ii) définition de la sémantique de ces systèmes,

(iii) transcription de tout programme écrit dans le langage étudié; c'est-à-dire qu'à tout programme P est associé un couple (Σ_P, I_P) d'un système schématique généralisé et d'une interprétation.

(iv) le couple (Σ_P, I_P) définit une fonction qui, par définition, est la fonction calculée par le programme P.

Plan de l'article

1. Deux exemples, aussi simples que possible, dont le choix illustre notre but et les solutions apportées aux problèmes soulevés plus haut.

2.

2.1. Description du langage de programmation

2.2. Algorithme de transcription

2.3. Fonction calculée par un programme.

3. Systèmes schématiques généralisés; définition et étude

4. Deux autres exemples.

2. Deux exemples

Ces deux exemples sont destinés à mettre en évidence certaine des propriétés de notre sémantique. Aussi reprendrons-nous le programme déjà donné dans l'introduction pour premier exemple; le second, quant à lui, concernera l'appel par la valeur des arguments d'une procédure.

De plus ils ont été choisis suffisamment simples pour que l'intuition du lecteur puisse pallier à l'absence des définitions qui sont reportées dans les paragraphes suivants.

2.1. *Exemple* 1

Il s'agit du programme suivant, noté P (*cf. p.* 5) :

```
DEBUT ENTIER PROCÉDURE fact (x); ENTIER x;
                  fact := SI x = 0 ALORS 1 SINON x * fact (x - 1);
      ENTIER a, b;
      Lire (a); b := fact (a);
      a := SI a = (a ÷ 2).* 2 ALORS 1 SINON a ↑ b + b;
      Ecrire (a);
FIN
```

Nous avons vu plus haut les problèmes posés par l'instruction $b :=$ fact (a) et qui exigent l'introduction d'un nouveau type de système d'équation. Soit le système schématique généralisé suivant :

$$\Sigma \begin{cases} \psi(u) = f(q(u), a, s(r(u, w), w) \\ \varphi(u) = f(g(u), a, h(u, \varphi(k(u)))) \\ w \quad = \varphi(u) \end{cases}$$

Le lecteur se convaincra aisément qu'il s'agit d'une transcription « mot à mot » du programme P : la première équation représentant le corps du programme et la seconde celui de la procédure. En outre une variable distinguée, w, jouant le un rôle tout à fait particulier montre la différence entre ce système et un schéma de programme au sens de Nivat. Différence qui se fait en deux points :

– dans le membre droit de la première équation il existe une occurrence d'une variable qui n'en a pas dans le membre gauche;

– cette variable distinguée est, elle aussi, définie par une équation.

Avant de montrer la méthode qui permet d'obtenir la sémantique d'un tel système rappelons que l'interprétation (discrète) induite par le programme P, notée J, est définie par :

– son domaine $\overline{\mathbf{Z}} = \mathbf{Z} \cup \{ \omega \}$

– pour tout couple (m, n) de $\overline{\mathbf{Z}} \times \overline{\mathbf{Z}}$ et t de { vrai, faux }

$$\begin{aligned} a_J \quad &= 1 \\ k_J(h) \quad &= n - 1 \\ h_J(m, n) \quad &= m \,.\, n \end{aligned}$$

$$r_J(m, n) = m^n \quad \text{si} \quad n > 0$$
$$s_J(m, n) = m + n$$
$$g_J(n) = \begin{cases} \text{vrai} & \text{si} \quad n \text{ est pair} \\ \text{faux} & \text{si} \quad n \text{ est impair} \end{cases}$$
$$g_J(n) = \begin{cases} \text{vrai} & \text{si} \quad n \text{ est égal à } 0 \\ \text{faux sinon} \end{cases}$$
$$f_J(t, m, n) = \begin{cases} m & \text{si} \quad t = \textit{vrai} \\ n & \text{si} \quad t = \text{faux} \end{cases}$$

Considérons le système Σ comme un système de réécriture sur le magma par adjonction d'un nouveau symbole de fonction de base Ω et d'équations s'y rattachant. D'où le système :

$$\Sigma \begin{cases} \psi(u) = f(q(u), a, s(r(u, w))) + \Omega \\ \varphi(u) = f(g(u), a, h(u, \varphi(k(u)))) + \Omega \\ w = \varphi(u) \end{cases}$$

que nous notons encore Σ. Soit F l'ensemble des symboles des fonctions de base (i.e. $F = \{ \Omega, a, g, h, k, q, r, s, f \}$), $L(\Sigma, \psi)$ est le langage engendré par le système sur le magma libre $M(F, \{ u, w \})$. Ce langage est fini et contient les deux mots :

- Ω
- $f(q(u), a, s(r(u, w), w))$ que nous noterons t; d'où $L(\Sigma, \psi) = \{ \Omega, t \}$.

Interprétation

C'est à ce point que nous allons opérer de façon distincte de celle décrite dans [15]. En effet pour interpréter les mots de $L(\Sigma, \psi)$ et, plus particulièrement, le mot t nous devons prendre une valuation c'est-à-dire une application de l'ensemble des variables dans le domaine de J. Mais nous allons faire une distinction entre la variable u qui sera *valuée* et la variable w qui sera *calculée*. Notons n la valuation qui à u associe l'entier n et (J, n) l'interprétation valuée correspondante, les mots de $L(\Sigma, \psi)$ sont alors interprétés de la façon suivante :

- $(J, n)(\Omega) = \omega$
- $(J, n)(t) = \begin{cases} \omega & \text{si} \quad (J, n)(w) = \omega \\ f_J(q_J(n), a_J, s_J(r_J(n, (J, n)(w)), (J, n)(w))) & \text{sinon} \end{cases}$

où $(J, n)(w)$ désigne le résultat du calcul de w sous l'interprétation (J, n).

Pour faire ce " calcul " nous allons utiliser l'équation w = φ(u) du système Σ. $\varphi(u)$ est le nombre droit d'une équation tout à fait usuel et nous sommes parfaitement en mesure de définir la valeur au point n de la fonction calculée par $\varphi(u)$ sous l'interprétation J ce que nous avons noté dans l'introduction $\text{Val}_J(\Sigma, \psi)(n)$ et qui dans cet exemple est clairement définie par :

$$\text{Val}_J(\Sigma, \psi)(n) = \begin{cases} n! & \text{si} \quad n \geqslant 0 \\ \omega & \text{sinon} \end{cases}$$

d'où l'interprétation du mot t

$$(J, n)(t) = \begin{cases} \omega \quad \text{si} \quad n < 0 \\ f_J(g_J(n), a_J, I_J(r_J(n, n!), n!)) \end{cases}$$

ou encore

$$(J, n)(t) = \begin{cases} \omega \quad \text{si} \quad n < 0 \\ 1 \quad \text{si} \quad n \geqslant 0 \text{ et } n \text{ est pair} \\ n^{n!} + n! \text{ si } n \geqslant 0 \text{ et } n \text{ est impair} \end{cases}$$

Finalement la fonction calculée par (Σ, ψ) sous l'interprétation J sera telle que

$$\text{Val}_J\ (\Sigma, \psi)(n) = \begin{cases} \omega \quad \text{si} \quad n < 0 \\ 1 \quad \text{si} \quad n \geqslant 0 \text{ et } n \text{ est pair} \\ n^{n!} + n! \text{ si } n \geqslant 0 \text{ et } n \text{ est impair} \end{cases}$$

ce qui est bien la fonction calculée par le programme P.

Sémantique

Nous pouvons, désormais, esquisser la sémantique des systèmes schématiques généralisée (en abrégé S.S.G.) :

(i) un S.S.G. est un système d'équations sur le magma $M(F, V \cup W)$ où V est l'ensemble des variables " normales " et W celui des variables " distinguées ",

(ii) un S.S.G. Σ est considéré comme un système de réécriture sur $M(F, V \cup W)$ qui engendre le langage $L(\Sigma, \varphi_1)$ (φ_1 est l'axiome de Σ)

(iii) soit (I, ν) une interprétation discrète valuée. Seules les variables de V sont valuées et celles de W sont calculées à l'aide de l'équation associée à chaque élément de W

(iv) soit t un élément de $L(\Sigma, \varphi_1)$; $(I, \nu)(t)$ est définie de la façon suivante :

$$(I, \nu)(t) = \begin{cases} \omega \text{ si il existe une occurrence d'un } w \text{ de } W \text{ dont le calcul ne} \\ \text{finit pas (i.e. vaut } \omega) \\ \text{sinon} \begin{cases} \nu(v) \text{ si } t = v \text{ pour quelque } v \text{ de } V \\ f_I((I, \nu)(t_1), \ldots, (I, \nu)(t_{a(f)})) \text{ pour quelque } f \text{ de } F, \\ t_1, \ldots, t_{a(f)} \text{ de } M(F, V \cup W) \text{ et } t = f(t, \ldots, t_{a(f)}) \end{cases} \end{cases}$$

D'où nous déduisons la définition de $\text{Val}_I\ (\Sigma, \varphi_1)$:

soit d un point du domaine D_I et ν la valuation correspondante alors

$$\text{Val}_I(\Sigma, \varphi_1)(d) = \begin{cases} d' \text{ si il existe } t \text{ de } L(\Sigma, \varphi_1) \text{ tel que } (I, \nu)(t) = d' \\ \omega \text{ sinon} \end{cases}$$

Ceci donne une première approximation de la sémantique des S.S.G. et montre la façon dont nous traitons l'affectation dans un langage de

programmation voisin d'Algol. L'exemple suivant va expliquer le traitement de l'appel par la valeur et montrer que cette sémantique exige une formulation plus complexe.

2.2. *Exemple 2*

Cet exemple est, lui aussi, classique mais particulièrement efficace car la différence entre appel par nom et appel par valeur y apparaît très clairement.

Soit P' le programme suivant :

```
DEBUT ENTIER PROCEDURE phi(x, y); VALEUR y; ENTIER x, y;
                phi := SI x = 0 ALORS 1 SINON phi (x - 1, phi (x, y))
      ENTIER a; Lire (a);
      a := phi (a, a); Ecrire (a);
FIN
```

P' est transcrit en le S.S.G. Σ' qui s'écrit

$$\Sigma' \begin{cases} \psi(u) & = \varphi(u, u) \\ \varphi(u, v) & = f(g(u), a, \varphi(k(u), \varphi(u, w))) \\ w & = v \end{cases}$$

Justifions cette transcription. Nous voyons que, au lieu et place de l'équation

$$\varphi(u, v) = f(g(u), a, \varphi(k(u), \varphi(u, v)))$$

transcription " naïve " du corps de la procédure phi, nous avons remplacé v par une variable " distinguée " (i.e. elle n'a pas d'occurrence dans le membre droit de cette équation) w qui est définie par l'équation $w = v$. Ceci correspond au traitement de l'appel par la valeur en Algol : lors de chaque appel de la procédure phi il y a création d'un sous-bloc dans lequel est déclarée une variable locale $- w -$ qui est initialisée par le second paramètre de l'appel de procédure d'où $w = v$.

Transcription

Comme dans l'exemple 1 considérons Σ' comme un système de réécriture sur le magma $M(F, \{ u, v, w \})$ (où $F = \{ \Omega, f, g, k \}$) par adjonction du symbole Ω et d'équations s'y rattachant. D'où le système :

$$\Sigma' \begin{cases} \psi(u) & = \varphi(u, u) + \Omega \\ \varphi(u, v) & = f(g(u), a, \varphi(k(u), \varphi(u, w))) + \Omega \\ w & = v \end{cases}$$

Ici apparaît un problème caractéristique des S.S.G. : chaque réécriture de entraîne la naissance d'une nouvelle équation $w =$ " second argument de φ " comme chaque appel de phi entraîne la création d'un sous-bloc avec déclaration d'une nouvelle variable. Si bien que nous sommes conduits à différencier de façon syntaxique les w introduits par les réécritures du symbole.

Nous simulons Σ' par un nouveau système *infini* d'équations en différenciant chacune des réécritures possibles de φ par l'introduction d'une infinité de symboles de fonction inconnue; d'où le système Σ'_∞ :

$$\begin{cases} \psi(u) = \varphi(u, u) + \Omega \\ \varphi_m(u, v) = f(g(u), a, \varphi_m(k(u), \varphi_{m\beta}(u, w_m))) + \Omega \\ w_m = v \\ m \in \{ \alpha, \beta \}^* \end{cases}$$

On vérifiera aisément que $L(\Sigma, \psi) = L(\Sigma'_\infty, \psi)$.

Soit $f(g(u), a, f(g(k(u)), a, \Omega))$ un élément de $L(\Sigma', \psi)$; pour l'obtenir nous avons engendré les équations :

$$\begin{aligned} w &= u \\ w &= \varphi(u, w) \end{aligned}$$

et bien d'autres; d'où un nouveau problème posé.

En effet lors de l'interprétation de $L(\Sigma'_\infty, \psi)$ nous allons calculer les variables w_m en utilisant l'équation définissant les w_m; or celles-ci font intervenir des symboles du type $w_{m'}$ qui vont à leur tour introduire des variables distinguées qu'il faudra de nouveau calculer. Nous voyons apparaître un processus sans fin ce qui n'est pas souhaitable et nous devons nous doter d'un moyen de le stopper. Ceci se fait – comme en machine – par limitation de la profondeur de récursion en remarquant que l'ensemble des index des symboles m est l'arbre binaire infini suivant :

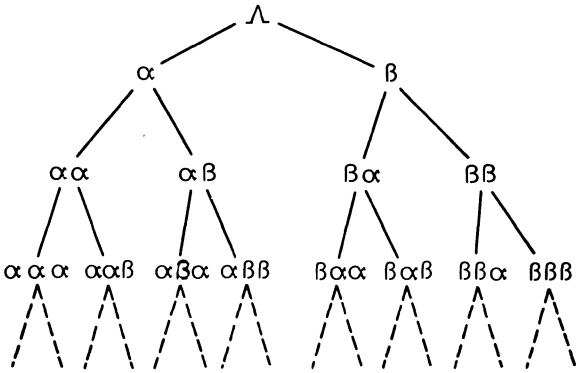


Figure 1

Nous notons T_n le sous-arbre initial de profondeur n.

Soit T_n un tel arbre, notons $L_n(\Sigma'_\infty, \psi)$ le langage *fini* engendré par $\psi(u)$ de la façon suivante : tout symbole φ_m tel que m ne soit pas un nœud de T_n est obligatoirement réécrit en Ω. Notons $W(t)$ l'ensemble

$$W(t) = \{ w_m \mid \varphi_m \text{ a été réécrit pour engendrer } t \text{ et } t \in L_n(\Sigma'_\infty, \psi) \}$$

On démontre (cf. § 3.3) que, T_n étant donné, il est possible de résoudre les éléments de $W(L_n)(= \cup W(t)$ pour t de $L_n(\Sigma'_\infty, \psi))$ c'est-à-dire qu'il est possible à partir des équations définissant les w_m de $W(L_n)$, d'associer à chaque élément de $W(L_n)$ un terme de $M(F, \{ u, v \})$ noté t_m qui permet de calculer w_m pour une interprétation donnée.

Interprétation

Rappelons que l'interprétation I', induite par le programme P', est définie par :

- son domaine $\overline{\mathbf{N}} = \overline{\mathbf{N}} \cup \{ \omega \}$
- pour tout couple (m, n) de $\overline{\mathbf{N}} \times \overline{\mathbf{N}}$ et s de $\{$ vrai, faux $\}$

$$a_I = 1$$
$$k_{I'}(n) = n - 1 \quad \text{si} \quad n > 0$$
$$g_{I'}(n) = \begin{cases} \text{vrai} & \text{si} \quad n = 0 \\ \text{faux} & \text{si} \quad n > 0 \end{cases}$$
$$f_{I'}(s, m, n) = \begin{cases} m & \text{si} \quad s = \text{vrai} \\ n & \text{si} \quad s = \text{faux} \end{cases}$$

n est la valuation qui à u associe l'entier naturel n et une interprétation valuée est un triplet (I', T_p, n). Nous sommes en mesure de donner l'interprétation d'un mot, t, de $L_n(\Sigma'_\infty, \psi)$:

- $(I', T_p, n)(\Omega) = \omega$
- $(I', T_p, n)(u) = n$
- $(I', T_p, n)(t) = \begin{cases} \omega \text{ si il existe } w_m \text{ de } W(t) \text{ tel que } (I', n)(t_m) = \omega \\ f_{I'}((I', T_p, n)(t_1), (I', T_p, n)(t_2), (I', T_p, n)(t_3)) \end{cases}$

avec $t = f(t_1, t_2, t_3)$

Si nous étudions le langage $\mathrm{L}(\Sigma'_\infty, \psi)$ nous constatons que les mots y appartenant forment une chaîne strictement croissante pour l'ordre défini par la taille; d'où $L(\Sigma'_\infty, \psi) = \{ t_p \mid p \in \mathbf{N} \}$. Plus précisément, T_p étant donné, nous avons la relation $L_p(\Sigma'_\infty, \psi) = \{ t_q \mid q \leqslant p + 1 \}$

D'autre part si t_q est un élément de $L_n(\Sigma'_\infty, \psi)$ nous avons les propositions suivantes dont la preuve résulte de manipulations du système Σ'_∞ :

- $\forall r \in \mathbf{N},\ 0 \leqslant r \leqslant q - 1 \rightarrow w_{\alpha^r} \in W(t_q)$ (cf. p. 10)
- $\forall \mathrm{r} \in \mathbf{N},\ 0 \leqslant r \leqslant p \quad t_{\alpha_r}$

$t_{\alpha^r} = f(g(k^{r-1}(u)), a, f(\ldots, f(g(k^{n-1}(u)), a, \Omega), \ldots))$

k^i désigne l'opérateur k appliqué i fois à u si $i \neq 0$ et l'identité si i est nul.

Nous pouvons passer à l'interprétation de Σ'.

Soit (I', T_p, n) une interprétation valuée,

l'interprétation du mot t_q de L_p exige la considération de trois cas :

(i) $n > p \geqslant 1$ $(I', T_p, n)(t_q) = \omega$ pour tout q compris entre 0 et $p + 1$ car

$(I', T_p, n)(t_0) = I', T_p, n)(t_1) = \omega$

et $(I', n)(t_\alpha) = \omega$ or w_α appartient à $W(t_q)$ pour q compris entre 2 et $p + 1$

(ii) $p \geqslant n \geqslant 1$

$$(I', T_p, n)(t_1) = \omega \quad \text{car} \quad n \geqslant 1.$$

De plus on démontre que $w_{\beta^{n-1}\alpha}$ appartient à $W(t_i)$ pour i compris entre 2 et $n + 1$. Or la résolution de $w_{\beta^{n-1}\alpha}$ conduit au mot $f(g(u), a, \Omega)$ qui est tel que

$$(I', n)(f(g(u), a, \Omega)) = \omega \quad \text{si} \quad n \geqslant 1$$

d'où

$$\forall i, \ 2 \leqslant i \leqslant n + 1 \quad (I', T_p, n)(t_i) = \omega$$

en vertu de la définition donnée ci-dessus.

Quant aux mots $t_{n+2}, \ldots, t_{p+1}$ — si ils existent — ils sont tels que w_{α^i} appartienne à $W(t_i)$ pour i variant de $n + 2$ à $p + 1$ et, finalement, nous obtenons le résultat :

$$\forall i, \ 0 \leqslant i \leqslant p + 1 \quad (I', T_p, n)(t_i) = \omega$$

(iii) $n = 0$

Dans ce cas nous avons les égalités

$$(I', 0)(t) = 0$$
$$(I', 0)(t_\alpha) = 1$$
$$(I', 0)(t_{\alpha^i}) = \omega \text{ pour } i \text{ compris entre 2 et } p + 1$$

D'où il vient

$(I', T_p, 0)(t_i) = \omega$ dès que i varie de 2 à $p + 1$ car w_{α^i} appartient à $W(t_i)$

mais

$$(I', T_p, 0)(t_1) = 1 \text{ car } W(t_1) = \{ w \} \text{ et } t_1 = f(g(u), a, \Omega)$$

Enfin la réunion des trois cas permet de conclure que

$$\mathrm{Val}_{I', T_p}(\Sigma'_\infty, \psi)(n) = \begin{cases} 1 & \text{si} \quad n = 0 \\ \omega & \text{sinon} \end{cases}$$

et ceci quelque soit l'arbre T_p c'est-à-dire quelle que soit la profondeur de récursion autorisée.

La fonction calculée par P' vaut 1 au point 0 et elle est non définie partout ailleurs (cette déduction est validée par les résultats du § 3) ce qui est le résultat attendu.

Cette conception de l'appel par la valeur peut sembler délicate d'un point de vue technique mais elle est sans doute préférable à celle consistant en l'introduction – artificielle – d'un prédicat " def " qui est vrai si et seulement si son argument est défini et faux sinon; si bien qu'au programme P′ est associé le programme récursif (cf. [12]) :

psi(n) = phi (n, n)

phi (m, n) = *si* ($m = 0 \wedge$ def (n)) *alors* 1 *sinon* phi ($m - 1$, phi (m, n))

ainsi que nous pouvons le voir dans [13].

2. SÉMANTIQUE D'UN LANGAGE DE PROGRAMMATION

2.1. Description du langage de programmation

Le langage dont nous voulons définir la sémantique, est très voisin d'Algol 60 ou de Pascal. Il contient les instructions élémentaires suivantes :

– affectation

– branchement conditionnel avec les deux formes SI... ALORS... et SI... ALORS... SINON...

– branchement inconditionnel : ALLERA

– itération sous la forme TANTQUE... FAIRE...

Il utilise la même structure faite de blocs, de sous-blocs, d'instructions composées et d'instructions élémentaires. Pour rendre la description, faite plus loin, claire nous supposerons que tout programme écrit dans ce langage vérifie les deux hypothèses suivantes :

(1) les données du programme sont toujours déclarées dans le bloc le plus extérieur du programme;

(2) dans un bloc ne sera déclaré qu'un seul identificateur qui sera immédiatement initialisé par une instruction d'affectation.

2.2. Algorithme de transcription

Transcrire un programme, c'est trouver un système schématique généralisé qui permet de définir la fonction calculée par le programme en question.

La transcription se fait bloc par bloc i.e. à chaque bloc correspondra un symbole de fonction inconnue dont l'arité sera égale au nombre d'identificateurs locaux dans ce bloc plus le nombre de variables globales et dont l'image sera un n-uplet si n est cette arité.

Cette construction entraîne la croissance de l'arité des symboles au fur et à mesure que nous nous "enfonçons" dans la structure du programme au cours de la transcription.

2.2.1. Différents cas doivent être examinés :

(i) B est un bloc, ne contenant aucun sous-bloc, composé de p instructions composées ou simples $\mathscr{I}_1, \mathscr{I}_2, \ldots \mathscr{I}_p$ et X est la liste des identificateurs connus dans ce bloc.

Ce bloc est transcrit par l'équation :

$$\varphi_B(V) = \varphi_p(\varphi_{p-1}(\ldots (\varphi_1(V)) \ldots))$$

où V est la liste des variables associées à X et les $\varphi_1, \ldots, \varphi_p$ sont les transcriptions des instructions $\mathscr{I}_1, \ldots \mathscr{I}_p$.

(ii) B est un bloc composé de la suite $B_1, B_2, \ldots, B_p$ de sous-blocs; dans chaque sous-bloc B_1 est déclaré l'identificateur x_i qui est initialisé dès le début par l'expression notée $E_i(X)$ où X est la liste des identificateurs connus dans B.

φ_B sera défini par l'équation

$$\begin{cases} \varphi_B(V) = \eta_{n+1}^{n+1}(\varphi_p(\eta_{n+1}^{n+1}(\ldots \varphi_2(\eta_{n+1}^{n+1}(\varphi_1(V, w_1)), w_2), \ldots, w_p))) \\ w_i \quad = e_i(V) \end{cases}$$

où V est la liste des variables associées à X, w_i correspond à la variable locale x_i et est défini par l'équation $w_i = e_i(V)$ e_i étant la transcription de $E_i(X)$. Remarquons que φ_i — transcription du bloc B_i — est un symbole d'arité $n + 1$ et calcule un $(n + 1)$-uplet ce qui explique l'emploi du symbole conventionnel η_{n+1}^{n+1}, qui est toujours interprété comme l'application qui à un $(n + 1)$-uplet associe le n-uplet ontenu en " oubliant " le dernier élément du $(n + 1)$-uplet.

(iii) B est un bloc composé de sous-blocs et d'instructions (simples ou composées); dans ce cas il suffit, pour le transcrire, de faire le " panachage " des deux types d'équations ci-dessus.

(iv) $\mathscr{I}$ est une instruction composée faite de la suite des instructions simples $I_1, I_2, \ldots, I_p$ et contenu dans un bloc dans lequel sont contenus n identificateurs. sera transcrit en

$$\varphi_{\mathscr{I}}(V) = \varphi_p(\varphi_{p-1}(\ldots (\varphi_1(V)) \ldots))$$

où V est la liste des variables associées aux n identificateurs et φ_j — j variant de 1 à p — est la transcription de I_j.

Les définitions qui viennent d'être données sont cohérentes et non ambiguës puisqu'elles se fondent implicitement sur l'associativité de la composition des applications partielles, c'est-à-dire des applications partielles associées à chaque symbole de fonction inconnue et à chaque symbole de fonction de base interprété.

Pour parachever la transcription d'un programme il faut maintenant définir la transcription de chaque instruction simple.

2.2.2. Transcription des instructions élémentaires

Nous ne donnerons pas ici la transcription de toutes les instructions élémentaires mais de celles qui sont les plus intéressantes.

(i) I est une instruction d'affectation. Deux cas peuvent se produire :

(1) I est l'instruction qui initialise une variable qui vient d'être déclarée dans un bloc B; d'où la structure du programme :

```
DEBUT
  .
  .
  .
  ⎧DEBUT; NOUVEAU x; x := E(X);
B ⎨        B'
  ⎩FIN
  .
  .
  .
FIN
```

X est la liste des identificateurs connus dans le plus petit bloc contenant B qui existe et qui est noté B_1; d'où le schéma suivant :

```
DEBUT
  .
  .
  .
  ⎧DEBUT
  ⎪      B'1
B1⎨      B
  ⎪      B''1
  ⎩FIN
  .
  .
  .
FIN
```

Soit V la liste des variables associée à X, nous obtenons la transcription suivante en plusieurs équations :

$$\begin{cases} \varphi_{B_1}(V) & = \varphi_{B_1''}(\eta_{n+1}^{n+1}(\varphi_B(\varphi_{B_1'}(V, w)))) \\ \varphi_B(V, v) & = \varphi_{B'}(V, v) \\ w & = e(V) \end{cases}$$

Remarquons que φ_B a une arité égale à $n + 1$ et définit un $(n + 1)$-uplet d'où l'emploi de η_{n+1}^{n+1} et que la liste $V \cup \{ v \}$ correspond aux $n + 1$ identificateurs connus dans B.

(2) I peut être une instruction qui modifie une variable à l'intérieur d'un programme et qui s'écrit :

$$x := E(X)$$

dans ce cas la transcription donne :

$$\varphi_I(v_1, \ldots, v_n) = i_n(v_1, \ldots, e(v_1, \ldots, v_n), \ldots, v_n)$$

où $\{ v_1, \ldots, v_n \}$ correspond à X, v_i à la variable x, $e(v_1, \ldots, v_n)$ est la transcription de E et i_n est le symbole toujours interprété en l'identité.

(ii) I est une instruction d'itération de la forme

TANTQUE $p(X)$ FAIRE B;

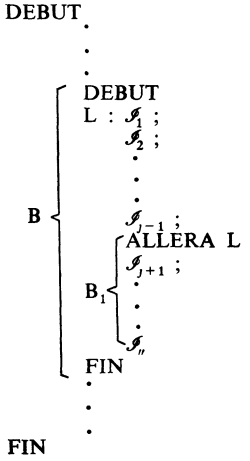
qui se traduit simplement en

$$\varphi_I(V) = C_n(f(V), \varphi_I(\varphi_B(V)), i_n(V))$$

où V est la liste des n variables correspondant à X, f correspond au prédicat p et C_n est un symbole conventionnel qui traduit le conditionnel usuel.

(iii) I est une instruction ALLERA; deux cas sont à distinguer :

(1) le branchement se situe avant l'instruction qu'il vise; d'où la structure du programme



et la transcription

$$\begin{cases} \varphi_B(V) = \varphi_{B_L}(\varphi_{j-1}(\ldots \varphi_2(\varphi_1(V))\ldots)) \\ \varphi_{B_L}(V) = \varphi_n(\ldots \varphi_2(\varphi_1(V))\ldots) \end{cases}$$

(2) Le branchement se situe après; d'où la transcription avec des notations identiques aux précédentes

$$\varphi_{B_L}(V) = \varphi_n(\ldots \varphi_j(V)\ldots)$$

(iv) I est un appel de procédure. Alors cette instruction est transcrite en $\varphi_P(V)$ où φ_P est la transcription du corps de la procédure P appelée et V la liste des arguments. (Cf. exemple 2, § 1).

2.3. Fonction calculée par un programme P

La transcription que nous venons de décrire met en évidence un ensemble F de symboles de fonction de base.

Par ailleurs les déclarations des variables définissent un domaine de données noté D_P. De plus, chaque élément de F correspond à une application dont l'ensemble est noté F_P. Avec D_P et F_P une interprétation, I_P, est définie.

L'algorithme de transcription a produit, à partir du programme P, un système schématique généralisé noté Σ_P. Si bien qu'au programme P nous sommes capables d'associer le couple (Σ_P, I_P).

Par définition, nous posons que la fonction calculée par le programme P est la fonction définie par le système Σ_P sous l'interprétation I_P ce que nous notons $\mathrm{Val}_{I_P} \Sigma_P$. Le sens de $\mathrm{Val}_{I_P} \Sigma_P$ va être définie dans le paragraphe suivant.

Enfin les deux exemples du 4 démontreront le mécanisme de la transcription.

3. SYSTÈMES SCHÉMATIQUES GÉNÉRALISÉS

3.1. Définition des S.S.G.

Un système schématique généralisé (noté S.S.G.) est un ensemble fini d'équations sur le magma $M(F \cup \Phi, V \cup W)$ où :

F est l'ensemble des symboles de fonctions de base.
Φ est l'ensemble des symboles de fonctions inconnues.
V l'ensemble des variables.
W l'ensemble des variables locales
qui s'écrit ainsi

$$\begin{cases} \varphi^i(v_1, \ldots, v_{n_i}) = \tau^i \\ w^{i,j} = \tau^{i,j} \\ i \in \{ 1, 2, \ldots, N \}, \quad j \in \{ 1, \ldots, \sigma_i \} \end{cases}$$

τ^i appartenant au magma $M(F \cup \Phi, \{ v_1, \ldots, v_{n_i} \} \cup \{ w^{i,1}, \ldots, w^{i,\sigma_i} \})$ et $\tau^{i,j}$ à $M(F \cup \Phi, \{ v_1, \ldots, v_{n_i} \})$. Le rôle particulier joué par les éléments de W est évident et leur dénomination de « variables locales » se fait par analogie avec ce qui se passe dans un programme ALGOL l'équation $w^{i,j} = \tau^{i,j}$ représentant « l'initialisation » de cette variable.

La sémantique des S.S.G. se construit en trois étapes.

3.2. Construction d'un système infini simulant un S.S.G.

Cette construction est fondée sur le principe suivant : soit g l'application de $\mathbf{Z}$ dans $\mathbf{Z}$ définie par

$$\forall x \in \mathbf{Z} \quad g(x) = \textit{si } x = 0 \textit{ alors } 1 \textit{ sinon } g(x + 1)$$

g peut être considérée, pour une interprétation précise, comme étant calculée par le système *infini* suivant noté Σ^∞

$$\Sigma^\infty \begin{cases} \varphi(u) = f(p(u), a, \varphi^1(s(u))) \\ \varphi^1(u) = f(p(u), a, \varphi^2(s(u))) \\ \\ \varphi^n(u) = f(p(u), a, \varphi^{n+1}(s(u))) \end{cases}$$

Σ^∞ n'est que la concrétisation de toutes les réécritures possibles et chaque réécriture est différenciée des autres par l'indexation du symbole φ.

De plus Σ^∞ peut être écrit sous une forme, légèrement différente, mais plus concise

$$\begin{cases} \varphi_m(u) = f(p(u), a, \varphi_m(s(u)) \\ m \in \{ \alpha \}^* \end{cases}$$

en utilisant un symbole spécial α.

C'est dans l'esprit de cet exemple que nous avons construit un algorithme [8, 9] permettant d'associer à tout S.S.G., Σ, un système infini, $\Sigma_{\alpha,\beta}$, qui le simule.

Utilisant A et B, deux ensembles de lettres, pour indexer les symboles de fonction inconnue et les variables locales qu'ils introduisent, $\Sigma_{\alpha,\beta}$ est de la forme :

$$\Sigma_{\alpha,\beta} \begin{cases} \varphi^i_m(v_1, \ldots, v_{n_i}) = \tau^i_m \quad ; \quad \tau^i_m \in M(F \cup \Phi_{\alpha,\beta}, \{ v_1, \ldots, v_{n_i} \} \cup W_{\alpha,\beta}) \\ w^{i,j}_m = \tau^{i,j}_m \\ m \in (A \cup B)^* \end{cases}$$

où $\Phi_{\alpha,\beta} = \{ \varphi^i_m \mid \varphi^i \in \Phi \text{ et } m \in (A \cup B)^* \}$ et $W_{\alpha,\beta}$
$= \{ w^{i,j}_m \mid w^{i,j} \in W \text{ et } m \in (A \cup B)^* \}$

Nous noterons Σ_∞ le système infini qui simule Σ sans tenir compte du rôle particulier des variables locales et qui s'écrit

$$\Sigma_\alpha \begin{cases} \varphi^i_m(v_1, \ldots, v_{n_i}) = \tau^i_m \\ m \in A^* \end{cases}$$

$\Sigma_{\alpha,\beta}$ (resp. Σ_α) est, bien sûr, un système de réécriture sur le magma $M(F, V \cup W_{\alpha,\beta})$ (resp. $M(F, V \cup W_\alpha)$) engendrant le langage $L(\Sigma_{\alpha,\beta}, \varphi^1)$(resp. $L(\Sigma_\alpha, \varphi^1)$).

3.3. « Résolution » des variables locales

La valeur de chaque variable locale est obtenue, au moment de l'interprétation, par l'intermédiaire de mots de $M(F, V)$; connaître ces expressions, c'est « résoudre » les variables locales.

A chaque variable locale, $w^{i,j}$, nous associons l'ensemble des mots-index de $(A \cup B)^*$ qu'il se voit attribuer par le « fonctionnement » de $\Sigma_{\alpha,\beta}$ comme système de réécriture. Cet ensemble, noté Θ^i_j, est un arbre fini ou non; par suite à chaque S.S.G. est associée une famille d'arbres dits « arbres de variables locales ».

Soit $\mathscr{F}$ l'ensemble $\{ T^i_j \mid i \in \{ 1, \ldots, N \}, j \in \{ 1, \ldots, \sigma_i \} \}$ tel que chaque T^i_j soit un sous-arbre initial *fini* de Θ^i_j. A $\mathscr{F}$ nous associons une dérivation qui est celle du système de réécriture $\Sigma_{\alpha,\beta}$ avec une contrainte supplémentaire : tout symbole de fonction inconnue indexé par un élément de $(A \cup B)^*$ qui n'est pas un nœud de T^i_j, pour quelque i et quelque j, est réécrit en Ω. Nous notons $L_{\mathscr{F}}(\Sigma_\alpha, \varphi^1)$ le langage ainsi engendré. Nous avons alors le résultat, technique, mais très important suivant :

PROPOSITION : *L'ensemble $\mathscr{F}$ étant donné, il est toujours possible d'associer à chaque variable locale, dont l'index est un nœud d'un élément de $\mathscr{F}$, un langage du magma $M(F, V)$.*

La démonstration se trouve dans [9].

3.4. Interprétation et sémantique

Soit Σ, le système associé au S.S.G. Σ_α, une interprétation discrète de Σ, sur le magma $M(F, V)$ est donnée par :

- un domaine discret D_I muni d'un plus petit élément
- pour chaque symbole f de F une application f_I de $D_I^{a(f)}$ dans D_I
- et un ensemble $\mathscr{F}$ de sous-arbres initiaux finis des « arbres de variables locales ».

Une telle interprétation est notée $(I, \mathscr{F})$. ν étant une valuation (i.e. une application de V dans D_I), nous définissons une application $(I, \mathscr{F}, \nu)$ de $L_{\mathscr{F}}(\Sigma_\alpha, \varphi^1)$ dans D_I par $- (I, \mathscr{F}, \nu)(\Omega) = \omega$; $(I, \mathscr{F}, \nu)(v) = \nu(v)$;

$$- (I, \mathscr{F}, \nu)(f(t_1, \ldots, t_{a(f)})) = \begin{cases} \omega \text{ si dans quelque } t_i \text{ il existe une variable} \\ \quad \text{locale, } w \text{ par exemple, d'index dans } \mathscr{F} \text{ telle} \\ \quad \text{que val}(w) = \omega \\ f_I((I, \mathscr{F}, \nu)(t_1), \ldots, (I, \mathscr{F}, \nu)(t_{a(f)})) \end{cases}$$

où val (w) est l'élément de D_I ainsi obtenu : w ayant un index qui est un nœud d'un élément de $\mathscr{F}$, est soluble d'après la proposition du § 3 . 3; il lui est donc associé un langage de $M(F, V)$ qui, I et ν étant données, définit un élément de D_I que nous notons, justement, val (w).

Le théorème crucial, pour la définition d'une sémantique des S.S.G. est alors atteint :

THÉORÈME : *Soit $(I, \mathscr{F})$ une interprétation de Σ_α, si t et t' sont deux mots de $L_{\mathscr{F}}(\Sigma_\alpha, \varphi^1)$ tels que $(I, \mathscr{F}, \nu)(t)$ et $(I, \mathscr{F}, \nu)(t')$ soient tous deux distincts de ω alors $(I, \mathscr{F}, \nu)(t)$ est égal à $(I, \mathscr{F}, \nu)(t')$.*

La démonstration de ce théorème est tout à fait semblable à celle du théorème correspondant dans [15]. De ce résultat découle immédiatement la définition de la sémantique de Σ_α.

Une interprétation $(I, \mathcal{F})$ étant donnée, l'application $\mathrm{Val}_{I,\mathcal{F}}(\Sigma_\alpha, \varphi^1)$ de $D_I^{n_1}$ dans D_I est définie, pour toute valuation ν, par :

$$\mathrm{Val}_{I,\mathcal{F}}(\Sigma_\alpha, \varphi^1)(\nu(v_1), \ldots, \nu(v_{n_1})) = \begin{cases} d \text{ si il existe } t \text{ de } L_{\mathcal{F}}(\Sigma_\alpha, \varphi^1) \\ \text{tel que } (I, \mathcal{F}, \nu)(t) = d \\ \omega \text{ sinon} \end{cases}$$

3.5. Sémantique des S.S.G.

Définir une sémantique d'un système schématique généralisé c'est définir l'application calculée par le système pour une interprétation donnée. Il nous faut donc dire ce qu'est une interprétation d'un S.S.G. Σ et ici une alternative s'offre à nous :

– soit définir les interprétations des S.S.G. comme au paragraphe précédent et nous posons tout simplement

$$\mathrm{Val}_{I,\mathcal{F}}(\Sigma, \varphi^1) = \mathrm{Val}_{I,\mathcal{F}}(\Sigma_\alpha, \varphi^1)$$

Cette sémantique dite « concrète », tient compte, en quelque sorte, des capacités physiques de la machine puisque nous limitons le nombre de réécritures possibles.

– soit prendre pour interprétation des S.S.G. les interprétations discrètes définies au § 1.3. Dans ce cas l'application associée au S.S.G. Σ, I étant une interprétation donnée, et définie pour toute valuation ν par

$$\mathrm{Val}_I(\Sigma, \varphi^1)(\nu(v_1), \ldots, \nu(v_{n_1})) = \begin{cases} d & \text{si il existe un ensemble, } \mathcal{F}, \text{ de sous-arbres} \\ & \text{initiaux tel que } \mathrm{Val}_{I,\mathcal{F}}(\Sigma_\alpha, \varphi^1)(\nu) = d \\ \omega & \text{sinon} \end{cases}$$

La cohérence de cette définition est assurée par la proposition suivante :

PROPOSITION : *Soient $\mathcal{F}$ et $\mathcal{F}'$ deux ensembles de sous-arbres initiaux finis des « arbres de variable locale » tels que chaque élément de $\mathcal{F}$ d'indice (i, j) soit un sous-arbre de l'élément correspondant de $\mathcal{F}'$, alors l'application* $\mathrm{Val}_{I,\mathcal{F}}(\Sigma_\alpha, \varphi^1)$ *est moins définie que l'application* $\mathrm{Val}_{I,\mathcal{F}'}(\Sigma_\alpha, \varphi^1)$ *pour tout I.*

C'est-à-dire que si d est un élément de $D_I^{n_1}$ alors la proposition suivante est vraie

$$\mathrm{Val}_{I,\mathcal{F}}(\Sigma_\alpha, \varphi^1)(d) \neq \omega \Rightarrow \mathrm{Val}_{I,\mathcal{F}'}(\Sigma_\alpha, \varphi^1)(d) = \mathrm{Val}_{I,\mathcal{F}}(\Sigma_\alpha, \varphi^1)(d)$$

La démonstration est laissée au lecteur.

4. EXEMPLES

Ces deux exemples vont nous permettre d'utiliser la construction décrite aux paragraphes précédents. Ils consistent en deux programmes qui calculent la racine carrée entière d'un entier naturel par deux méthodes différentes.

Nous allons définir la fonction calculée par chacun des programmes et constater leur identité.

4.1. Soit P_2 le programme suivant :

```
DEBUT ENTIER n; Lire (n);
   DEBUT ENTIER x; x := 0;
      DEBUT ENTIER y; y := 1;
         DEBUT ENTIER z; z := 1;
            TANTQUE y ⩽ n FAIRE DEBUT
                                    x := x + 1;
                                    z := z + 2;
                                    y := y + z;
                                  FIN;
         FIN;
      FIN;
      n := x;
   FIN;
   Ecrire (n);
FIN de P1
```

Par transcription il lui est associé le S.S.G. Σ_1

$$\Sigma_1 \begin{cases} \varphi^0(v_1) = \pi_2^2(\varphi^1(v_1, w^0)) \\ \varphi^1(v_1\ v_2) = \eta_3^3(\varphi^2(v_1, v_2, w^1)) \\ \varphi^2(v_1, v_2, v_3) = \eta_4^4(\varphi^3(v_1, v_2, v_3, w^2)) \\ \varphi^3(v_1, v_2, v_3, v_4) = C_4(p(v_1, v_3), \varphi^3(\varphi^4(v_1, v_2, v_3, v_4)), i_4(v_1, v_2, v_3, v_4)) \\ \varphi^4(v_1, v_2, v_3, v_4) = \varphi^7(\varphi^6(\varphi^5(v_1, v_2, v_3, v_4))) \\ \varphi^5(v_1, v_2, v_3, v_4) = i_4(v_1, f(v_2), v_3, v_4) \\ \varphi^6(v_1, v_2, v_3, v_4) = i_4(v_1, v_2, v_3, g(v_4)) \\ \varphi^7(v_1, v_2, v_3, v_4) = i_4(v_1, v_2, s(v_3, v_4), v_4) \\ w^0 = a \\ w^1 = b \\ w^2 = b \end{cases}$$

Σ_1 peut être aménagé de façon analogue à une grammaire et condensé en un nouveau système

$$\Sigma_1 \begin{cases} \varphi^0(v_1) = \pi_2^2(\varphi^1(v_1, w^0)) \\ \varphi^1(v_1, v_2) = \eta_3^3(\eta_4^4(\varphi^2(v_1, v_2, w^1, w^2))) \\ \varphi^2(v_1, v_2, v_3, v_4) = C_4(p(v_1, v_3), \varphi^2(v_1 f(v_2), s(v_3, g(v_4)), g(v_4)), i_4(v_1, v_2, v_3, v_4)) \\ w^0 = a \\ w^1 = b \\ w^2 = b \end{cases}$$

qui est encore noté Σ_1 et en utilisant les symboles φ^0, φ^1 et φ^2 bien qu'ils ne soient plus définis par les mêmes équations.

Le système $\Sigma_{1_{\alpha,\beta}}$, simulant dans la construction du § 3 le système Σ_1, est ici identique à Σ_1 puisque le seul symbole de fonction inconnue récursif n'introduit pas de variable locale. Les « arbres de variables locales », notés Θ_0, Θ_1 et Θ_2, sont égaux à l'arbre réduit à sa seule racine; donc tout ensemble de sous-arbres initiaux est égal à $\{ \Theta_0, \Theta_1, \Theta_2 \}$. Cet ensemble sera noté Θ.

D'après les résultats généraux du § 3 nous avons dans ce cas particulier

$$\mathrm{Val}_{I_1}(\Sigma_1, \varphi^0) = \mathrm{Val}_{I_1,\Theta}(\Sigma_1, \varphi^0)$$

L'interprétation I_1 est définie :

– par son domaine $\overline{\mathbf{N}} = \overline{\mathbf{N}} \cup \{ \omega \}$

– par les applications définies pour tout couple (m, n) de $\overline{\mathbf{N}} \times \overline{\mathbf{N}}$:

$$\begin{aligned} a_{I_1} &= 0 \\ b_{I_1} &= 1 \\ f_{I_1}(n) &= n + 1 \\ g_{I_1}(n) &= n + 2 \\ s_{I_1}(m, n) &= m + n \\ p_{I_1}(m, n) &= \begin{cases} \text{vrai si } m \leqslant n \\ \text{faux si } m > n \end{cases} \end{aligned}$$

– les symboles $i_4, C_4, \pi_2^2, \eta_3^3$ et η_4^4 étant interprétés de façon conventionnelle (cf. § 2.2).

Examinons le langage $L(\Sigma_1, \varphi^0)$ qui, outre les mots Ω, $\pi_2^2(\Omega)$, $\pi_2^2(\eta_3^3(\eta_4^4(\Omega)))$, contient la suite $\{ t_n \mid n \in \mathbf{N} \}$, t_n désignant le mot

$$\pi_2^2(\eta_3^3(\eta_4^4(C_4(p(v_1, w^1), C_4(\ldots, C_4(p(v, t_n^3), \Omega, i_4(v_1, t_n^2, t_n^3, t_n^4)) \ldots)))))$$

où $t_n^2 = f^n(w^0)$

$t_n^3 = s(s(\ldots s(w, g(w^2)), \ldots, g^{n-1}(w^2)), g^n(w^2))$

$t_n^4 = g^n(w^2)$

Soit (I_1, Θ, m) une interprétation valuée; il est clair que

$$\mathrm{val}\ (w^0) = 0$$

et

$$\mathrm{val}\ (w^1) = \mathrm{val}\ (w^2) = 1$$

et, par conséquent, que les variables locales ne soulèvent pas de problèmes. D'où les égalités :

$$(I_1, \Theta, m)(\Omega) = (I_1, \Theta, m)(\pi_2^2(\Omega)) = (I_1, \Theta, m)(\pi_2^2(\eta_3^3(\eta_4^4(\Omega)))) = \omega$$
$$(I_1, \Theta, m)(t_n^2) = n$$

$$(I_1, \Theta, m)(t_n^3) = 1 + 3 + \ldots + (2n + 1)$$
$$(I_1, \Theta, m)(t_n^4) = 2n + 1$$

et, finalement

$$(I_1, \Theta, m)(t_n) = \begin{cases} p \text{ si et seulement si } n \geqslant p \text{ et} \\ \quad 1 + 3 + \ldots + (2p - 1) \leqslant m < 1 + 3 + \ldots + (2p + 1) \\ w \text{ sinon} \end{cases}$$

Par définition de $\mathrm{Val}_{I_1}(\Sigma_1, \varphi^0)$ nous avons

$$\mathrm{Val}_{I_1}(\Sigma_1, \varphi^0)(m) = \text{racine carrée entière de } m$$

en vertu de la propriété arithmétique

$$\forall p \in \mathbf{N} \quad p \geqslant 1 \Rightarrow p^2 = 1 + 3 + \ldots + (2p - 1)$$

4.2. Soit maintenant le programme P_2

```
DEBUT ENTIER n; Lire (n);
  DEBUT ENTIER x; x := 0;
    DEBUT ENTIER y; y := 1;
      DEBUT ENTIER z; z := 1;
        E : Si y ⩽ n ALORS DEBUT
                              x := x + 1;
                              z := z + 1;
                              DEBUT ENTIER t;
                                t := z + 1;
                                F : SI t > 0 ALORS DEBUT
                                                     y := y + 1;
                                                     t := t - 1;
                                                     ALLERA F;
                                                   FIN;
                                z := z + 1;
                                ALLERA E;
                              FIN;
                            FIN;
      FIN;
    FIN;
    n := x;
  FIN;
  Ecrire (n);
FIN de P2
```

Par transcription nous associons à P_2 le S.S.G. Σ_2

$$\Sigma_2 \begin{cases} \varphi^0(v_1) = \pi_2^2(\varphi^1(v_1, w^0)) \\ \varphi^1(v_1, v_2) = \eta_3^3(\varphi^2(v_1, v_2, w^1)) \\ \varphi^2(v_1, v_2, v_3) = \eta_4^4(\varphi^3(v_1, v_2, v_3, w^2)) \\ \varphi^3(v_1, v_2, v_3, v_4) = C_4[p(v_1, v_3), \eta_5^5(\varphi^4(v_1, f(v_2), v_3, f(v_4), w^3)), i_4(v_1, v_2, v_3, v_4)] \\ \varphi^4(v_1, v_2, v_3, v_4, v_5) = C_5[r(v_5), \varphi^4(v_1, v_2, f(v_3), v_4, h(v_5)), i_5(\varphi_3(v_1, v_2, v_3, f(v_4)), v_5)] \\ w^0 = a \\ w^1 = b \\ w^2 = b \\ w^3 = s(v_4) \end{cases}$$

Dans le cas présent le système $\Sigma_{2_{\alpha,\beta}}$ est distinct de Σ_2 et s'écrit sous la forme :

$$\Sigma_{2_{\alpha,\beta}} \begin{cases} \varphi^0(v_1) = \pi_2^2(\varphi^1(v_1, w^0)) \\ \varphi^1(v_1, v_2) = \eta_3^3(\varphi^2(v_3, v_2, w^1)) \\ \varphi^2(v_1, v_2, v_3) = \eta_4^4(\varphi^3(v_1, v_2, v_3, w^2)) \\ \varphi_m^3(v_1, v_2, v_3, v_4) = C_4[p(v_1, v_3), \eta_5^5(\varphi_m^4(v_1, f(v_2), v_3, f(v_4), w_m^3)), i_4(v_1, v_2, v_3, v_4)] \\ \varphi_m^4(v_1, v_2, v_3, v_4, v_5) = C_5[r(v_5), \varphi_{m_\alpha}^4(v_1, v_2, f(v_3), v_4, h(v_5)), i_5(\varphi_{m_\beta}^3(v_1, v_2, v_3, f(v_4)), v_5)] \\ w^0 = a \\ w^1 = b \\ w^2 = b \\ w_m^3 = s(v_4) \\ m \in \{ \alpha, \beta \}^* \end{cases}$$

Ce système utilise les deux lettres auxiliaires α et β pour indexer les deux symboles de fonction inconnue qui introduisent la variable locale w^3. Il y a donc quatre « arbres de variables locales » qui sont notés $\Theta_0, \Theta_1, \Theta_2, \Theta_3$ et égaux à :

- l'arbre réduit à sa racine pour les trois premiers
- l'arbre binaire infini complet pour Θ_3, soit

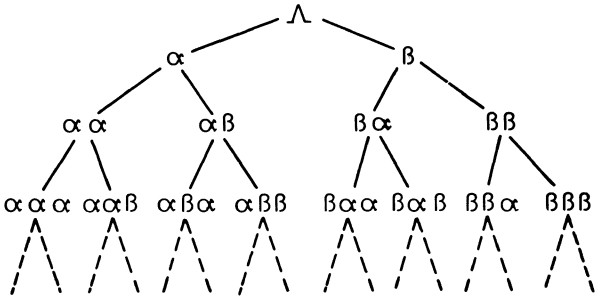


Fig. 2

Pour interpréter $\Sigma_{2_{\alpha,\beta}}$ nous avons besoin de sous-arbres initiaux finis (cf. § 3.3).

Notons $\mathscr{F}_n$ l'ensemble $\{ \Theta_0, \Theta_1, \Theta_2, T_n \}$ où T_n est l'arbre binaire complet de profondeur n. La considération des $\mathscr{F}_n (n \geqslant 0)$ suffit en vertu de la proposition 3.5 pour définir la sémantique de Σ_2.

Soit I_2 l'interprétation induite par le programme P_2 qui est définie par :

- son domaine $\overline{\mathbf{N}} = \mathbf{N} \cup \{ \omega \}$
- les applications suivantes définies pour (m, n) de $\overline{\mathbf{N}} \times \overline{\mathbf{N}}$

$$a_{I_2} = 0$$
$$b_{I_2} = 1$$

$$f_{I_2}(n) = n + 1$$
$$h_{I_2}(n) = n - 1$$
$$r_{I_2}(n) = \begin{cases} \text{vrai si } n > 0 \\ \text{faux si } n = 0 \end{cases}$$
$$p_{I_2}(m, n) = \begin{cases} \text{vrai si } m \leqslant n \\ \text{faux sinon} \end{cases}$$

– les symboles $\pi_2^2, \eta_3^3, \eta_4^4, \eta_5^5, C_4$ et C_5 sont interprétés comme il est décrit au § 2.2.

Soit $(I_2, \mathscr{F}_n, m)$ une interprétation valuée, nous pouvons interpréter les mots de $L_{\mathscr{F}_n}(\Sigma_{2_{\alpha,\beta}}, \varphi^0)$ en vertu du théorème 3.4 et définir l'application $\mathrm{Val}_{I_2,\mathscr{F}_n}(\Sigma_{2_{\alpha,\beta}}, \varphi^0)$ par

$$\mathrm{Val}_{I_2,\mathscr{F}_n}(\Sigma_{2_{\alpha,\beta}}, \varphi^0)(m) = \begin{cases} \text{racine carrée entière de } m \text{ si } m \leqslant n \\ \omega \text{ sinon} \end{cases}$$

Si bien que l'application calculée par Σ_2 sous l'interprétation I_2 est égale à

$$\mathrm{Val}_{I_2}(\Sigma_2, \varphi^0)(m) = \text{racine carrée entière de } m$$

car, par construction, $\mathrm{Val}_{I_2}(\Sigma_2, \varphi^0)(m) = q\ (\in \mathbf{N})$ si et seulement si il existe un ensemble $\mathscr{F}_n$ (pour quelque entier n) tel que $\mathrm{Val}_{I_2,\mathscr{F}_n}(\Sigma_{2_{\alpha,\beta}}, \varphi^0)(m) = q$. Or, ici, il suffit de prendre n égal à m pour être assuré que $\mathrm{Val}_{I_2,\mathscr{F}_n}(\Sigma_{2_{\alpha,\beta}}, \varphi^0)(m)$ = racine carrée entière de m.

Finalement nous avons montré que

$$\mathrm{Val}_{I_1}(\Sigma_1) = \mathrm{Val}_{I_2}(\Sigma_2)$$

ce qui est bien ce que nous attendions de la sémantique que nous avons définie.

BIBLIOGRAPHIE

1. J. J. ARSAC, *Nouvelles leçons de programmation*, Publication de l'I.P., n° 75-29, Université Paris VI, Paris, 1975.
2. R. BURSTALL and J. DARLINGTON, *A Transformation System for Developing Recursive Programs*, Journal of Ass. Comp. Mach., Vol. 24, 1977, pp. 44-67.
3. G. BERRY et B. COURCELLE, *Program Equivalence and Canonical Forms in Stable Discrete Interpretations*, 3° Colloque International, *Automata, Languages and Programming*, S. Michaelson, R. Milner Eds., pp. 168-188, Edinburgh University Press, 1976.
4. G. COUSINEAU, *Les arbres à feuilles indicées : un cadre algébrique pour l'étude des structures de contrôle*, Thèse d'État, Université Paris-VII, Paris, 1977.
5. I. GUESSARIAN, *Semantic Equivalence of Program Schemes and its Syntactic Characterization*, 3° Colloque International, *Automata, Languages and Programming*, S. Michaelson, R. Milner Eds., Edinburgh University Press, 1976, pp. 189-200.

6. I. GUESSARIAN, *Les tests et leur caractérisation syntaxique*, R.A.I.R.O. série Informatique Théorique n° 2, 1977, pp. 133-156.
7. I. IANOV, *The Logical Schemes of Algorithms*, in Problems of Cybernetics, London, Pergamon Press, 1960, pp. 82-140.
8. L. KOTT, *Systèmes schématiques généralisés*, Theoretical Computer Science, 3rd GI Conference, Darmstadt, Lecture Notes in Computer Science 48, H. Tzschach, H. Waldschmidt, H. K.-G. Walter Eds., Springer-Verlag, 1977, pp. 184-189.
9. L. KOTT, *Approche par le magma d'un langage de programmation type Algol : sémantique et vérification de programme*, Thèse 3° cycle, Université Paris-VII, Paris, 1976.
10. L. KOTT, *Sémantique algébrique et principe d'induction : l'induction de Kleene*, soumis à publication, 1977.
11. J. MCCARTHY, *A Basis for a Mathematical Theory of Computation*, in Computer Programming and Formal Systems, P. Braffort, D. Hirschberg Eds., North-Holland, Amsterdam, 1963, pp. 33-70.
12. Z. MANNA, *Mathematical Theory of Computation*, McGraw-Hill, New York, 1974.
13. Z. MANNA and J. VUILLEMIN, *Fixpoint Approach to the Theory of Computation*, Comm. of Ass. Comp. Mach. 15, 1972, pp. 528-536.
14. M. NIVAT, *Sur l'interprétation des schémas de programme monadiques*, Rapport IRIA-Laboria n° 1, 1972.
15. M. NIVAT, *On the Interpretation of Recursive Polyadic Schemes*, Instituto Nazionale di Alta Matematica, Symposia Mathematica, Volume XV, 1974, pp. 251-281.
16. D. PARK, *Fixpoint Induction and Proof of Program Properties*, *Machine Intelligence 5*, B. Meltzer, D. Michie Eds., 1969, pp. 59-78.
17. D. SCOTT and C. STRACHEY, *Towards a Mathematical Semantics for Computer Languages*, Technical Memo PRC-G, Oxford University, Oxford, 1970.