RAIRO. INFORMATIQUE THÉORIQUE

BERNARD ROBINET
FRANÇOIS NOZICK

Sémantique des structures de contrôle

RAIRO. Informatique théorique, tome 11, n° 1 (1977), p. 63-74
<http://www.numdam.org/item?id=ITA_1977__11_1_63_0>

R.A.I.R.O. Informatique théorique/Theorical Computer Science
(vol. 11, n° 1, 1977, p. 63 à 74)

SÉMANTIQUE DES STRUCTURES DE CONTRÔLE (*)

par Bernard ROBINET et François NOZICK (1)

Communiqué par J.-F. PERROT

Résumé. — *On présente les structures de contrôle du surlangage EXEL, leur sémantique étant décrite informellement. On définit un interprète, traduisant EXEL en organigrammes construits à partir des diagrammes de Böhm et Jacopini. On donne une preuve de la correction de cet interprète en vérifiant que la définition décrite dans l'étape précédente est une formalisation convenable de la description informelle.*

INTRODUCTION

Les dernières années ont été fertiles en discussions et en propositions sur les avantages du bannissement de l'instruction de contrôle GO TO [7]; l'emploi de cette instruction rend difficile la lecture des programmes et, par là, leur compréhension : c'est pourquoi l'usage d'instructions de contrôle WHILE est généralement jugé préférable, même au prix de l'introduction de variables booléennes supplémentaires, car il permet de trouver des méthodes assez simples de preuves, d'équivalence ou de terminaison de programmes [3].

Cette suppression du GO TO a conduit divers auteurs tels G. Bochman [4] ou C. T. Zahn [20], à définir de nouvelles structures de contrôle et même, pour certains, à proposer de nouveaux langages de programmation comme Bliss [19] ou Pascal [18].

Dernièrement, une équipe composée d'universitaires et d'ingénieurs a présenté le système EXEL [2] où une distinction totale est faite entre structures de contrôle et instructions de base ou élémentaires.

Suivant les choix exprimés pour celles-ci, on peut parler de EXEL-Fortran ou de EXEL-APL, etc. [14], ce qui autorise à qualifier EXEL de surlangage.

EXEL est construit par composition sur les trois structures de contrôle que sont l'alternative, l'itération et la sortie de niveau quelconque, généra-

(*) Reçu octobre 1976.

(1) Université Pierre-et-Marie-Curie, Institut de Programmation, Paris.

lisation de la notion de sortie de premier niveau introduite indépendamment par Floyd et Knuth [10] et par Kott [11]. Cette instruction permet, à l'exécution, de sortir des n itérations englobantes, ce nombre n étant explicité dans l'instruction : c'est là que réside une des richesses de EXEL car pareille instruction évite d'introduire dans un programme des variables auxiliaires toujours nuisibles à la clarté du discours.

A des fins de traduction ou de preuve, il faut définir de façon précise la signification d'un programme EXEL ou, pour être plus exact, définir la sémantique de ce surlangage. Ne prenant pas en compte les instructions élémentaires, nous donnons dans le présent article une sémantique des structures de contrôle du langage EXEL.

Reste à préciser ce que l'on entend par sémantique. Étant donné un langage L, un système formel S et une application σ de L dans S, on appelle sémantique de L l'application σ. Suivant la nature du système formel et la finalité de cette définition, il est d'usage de qualifier la sémantique d'opératoire ou de mathématique : elle sera opératoire si l'on souhaite obtenir une traduction automatique de tout programme [12] et elle sera mathématique si l'on désire mettre en œuvre des techniques d'équivalence ou de preuve [17].

La sémantique mathématique du surlangage EXEL a déjà été définie en termes de λ-calcul par Nolin et Ruggiu [14]. Ce que nous proposons ici est une définition de la sémantique opératoire de EXEL ; pour ce faire, le système formel choisi est le système de diagrammes de Böhm et Jacopini [5].

Deux raisons motivent ce choix : la première réside en l'existence d'un théorème exprimant la possibilité de représenter tout organigramme à l'aide de trois diagrammes de base et d'une pile booléenne accompagnée de ses opérateurs usuels ; la seconde est que la sémantique des organigrammes est bien connue, qu'elle soit intuitive de par notre pratique de Fortran ou qu'elle soit plus formelle, définie en termes de prédicats de Floyd [8].

La méthode de Böhm et Jacopini a été critiquée, notamment par Cooper [6] et, à sa suite, par Floyd et Knuth [10] puis Ashcroft et Manna [3], principalement pour deux raisons : d'une part elle ne conservait pas la « topologie » de l'organigramme initial, d'autre part, pour un organigramme donné, on peut trouver une borne supérieure de la longueur de la pile booléenne et il suffit donc d'utiliser un vecteur. Ces deux arguments sont aisément réfutables dans le contexte présent : la première raison est qu'un programme écrit en EXEL ne possède pas de topologie ; la seconde raison est que, pour définir la sémantique opératoire, nous construisons la traduction d'un programme EXEL *quelconque* et ne pouvons donc donner une borne supérieure *a priori* ; enfin, la pile est une structure de donnée qui se modélise bien et dont on connaît fort bien les propriétés [16].

Lorsque la sémantique d'un langage est définie en termes d'un autre langage, il est d'usage de la dénommer fonction d'interprétation et sa

réalisation d'interprète. L'interprète que nous proposons pour EXEL ne fournit surement pas une description optimalisée d'un programme EXEL mais le résultat qu'il engendre peut être très facilement amélioré du point de vue du temps de calcul ou de l'encombrement-mémoire, par exemple grâce aux transformations syntaxiques d'Arsac [2].

D'autre part, nous savons traduire les diagrammes de Böhm et Jacopini, ainsi que les manipulations de la pile booléenne, en λ-calcul. Par composition de la fonction d'interprétation que nous proposons et de la sémantique du λ-calcul, qui est définie de manière simple et rigoureuse, nous pouvons alors retrouver la sémantique mathématique de Nolin et Ruggiu et unifier ainsi les deux familles de sémantique : ceci fera l'objet d'un prochain article [15].

Remarquons, enfin, que la démarche proposée ici dépasse le cadre strict du surlangage EXEL et est applicable à la définition de la sémantique de toutes les structures de contrôle actuellement utilisées dans les langages de programmation.

1. LE SURLANGAGE EXEL

1.1. Syntaxe

1.1.1. *Grammaire*

Soient :

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

le vocabulaire terminal, $V_A = (E, \Psi)$ le vocabulaire auxiliaire et E l'axiome; le surlangage EXEL est engendré par la grammaire suivante :

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

1.1.2. *Profondeur d'une expression*

Soit E_1 une expression EXEL bien formée et E_2 une sous-expression propre de E_1.

Soit n_0 (resp. n_F) le nombre d'accolades ouvrantes (resp. fermantes) à gauche de E_2 dans E_1. Par analogie avec le langage des expressions aithmétiques, nous appelerons profondeur de E_2 dans E_1 le nombre $n_0 - n_F$.

1.2. Sémantique

Soit X un ensemble d'objets x nommés contextes; nous interpréterons les terminaux α, β, ... comme des prédicats unaires sur X, a, b, ... comme

des applications de X dans X, c'est-à-dire comme des instructions de base, et Λ comme l'application identité.

Les règles 3 et 4 expriment la façon de décrire la composition et l'alternance auxquelles on associe leurs sens habituels.

La règle 5 définit l'itération dont la signification est la suivante :

- soit p la profondeur de la liste d'instructions entre accolades ;
- cette liste d'instructions est exécutée et répétée tant que l'on ne rencontre pas une instruction de sortie de niveau quelconque, notée ici $!n$, avec $n \geqslant 1$;
- si le cas se produit, le calcul se poursuit à l'instruction située immédiatement à droite de l'accolade fermante de l'itération de profondeur p-n, si elle existe : dans le cas contraire, on convient de considérer cela comme une erreur.

On conviendra aussi de dire que !0 et Λ ont le même effet.

1.3. Exemple

Un programme EXEL-ALGOL de recherche en table peut être écrit comme suit :

$$i := 1;\ \left\{\begin{array}{l}\text{SI } i > n \text{ ALORS! 1 SINON IS;}\\ \qquad\text{SI } a(i) = x \text{ ALORS. 1 SINON } i := i + 1 \text{ IS}\end{array}\right\}$$

2. LES DIAGRAMMES DE BÖHM ET JACOPINI

2.1. Syntaxe et sémantique

Comme au paragraphe 1.2, nous poserons les définitions suivantes :

- soit X un ensemble d'objets x dénotant un environnement de calcul à un instant donné, ou contexte ;

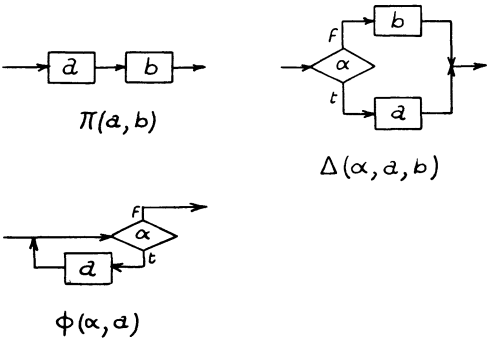

Figure 1.
Les diagrammes de base.

- α un prédicat unaire quelconque sur X;
- a, b deux applications quelconques de X dans X.

Les trois diagrammes de base, nommés Π, Δ et Φ, et représentant la composition, l'alternative et la boucle ont les significations suivantes, comme l'illustre la figure 1 :

- $\Pi(a, b)(x) = b(a(x))$;
- $\Delta(\alpha, a, b)(x) = \Delta(\alpha(x), a(x), b(x))$
  $= a(x)$ si $\alpha(x)$ est vrai
  $= b(x)$ si $\alpha(x)$ est faux;
- $\Phi(\alpha, a)(x) = a(a(\ldots(a(x))\ldots)) = a^n(x)$ avec n le plus petit entier tel que $\alpha(a^n(x))$ soit faux.

On sait qu'il n'est en général pas possible de construire un organigramme ne contenant que les diagrammes de base, équivalent à un organigramme donné.

Toutefois l'ajout d'un prédicat particulier, noté ω, et de trois nouvelles opérations, notées T, F et K, rend cette construction possible.

L'effet des opérations T et F est de transformer l'objet x, dénotation du contexte, en une paire ordonnée, de la façon suivante :

$$x \xrightarrow{T} \langle t, x \rangle,$$

$$x \xrightarrow{F} \langle f, x \rangle,$$

où t et f représentent les deux valeurs booléennes.

Quant à l'opération K, elle supprime la première composante d'une paire ordonnée, valeur du contexte à un instant donné, comme suit

$$\langle v, x \rangle \xrightarrow{K} x \quad \text{où } v \text{ a la valeur } t \text{ ou } f.$$

Enfin la définition du prédicat ω est la suivante :

$$\omega(\langle v, x \rangle) = t \Leftrightarrow v = t,$$
$$\omega(\langle v, x \rangle) = f \Leftrightarrow v = f.$$

Les contextes, ainsi structurés, fonctionnent donc en LIFO : les opérations T, F, K et le prédicat ω n'agissent que sur la valeur booléenne dernière rentrée. On a donc bien à faire avec une pile, laquelle était l'objet de la critique de Cooper évoquée dans l'introduction.

Nous énoncerons donc le résultat suivant dû à Böhm et Jacopini [2].

2.2. THÉORÈME : *Soient :*

- *X un ensemble d'objets x complété éventuellement avec les valeurs booléennes t et f ;*
- Ψ *un ensemble de prédicats unaires,* α, β ... sur X;
- Ω *un ensemble d'applications de X dans X contenant* Λ *l'application identité.*
- $\mathscr{D}(\Psi, \Omega)$ *la classe des applications de X dans X, représentables au moyen d'organigrammes construits sur* $\Psi \cup \Omega$.

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

3. L'INTERPRÈTE

Nous nous proposons de définir l'interprétation σ_E du surlangage EXEL dans l'ensemble des organigrammes composés par Π à partir des diagrammes Δ et Φ construits sur $\Psi \cup \Omega \cup (K, T, F, \omega)$.

3.1. Définition informelle de σ_E

Böhm et Jacopini utilisent la valeur booléenne au sommet de la pile de contexte pour savoir si à un moment donné du déroulement du calcul de l'organigramme considéré, l'instruction présente doit être exécutée ou non : utiliser la pile de contexte pour garder trace des niveaux d'emboitement des itérations, telle est l'idée de base de la définition de σ_E.

L'image par σ_E d'un programme EXEL contiendra un test sur le sommet de la pile avant chaque instruction, cette dernière devant être exécutée ssi ce sommet est t. Une sortie de niveau n (instruction $!n$) a pour effet d'empiler n fois la valeur f : le nombre n est ainsi mémorisé dans la pile. L'exécution d'une itération est réalisée à l'aide du diagramme Φ, la sortie se produit lorsque le sommet de pile a la valeur f, c'est-à-dire après l'exécution d'une occurrence de $!n$, avec $n \geqslant 1$. Après chaque sortie d'itération, une constante booléenne f est effacée au sommet de la pile par l'opération K: ainsi la partie de programme située entre l'instruction $!n$ et l'accolade fermante de la n-ième itération englobante n'est pas exécutée, et, après n sorties d'itérations, la pile est dans l'état $\langle t, x' \rangle$: x' est la nouvelle valeur du contexte, obtenue par exécution des diverses instructions de base présentes dans le programme.

3.2. Définition formelle de σ_E

Soient :

- $\mathscr{A}_1$, $\mathscr{A}_2$, $\mathscr{B}$ des suites d'instructions quelconques ;

- a une instruction élémentaire quelconque ;
- α un prédicat quelconque.

Alors l'application σ_E est définie inductivement comme suit :

$$\sigma_E(a) = \Delta(\omega, \Pi(\Pi(K, a), T), \Lambda) \qquad \text{et} \qquad \sigma_E(\Lambda) = \Lambda,$$

$$\sigma_E(\text{si } \alpha \text{ ALORS } \mathscr{A}_1 \text{ SINON } \mathscr{A}_2 \text{ is}) = \Delta(\omega, \Pi(\Pi(K, \Delta(\alpha, \sigma_E(\mathscr{A}_1), \sigma_E(\mathscr{A}_2))), T), \Lambda),$$

$$\sigma_E(\{\mathscr{B}\}) = \Delta(\omega, \Pi(\Phi(\omega, \sigma_E(\mathscr{B})), K), \Lambda)$$

$$\sigma_E(!n) = \Delta(\omega, \Pi(\Pi(\ldots, \Pi(\Pi(F, F), F), \ldots, F), F), \Lambda)$$

où F figure n fois et Π figure n-1 fois,

$$\sigma_E(\mathscr{A}_1 ; \mathscr{A}_2) = \Pi(\sigma_E(\mathscr{A}_1), \sigma_E(\mathscr{A}_2)).$$

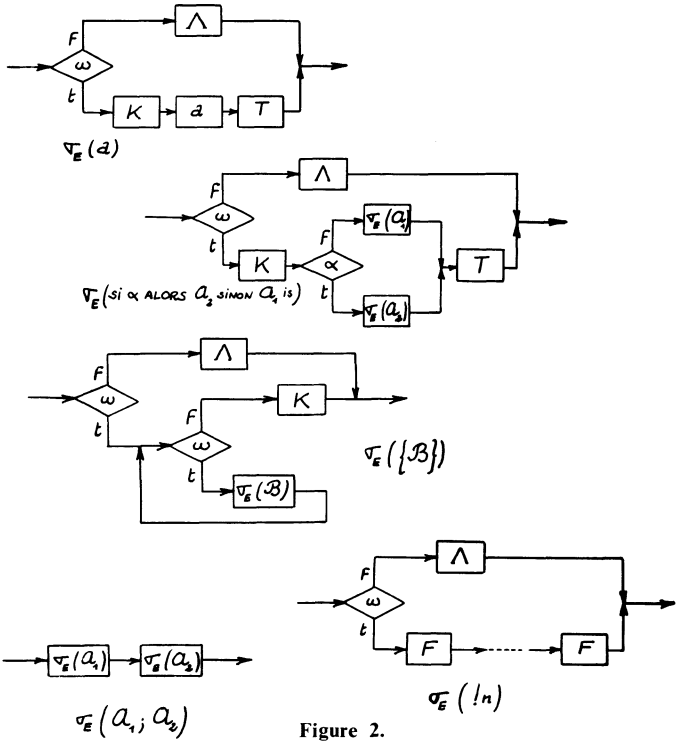

Figure 2.
Organigrammes de σ_E.

La figure 2 illustre cette définition ; on remarquera la présence systématique du test sur le sommet de pile ainsi que la présence de l'opérateur K rendant accessible le contexte pour les instructions élémentaires et les prédicats ; l'opérateur T empile la valeur t en vue de l'exécution de l'instruction suivante.

3.3. Correction de σ_E

Suivant McCarthy, il nous reste à prouver le bien-fondé de la définition de σ_E ([13, 9]).

3.3.1. DÉFINITION : Étant donnés un langage L, une structure E et une interprétation σ de L dans E, nous dirons que σ est correcte si elle respecte la sémantique naïve de L.

3.3.2. THÉORÈME : *L'application σ_E est correcte.*

Preuve : Celle-ci se fait par induction sur la structure des formules du langage EXEL.

$$\sigma_E(a) \langle t, x \rangle = \langle t, a(x) \rangle ;$$

En effet,

$$\begin{aligned}\sigma_E(a) \langle t, x \rangle &= \Delta(\omega, \Pi(\Pi(K, a), T), \Lambda) \langle t, x \rangle \\ &= \Delta(\omega \langle t, x \rangle, \Pi(\Pi(K, a), T) \langle t, x \rangle, \Lambda \langle t, x \rangle) \\ &= \Delta(t, \Pi(\Pi(K, a), T) \langle t, x \rangle, \Lambda \langle t, x \rangle) \\ &= \Pi(\Pi(K, a), T) \langle t, x \rangle \\ &= T(a(K \langle t, x \rangle)) = \langle t, a(x) \rangle\end{aligned}$$

et ceci d'après la sémantique des diagrammes Δ et Π énoncée au paragraphe 2.1.

$$\begin{aligned}&\sigma_E \text{ (SI } \alpha \text{ ALORS } \mathscr{A}_1 \text{ SINON } \mathscr{A}_2 \text{ IS)} \langle t, x \rangle \\ &\quad = \sigma_E(\mathscr{A}_1) \langle t, x \rangle \quad \text{si} \quad \alpha(x) \quad \text{est vrai} \\ &\quad = \sigma_E(\mathscr{A}_2) \langle t, x \rangle \quad \text{si} \quad \alpha(x) \quad \text{est faux} ;\end{aligned}$$

d'après ce qui précède le résultat est évident.

$$\begin{aligned}\sigma_E(\{ \mathscr{B} \}) \langle t, x \rangle &= \Delta(\omega, \Pi(\Phi(\omega, \sigma_E(\mathscr{B})), K), \Lambda) \langle t, x \rangle \\ &= \Pi(\Phi(\omega, \sigma_E(\mathscr{B})), K) \langle t, x \rangle \\ &= K(\Phi(\omega, \sigma_E(\mathscr{B})) \langle t, x \rangle) \\ &= K(\Phi(\omega, \sigma_E(\mathscr{B}))(\sigma_E(\mathscr{B}) \langle t, x \rangle)),\end{aligned}$$

d'après la sémantique du diagramme Φ.

Deux cas peuvent se présenter :

● ● $\mathscr{B} = \mathscr{B}_1 ; \mathscr{B}_2$ avec $\mathscr{B}_1$ et $\mathscr{B}_2$ ne contenant ni itération, ni $!n$; alors :

$$\begin{aligned}\sigma_E(\mathscr{B}) \langle t, x \rangle &= \sigma_E(\mathscr{B}_1 ; \mathscr{B}_2) \langle t, x \rangle \\ &= \Pi(\sigma_E(\mathscr{B}_1), \sigma_E(\mathscr{B}_2)) \langle t, x \rangle \\ &= \sigma_E(\mathscr{B}_2)(\sigma_E(\mathscr{B}_1) \langle t, x \rangle)\end{aligned}$$

Comme le cas des instructions élémentaires et de l'alternative ont été déjà traités, un raisonnement par récurrence sur la longueur des formules $\mathscr{B}_i$ montre que

$$\begin{aligned}\sigma_E(\mathscr{B})\langle t, x\rangle &= \langle t, \sigma_E(\mathscr{B}_2)(\sigma_E(\mathscr{B}_1)x)\rangle \\ &= \langle t, \sigma_E(\mathscr{B})x\rangle.\end{aligned}$$

Il vient donc

$$\begin{aligned}\sigma_E(\{\mathscr{B}\})\langle t, x\rangle &= K(\Phi(\omega, \sigma_E(\mathscr{B}))\langle t, \sigma_E(\mathscr{B})x\rangle) \\ &= K(\Phi(\omega, \sigma_E(\mathscr{B}))\langle t, \sigma_E^n(\mathscr{B})x\rangle)\end{aligned}$$

quel que soit n.

C'est donc le cas où l'itération boucle.

● ● $\mathscr{B} \equiv \mathscr{B}_1 ; !n ; \mathscr{B}_2$ avec $\mathscr{B}_1$ ne contenant ni itération, ni sortie et $\mathscr{B}_2$ quelconque, le cas où $!n$ serait un alternant se ramenant à celui-ci; alors :

$$\begin{aligned}\sigma_E(\mathscr{B})\langle t, x\rangle &= \sigma_E(\mathscr{B}_1 ; !n ; \mathscr{B}_2)\langle t, x\rangle \\ &= \Pi(\Pi(\sigma_E(\mathscr{B}_1), \sigma_E(!n)), \sigma_E(\mathscr{B}_2))\langle t, x\rangle \\ &= \sigma_E(\mathscr{B}_2)(\sigma_E(!n)(\sigma_E(\mathscr{B}_1)\langle t, x\rangle)) \\ &= \sigma_E(\mathscr{B}_2)(\sigma_E(!n)\langle t, \sigma_E(\mathscr{B}_1)x\rangle)\end{aligned}$$

d'après ce qui précède.

Ainsi, de par la définition de $\sigma_E(!n)$, il vient :

$$\begin{aligned}&\sigma_E(\mathscr{B})\langle t, x\rangle \\ &\quad= \sigma_E(\mathscr{B}_2)(\Delta(\omega, \Pi(\Pi(\ldots, \Pi(\Pi(F, F), F), \ldots, F), F), \Lambda)\langle t, \sigma_E(\mathscr{B}_1)x\rangle) \\ &\quad= \sigma_E(\mathscr{B}_2)\langle f, \langle f, \langle \ldots, \langle f, \langle t, \sigma_E(\mathscr{B})x\rangle\rangle \ldots \rangle\rangle\rangle,\end{aligned}$$

car il est aisé de montrer, par récurrence sur n, que

$$\sigma_E(!n)\langle t, x\rangle = \langle f, \langle \ldots \langle f, \langle t, x\rangle\rangle \ldots \rangle\rangle, \qquad \forall n \geqslant 0, \quad \forall x.$$

Comme $\mathscr{B}_2$ n'a pu être engendré par composition qu'à partir des structures précédentes, $\sigma_E(\mathscr{B}_2)$ est de la forme $\Delta(\omega, \ldots, \Lambda)$; on a donc

$$\begin{aligned}&\sigma_E(\mathscr{B})\langle t, x\rangle \\ &\quad= \Delta(\omega, \ldots, \Lambda)\langle f, \langle f, \langle \ldots, \langle f, \langle t, \sigma_E(\mathscr{B}_1)x\rangle\rangle \ldots \rangle\rangle\rangle \\ &\quad= \langle f, \langle f, \langle \ldots, \langle f, \langle t, \sigma_E(\mathscr{B}_1)x\rangle\rangle \ldots \rangle\rangle\rangle,\end{aligned}$$

où f figure n fois.

Il vient donc

$$\begin{aligned}\sigma_E(\{\mathscr{B}\})\langle t, x\rangle &= K(\Phi(\omega, \sigma_E(\mathscr{B}))\langle f, \langle \ldots, \langle f, \langle t, \sigma_E(\mathscr{B}_1)x\rangle\rangle \ldots \rangle\rangle) \\ &= K\langle f, \langle \ldots, \langle f, \langle t, \sigma_E(\mathscr{B}_1)x\rangle\rangle \ldots \rangle\rangle \\ &\quad \text{d'après la sémantique de } \Phi \\ &= \langle f, \langle \ldots, f\langle t, \langle t, \sigma_E(\mathscr{B}_1)x\rangle\rangle \ldots \rangle\rangle \\ &\quad \text{mais où } F \text{ ne figure plus que } n-1 \text{ fois.}\end{aligned}$$

Un raisonnement simple par récurrence sur la profondeur des itérations montre que l'exécution de $!n$ fait sortir des n itérations englobantes.

Soit $\mathscr{P}$ un programme EXEL, le résultat de $\sigma_E(\mathscr{P})\langle t, x\rangle$ ne peut donc être que l'un des deux suivants : ou bien

$$\sigma_E(\mathscr{P})\langle t, x\rangle = \langle t, \mathscr{P}(x)\rangle,$$

c'est-à-dire que σ_E a bien calculé le résultat de $\mathscr{P}$ ou bien

$$\sigma_E(\mathscr{P})\langle t, x\rangle = \langle f, \langle f, \langle \ldots, \langle f, \langle t, (x)\rangle\rangle \ldots \rangle\rangle\rangle,$$

ce qui correspond au cas où le programme exécute une occurrence de $!n$ avec n supérieur au nombre d'itérations l'englobant : on avait convenu de considérer ce cas comme une erreur.

Quant à $\sigma_E(\mathscr{P})\langle f, \langle \ldots \rangle\rangle$, on montre aisément que sa valeur est $\langle f, \langle \ldots \rangle\rangle$, quel que soit $\mathscr{P}$.

Q.E.D.

4. EXEMPLE DE TRADUCTION

Reprenant le programme de recherche en table donné en 1.3, posons

$$a \underset{\text{def}}{=} i := 1, \qquad b \underset{\text{def}}{=} i := i + 1, \qquad \alpha \underset{\text{def}}{=} i > n, \qquad \beta \underset{\text{def}}{=} a(i) = x.$$

alors ce programme s'écrit :

$$\mathscr{P} \underset{\text{def}}{=} a;\ \{\ \text{SI } \alpha \text{ ALORS } !1 \text{ SINON } \Lambda \text{ IS};\ \text{SI } \beta \text{ ALORS } !1 \text{ SINON } b \text{ IS}\ \}$$

Son expression en termes de diagrammes de Böhm et Jacopini est donc la suivante :

$$\sigma_E(\mathscr{P}) = \Pi(\sigma_E(a), \sigma_E(\{\ \mathscr{A}\ ;\ \mathscr{B}\ \})),$$

avec $\mathscr{A} = \text{SI } \alpha \text{ ALORS } !1 \text{ SINON } \Lambda \text{ IS}$

et $\beta = \text{SI } \beta \text{ ALORS } !1 \text{ SINON } b \text{ IS}.$

$$\sigma_E(a) = \Delta(\omega, \Pi(\Pi(K, a), T), \Lambda)$$

$$\sigma_E(\{\ \mathscr{A}\ ;\ \mathscr{B}\ \}) = \Delta(\omega, \Pi(\Phi(\omega, \sigma_E(\mathscr{A}\ ;\ \mathscr{B})), K), \Lambda)$$

$$= \Delta(\omega, \Pi(\Phi(\omega, \Pi(\sigma_E(\mathscr{A}), \sigma_E(\mathscr{B}))), K), \Lambda).$$

Comme

$$\sigma_E(\mathscr{A}) = \Delta(\omega, \Pi(\Pi(K, \Delta(\alpha, \sigma_E(!1), \sigma_E(\Lambda))), T), \Lambda)$$

on a

$$\sigma_E(\mathscr{A}) = \Delta(\omega, \Pi(\Pi(K, \Delta(\alpha, \Delta(\omega, F, \Lambda), \Lambda)), T), \Lambda)$$

et, de façon similaire,

$$\sigma_E(\mathscr{B}) = \Delta(\omega, \Pi(\Pi(K, \Delta(\beta, \Delta(\omega, F, \Lambda), \Delta(\omega, \Pi(\Pi(K, b), T), \Lambda))), T), \Lambda)$$

en substituant les diverses expressions, il vient

$$\sigma_E(\mathscr{P}) = \Pi(\Delta(\omega, \Pi(\Pi(K, a), T), \Lambda),$$
$$\Delta(\omega, \Pi(\Phi(\omega, \Pi(\Delta(\omega, \Pi(\Pi(K, \Delta(\alpha\ \Delta(\omega, F, \Lambda), \Lambda)), T), \Lambda),$$
$$\Delta(\omega, \Pi(\Pi(K, \Delta(\beta, \Delta(\omega, F, \Lambda),$$
$$\Delta(\omega, \Pi(\Pi(K, b), T), \Lambda))), T), \Lambda))), K), \Lambda)).$$

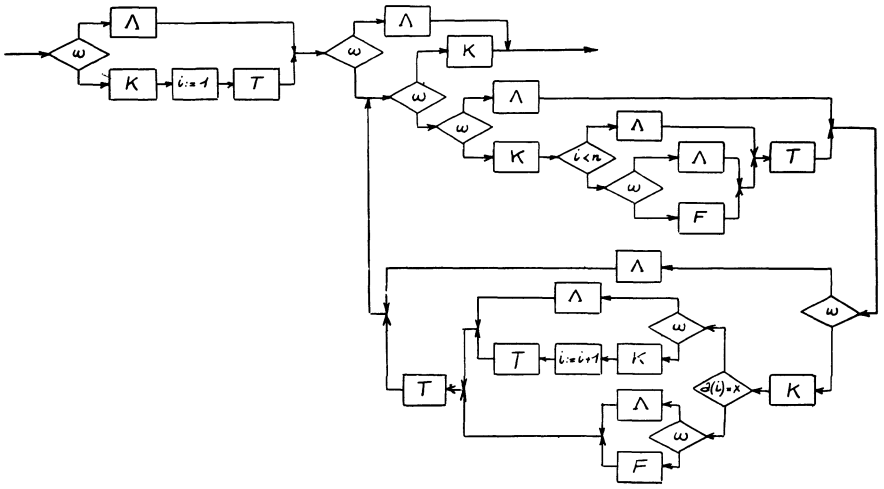

Figure 3.
Sémantique d'un programme EXEL.

La figure 3 illustre ce résultat; par convention, les sorties « vrai » des tests sont dirigées vers le bas de la figure.

BIBLIOGRAPHIE

1. J. ARSAC, *Nouvelles leçons de programmation*, Publications de l'Institut de Programmation, n° IP 75-29.
2. 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.
3. E. ASHCROFT et Z. MANNA, *The Translation of « GO TO » Programs to « WHILE » Programs*, I.F.I.P. 71, C. V. FREIMAN éd., North-Holland Pub., 1972, p. 250-255.
4. G. V. BOCHMANN, *Multiple Exists From a Loop Without the Goto*, Comm. A.C.M., vol. 16, n° 6, 1973, p. 443-444.
5. C. BÖHM et G. JACOPINI, *Flow-diagrams, Turing Machines and Languages with Only Two Formation Rules*, Comm. A.C.M., vol. 9, n° 5, 1966, p. 365-371.

6. D. C. COOPER, *Böhm and Jacopini's Reduction of Flow Charts*, Comm. A.C.M., vol. 10, n° 8, 1967, p. 463-464.
7. E. W. DIJKSTRA, *Goto Statement Considered Harmful*, Comm. A.C.M., vol. 11, n° 3, 1968, p. 147-148.
8. R. W. FLOYD, *Assigning Meanings to Programs*, Proc. of a Symposium in Applied Mathematics, Math. Aspects of Computer Science, A.M.S., 1967, p. 19-32.
9. P. HENDERSON, *Derived Semantics for Some Programming Language Constructs*, Comm. A.C.M., vol. 15, n° 11, 1972, p. 967-973.
10. D. E. KNUTH et R. W. FLOYD, *Notes on Avoiding « GO TO » Statements*, Inf. Process. Letters, n° 1, 1971, p. 23-31.
11. J. KOTT, *Remarques sur la structure des schémas de programmes. Automata, Languages and Programming*, M. NIVAT éd., North-Holland Pub., 1973, p. 265-271.
12. B. LORHO, *De la définition à la traduction des langages de programmation : méthode des attributs sémantiques*, Thèse, Université Paul-Sabatier de Toulouse, 1974.
13. J. MCCARTHY, *Towards a mathematical Science of Computation*, Proc. I.F.I.P. Cong., North Holland Pub. Co., 1962, p. 21-28.
14. L. NOLIN et G. RUGGIU, *Formalization of Exel*, Proc. of the ACM Symp. on Principles of Programming Languages, Boston, 1973, p. 108-119.
15. B. ROBINET, *Un modèle fonctionnel des structures de contrôle*, R.A.I.R.O., Informatique théorique (à paraître).
16. B. ROBINET, *About the Logical Foundations of Data Types*, New Directions on Algorithmic Languages, 1976, S. SCHUMAN éd., Proc. of a WG. 2.1. Meeting.
17. R. D. TENNENT, *The Denotational Semantics of Programming Languages*, Comm. A.C.M., vol. 19, n° 8, 1976, p. 437-453.
18. N. WIRTH, *The Programming Language Pascal* (Revised Report), E.T.H. Zürich, Berichte der Fachgruppe Computer Wissenschaften, n° 5, 1972.
19. W. A. WULF, D. B. RUSSELL et A. N. HABERMAN, *Bliss : a Language for Programming Systems*, Comm. A.C.M., vol. 14, n° 12, 1971, p. 780-790.
20. C. T. ZAHN, *A Control Statement for Natural Top-down Programming*, Programming Symposium, B. ROBINET éd., Springer-Verlag, Lecture Notes in Computer Science, vol. 19, 1974, p. 170-179.