REVUE FRANÇAISE D'AUTOMATIQUE INFORMATIQUE
RECHERCHE OPÉRATIONNELLE. INFORMATIQUE THÉORIQUE

J.-P. FINANCE

Une formalisation de la sémantique des langages de programmation

Revue française d'automatique informatique recherche opérationnelle. Informatique théorique, tome 10, n° R3 (1976), p. 5-21

<http://www.numdam.org/item?id=ITA_1976__10_3_5_0>

R.A.I.R.O. Informatique Théorique
(vol. 10, n° 12, décembre 1976, p. 5 à 21).

UNE FORMALISATION DE LA SÉMANTIQUE DES LANGAGES DE PROGRAMMATION (*)

Deuxième partie

par J.-P. FINANCE (1)

Communiqué par E. Engeler

Résumé. — *Dans la première partie nous avons défini la valeur sémantique d'une phrase d'un langage de programmation comme étant un couple*

accès (formalisant la notion de valeur)
ensemble de calculs

et nous avons précisé la notion de structure d'information (S. I.) qui apparait comme un cadre permettant de formaliser ces concepts. A cette occasion nous avons illustré notre formalisation sur le sous langage GOL *d'Algol* 68.

Dans cette deuxième partie nous définissons le terme ensemble de calcul (§ 5) *puis nous complétons la définition formelle de* GOL (§ 6).

La nette distinction entre l'aspect statique de la sémantique d'un langage (axiomes caractérisant les objets manipulés) et l'aspect dynamique (calculs) permet de donner une caractérisation « orthogonale » d'un langage certainement utilisable à la fois pour construire une implémentation et pour définir des méthodes de preuves de programme.

5. SYSTÈMES DE CALCULS

5.1. Calculs finis et infinis

Une élaboration d'un programme P d'un langage algorithmique $\mathscr{L}$ est une suite d'élaborations de phrases élémentaires de P.

Une telle suite s'appellera un calcul. On est conduit à :

DÉFINITION : Un calcul du langage $\mathscr{L}$ est une suite finie ou non d'éléments de l'ensemble $\mathscr{M}$od des modifications élémentaires de la structure d'information de $\mathscr{L}$.

Un calcul fini est un élément du monoïde libre $\mathscr{M}$od*.

Nous noterons $\mathscr{M}$od$^{\omega}$ l'ensemble des calculs infinis de $\mathscr{L}$ et

$$\mathscr{M}\text{od}^{\infty} = \mathscr{M}\text{od}^* \cup \mathscr{M}\text{od}^{\omega}.$$

On étend l'opération de concaténation à $\mathscr{M}$od$^{\infty}$ par :

— si

$$m_1 = a_1 a_2 \ldots a_n \in \mathscr{M}\text{od}^*$$

(*) Reçu mai 1975 (version remaniée reçue le 23 février 1976). La première partie de cet article a paru dans le n° d'août de la *R.A.I.R.O.* (R-2), 1976.

(1) Université de Nancy II, I.U.T. d'Informatique.

et

$m_2 = b_1 b_2 \ldots b_m \in \mathcal{M}\text{od}^{\omega}$ alors $m_1 . m_2 = a_1 a_2 \ldots an\, b_1 b_2 \ldots b_m$,

– si $m_1 \in \mathcal{M}\text{od}^{\omega}$ et $m_2 \in \mathcal{M}\text{od}^{\infty}$ alors $m_1 . m_2 = m_1$.

Ainsi $(\mathcal{M}\text{od}^*, ., \wedge)$ et $(\mathcal{M}\text{od}^{\infty}, ., \wedge)$ sont deux monoïdes (où $\wedge$ est le mot vide).

Exemple 1 : Au début de programme considéré en (§ 3.5 exemple 2) est associé le calcul :

ident(*x*, *poss 3*).ident(*y*, *poss loc réel*).affect(*poss y*, *poss* 4.1)
.ident(*z*, *poss*(*y*, *x*)).

Tout calcul fini peut être interprété comme une modification de $\mathcal{S}$ lorsqu'on interprète la concaténation comme la composition (dans l'ordre d'écriture) des modifications élémentaires.

Ainsi $C = \pi_1 . \pi_2 \ldots \pi_n$ sera interprété comme la modification

$$\pi_n \circ \pi_{n-1} \circ \ldots \circ \pi_1 \in \widehat{\mathcal{M}\text{od}}.$$

On retrouve la notion de changement d'état, sans oublier pour autant l'histoire de l'élaboration du programme.

Nous n'interpréterons pas les calculs infinis.

5.2. Ensembles de calculs

Si la concaténation de $\mathcal{M}\text{od}^{\infty}$ permet d'exprimer l'élaboration des phrases sérielles (au sens d'Algol 68), il faut pouvoir rendre compte des propositions conditionnelles. En effet à tout programme qui contient une telle proposition *C* sont associées plusieurs élaborations possibles selon qu'on élabore, en fonction des données, la « proposition alors » ou la « proposition sinon » de *C*. La signification d'un tel programme n'est pas un calcul mais un ensemble de calculs. Cette notion est également nécessaire pour rendre compte de l'indéterminisme, et en particulier de celui qui provient des propositions collatérales.

Exemple 2 : A la phrase *conditionnelle* (*b*, *affectation* (*x*, *3*), *affectation* (*y*, *4*)) est associé l'ensemble des calculs :

$$j(\text{poss } b, \text{ poss } \underline{vrai}).\text{affect}(\text{poss } x, \text{ poss } 3) \cup$$
$$\bar{j}(\text{poss } b, \text{ poss } \underline{vrai}).\text{affect}(\text{poss } y, \text{ poss } 4).$$

5.3. Système de calculs et opérations sur les ensembles de calculs

5.3.1. *Système associé à un programme*

Nous avons déjà remarqué que l'ensemble de calculs associé à une phrase dépend de ceux qui sont associés aux phrases qui la composent. Un tel ensemble

est défini par un système d'égalités, ou plus exactement un système à point fixe si le langage admet la récursivité.

Exemple 3 : Soit P la phrase GOL_0 :

$$conditionnelle(B, E_1, E_2)$$

on lui associe le système $\mathscr{M}(P)$:

$$\begin{cases} \tilde{P} = \tilde{B}.(j(poss\,B, poss\,\underline{vrai}).E_1 \cup \bar{j}(poss\,B, poss\,\underline{vrai}).\tilde{E}_2), \\ \mathscr{M}(B), \\ \mathscr{M}(E_i) \qquad \text{pour} \quad i = 1, 2. \end{cases}$$

Il est formé d'une équation liant l'inconnue $\tilde{P}$ associée à P aux inconnues associées aux phrases composantes « directes » de P, ainsi que des sous-systèmes associés à chacune de ces phrases.

Ainsi définir l'ensemble de calculs associé à un programme quelconque d'un langage $\mathscr{L}$, c'est associer à chaque type de construction des phrases du langage un système à point fixe, appelé système des calculs, dans le treillis $(\mathfrak{P}(\mathscr{M}od^{\infty}), \subset)$ où $\subset$ est l'inclusion ensembliste.

NOTATION : Si P est une phrase d'un langage $\mathscr{L}$, $\mathscr{M}(P)$ désigne le système de calculs associé à P; $\tilde{P}$ est l'inconnue correspondant à P dans ce système.

Les fonctions de base des systèmes de calculs, sont les opérations sur les schémas de calculs. Définissons-les maintenant.

5.3.2. *Opérations sur les ensembles de calculs*

a) *Concaténation :* La concaténation « . » de $\mathscr{M}od^{\infty}$ s'étend en une loi de composition interne dans $\mathfrak{P}(\mathscr{M}od^{\infty})$ définie par

$$C_1.C_2 = \{\alpha.\beta \mid \alpha \in C_1, \beta \in C_2\}.$$

Exemple 4 : Soit E la proposition sérielle de GOL_0 :

$$serie(p, listser_n(E_1, E_2, \ldots, E_n)).$$

Le système de calculs qui lui est associé est

$$\mathscr{M}(E)\begin{cases} \tilde{E} = \tilde{E}_1.\tilde{E}_2 \ldots \tilde{E}_n, \\ \mathscr{M}(E_i), \qquad i = 1, \ldots, n. \end{cases}$$

b) *Réunion :* C'est l'opérateur ensembliste « $\cup$ » habituel; l'exemple 3 au paragraphe précédent illustre son utilisation.

c) *Mélange :* Si la multiplication permet de rendre compte de la notion de proposition sérielle, le mélange m permettra de formaliser celle de proposition collatérale : on traduit que deux propositions s'élaborent collatéra-

lement en exprimant qu'il n'y a aucun ordre à priori entre l'élaboration de l'un des constituants de la première proposition et un constituant de l'autre.

La loi de composition interne m de $\mathfrak{P}(\mathscr{M}\mathrm{od}^{\infty})$ est définie par :

1° si α et β appartiennent à $\mathscr{M}\mathrm{od}^{*}$ alors :

$$m(\alpha, \beta) = \{ a_0 . b_1 \ldots b_n . a_n \mid a_0 a_1 \ldots a_n = \alpha, b_1 \ldots b_n = \beta \} \in \mathfrak{P}(\mathscr{M}\mathrm{od}^{\infty});$$

2° si $\alpha \in \mathscr{M}\mathrm{od}^{\omega}$ et $\beta \in \mathscr{M}\mathrm{od}^{*}$ alors :

$$m(\alpha, \beta) = \{ m(\alpha_1, \beta) . \alpha_2 \mid \alpha_1 \in \mathscr{M}\mathrm{od}^{*}, \alpha_1 . \alpha_2 = \alpha \};$$

3° si α et β appartiennent à $\mathscr{M}\mathrm{od}^{\omega}$ alors :

$$m(\alpha, \beta) = \{ m(\alpha_1, \beta_1) . \alpha_2 \mid \alpha_1, \beta_1 \in \mathscr{M}\mathrm{od}^{*}, \alpha_1 . \alpha_2 = \alpha, \beta_1 . \beta_2 = \beta \}$$
$$\cup \{ m(\alpha_1, \beta_1) . \beta_2 \mid \alpha_1, \beta_1 \in \mathscr{M}\mathrm{od}^{*}, \alpha_1 . \alpha_2 = \alpha, \beta_1 . \beta_2 = \beta \};$$

4° si A et B appartiennent à $\mathfrak{P}(\mathscr{M}\mathrm{od}^{\infty})$ alors :

$$m(A, B) = \{ m(\alpha, \beta) \mid \alpha \in A, \beta \in B \}.$$

On peut vérifier que m est associative et définir le mélange d'un nombre quelconque (fini) d'ensembles de calculs.

En particulier la notation : $\underset{1 \leqq i \leqq n}{m}$ sera préférée à $m(C_1, C_2, \ldots, C_n)$.

Exemple 5 : Soit P la proposition collatérale de GOL_0 :
coll (*listcoll*$_n$ $(E_1, E_2, \ldots, E_n)$, μ); le système de calculs associé est

$$\mathscr{M}(P) \begin{cases} \tilde{P} = \underset{1 \leqq i \leqq n}{m} (\tilde{E_i}), \\ \mathscr{M}(E_i), \qquad i = 1, \ldots, n. \end{cases}$$

d) *Substitution* : Lors de l'appel d'une procédure récursive, tout identificateur local représente un objet différent pour chaque réincarnation de cette procédure.

Pour traiter intuitivement ce problème, on peut concevoir de transformer, à chaque réincarnation de la procédure, tout identificateur local en un identificateur qui n'a jamais été utilisé et ne le sera plus (on peut envisager de transformer l'identificateur x en x' puis en x'' ...).

Nous formalisons cette idée en associant à tout appel A de procédure une application τ_A de $\mathfrak{P}(\mathscr{M}\mathrm{od}^{\infty})$ dans lui-même. Plus précisément le symbole fonctionnel binaire *sub* de la S.I. permet d'exprimer la « transformation » de tout identificateur local x à la procédure P en un nouvel identificateur *sub A x*. τ_A est défini par

$$\tau_A(\pi(u_1, \ldots, u_n)) = \pi(sub\, A\, u_1, \ldots, sub\, A\, u_n)$$

pour tout schéma de modifications élémentaires π à n arguments. τ_A se prolonge alors immédiatement aux ensembles de calculs si on impose que

(i) $\tau_A(C_1.C_2) = \tau_A(C_1).\tau_A(C_2)$;

(ii) $\tau_A(C_1 \cup C_2) = \tau_A(C_1) \cup \tau_A(C_2)$

pour tous ensembles de calculs C_1, C_2.

[En particulier on déduit de (i), (ü) et (*c*) que

$$\tau_A(m(C_1, C_2)) = m(\tau_A(C_1), \tau_A(C_2))].$$

REMARQUES : 1) On traite le cas des identificateurs globaux en imposant des axiomes *sub A* $y \equiv y$ pour tout tel identificateur.

2) *sub* permet aussi de rendre compte des paramètres des procédures dans le cas d'Algol 68 (et donc de GOL).

On vérifie que concaténation, réunion, mélange et substitution sont continues dans le treillis $(\mathfrak{P}(\mathscr{M}\mathrm{od}^\infty), \subset)$. Pour tout programme P, le système $\mathscr{M}(P)$ admet donc un point fixe minimal : par définition l'ensemble de calculs associé à P est la composante correspondant à $\tilde{P}$ dans cette solution minimale.

3) il peut sembler plus naturel, dans la plupart des langages de programmation, de rendre compte de la notion de procédure en associant à un tel objet, non pas un ensemble de calculs, mais un schéma d'ensembles de calculs (i. e. ensemble de calculs dépendant de paramètres). Cette notion est également nécessaire pour rendre compte de propositions unitaires définies de manière récursive (telles que l'affectation en Algol 68) pour lesquelles l'utilisation de modifications élémentaires se révèle insuffisante. Une telle formalisation est présentée en [5]. Nous n'utiliserons ici que très ponctuellement cette notion de schéma d'ensembles de calculs, au paragraphe 6.2.10 (iv).

6. SÉMANTIQUE DE GOL

6.1. Introduction

Rappelons que, pour nous, définir la sémantique d'un langage $\mathscr{L}$ consiste à :

a) (1) introduire le langage pivot $\mathscr{L}_0$ (2) préciser une fonction de traduction concrète.

b) (1) caractériser les objets internes qui sont manipulés dans $\mathscr{L}$ ainsi que (2) les modifications élémentaires associées à certaines propositions de GOL_0.

c) définir une fonction sémantique associant à toute phrase de $\mathfrak{P}_0$ un couple [calculs, valeur interne (accès)]. Les calculs sont caractérisés par un système à point fixe et la valeur interne par des axiomes d'accès.

a) (1) et *b*) ont été précisés dans les sections précédentes pour le langage GOL (on caractérise ainsi la S.I. de GOL). Nous présentons, dans cette section, les étapes *a*) (2) et *c*) en associant à chaque type de phrase P de GOL_0 :

- traduction concrète;
- axiomes d'accès;
- système de calculs $\mathscr{M}(P)$.

décembre 1976.

Ce qu'on peut résumer sur la figure 6.

Pour alléger la description qui suit, nous adoptons les conventions suivantes :

Dans chaque paragraphe, l'alinéa (i) précise le type de la phrase considérée (qui sera, le plus souvent, appelée P); les alinéas (ii), (iii) et (iv) définissent respectivement la traduction concrète, les axiomes d'accès et le système de calculs associés à cette phrase. En particulier, comme dans la définition de X'_p, les axiomes d'accès seront regroupés en schémas notés SH''.

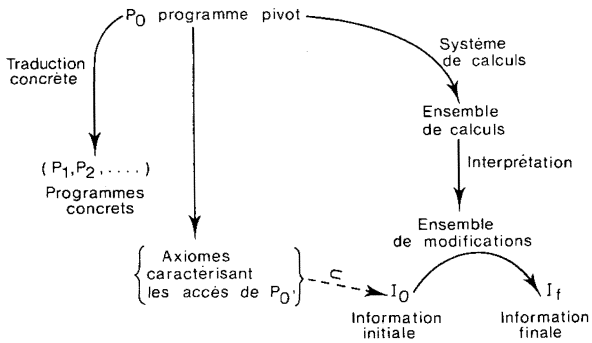

Figure 6.

Objets associés à un programme pivot.

6.2. Traduction concrète, calculs et accès associés aux phrases de GOL_0

Nous traitons d'abord les propositions unitaires et ensuite les propositions parenthésées.

6.2.1. *Propositions unitaires dereperées*

(i) $P = derep\ P_1$ où $P_1 \in$ UNIT;

(ii) $\mathscr{T}(P) = \mathscr{T}(P_1)$. On supprime ainsi les symboles *derep* qui se trouvent en tête de P;

(iii) $SH''_{1.1} = \{ poss\ derep\ P_1 \equiv rep\ poss\ P_1 \mid P_1 \in \text{UNIT} \}$ qui caractérise le dereperage syntaxique;

(iv) $\begin{cases} \tilde{P} = \tilde{P}_1; \\ \mathscr{M}(P_1). \end{cases}$

6.2.2. *Formules*

— (i) $P = sigmoins\ P_1\ P_2$ où $P_1, P_2 \in SEC$;

(ii) $\mathscr{T}(P) = \{ \mathscr{P}_1 - \mathscr{P}_2 \mid \mathscr{P}_i \in \mathscr{T}(P_i)\ i = 1, 2 \}$;

(iii) $SH''_{2.1} = \{ poss\ sigmoins\ P_1\ P_2 \equiv moins\ poss\ P_1\ poss\ P_2 \mid P_i \in \text{SEC}$ pour $i = 1, 2 \}$;

(iv) $\begin{cases} \tilde{P} = m(\tilde{P}_1, \tilde{P}_2) \\ \mathscr{M}(P_i)\ i = 1, 2. \end{cases}$

— (i) P = *inférieur* $P_1\ P_2$ où $P_1, P_2 \in$ SEC;
(ii) $\mathscr{T}(P) = \{ \mathscr{P}_1 < \mathscr{P}_2 \mid \mathscr{P}_i \in \mathscr{T}(P_i)\ i = 1, 2 \}$;
(iii) $\mathrm{SH}''_{2,2} = \{$ *poss inférieur* $P_1\ P_2 \equiv$ *inf poss* P_1 *poss* $P_2 \mid P_i \in$ SEC pour $i = 1, 2 \}$;
(iv) $\begin{cases} \tilde{P} = m(\tilde{P}_1, \tilde{P}_2), \\ \mathscr{M}(P_i)\ i = 1,2. \end{cases}$

— (i) P = *égal* $P_1\ P_2$ avec $P_1, P_2 \in$ SEC;
(ii) $\mathscr{T}(P) = \{ \mathscr{P}_1 = \mathscr{P}_2 \mid \mathscr{P}_i \in \mathscr{T}(P_i)$ pour $i = 1, 2 \}$;
(iii) $\mathrm{SH}''_{2.3} = \{$ *poss* $P_1 \not\equiv$ *poss* $P_2 \supset$ *poss égal* $P_1\ P_2 \equiv$ *poss faux*, *poss* $P_1 \equiv$ *poss* $P_2 \supset$ *poss égal* $P_1\ P_2 \equiv$ *poss vrai* $\mid$ $P_1, P_2 \in$ SEC $\}$;
(iv) $\begin{cases} \tilde{P} = m(\tilde{P}_1, \tilde{P}_2), \\ \mathscr{M}(P_i)\ i = 1, 2. \end{cases}$

6.2.3. *Déclarations d'identité*

(i) D = *decid* $\mu\ x\ P_1$ où $\mu \in L_0^{\mathrm{id}}$, $x \in L_0^{\mathrm{id}}$ et $P_1 \in$ UNIT;
(ii) $\mathscr{T}(\mathrm{D}) = \{ \mu\ x = \mathscr{P}_1 \mid \mathscr{P}_1 \in \mathscr{T}(P_1) \}$;
(iii) Aucun axiome d'accès ne caractérise D car une déclaration d'identité ne possède aucune valeur.
(iv) $\begin{cases} \tilde{D} = \tilde{P}_1.\mathrm{ident}(x, \textit{poss}\ P_1); \\ \mathscr{M}(P_1). \end{cases}$

En effet, GOL ne contenant ni valeurs multiples, ni déclarations de modes, la définition d'une telle déclaration donnée en [6] (§ 7.4.2) est considérablement simplifiée. Elle consiste en :
— l'élaboration de la proposition unitaire P_1;
— la « définition » du schéma fonctionnel *poss x.* : c'est le rôle du schéma de modifications élémentaires ident [§ 3.6.4 *a*].

6.2.4. *Affectations*

(i) P = *affectation* $z\ P_1$ où $z \in$ PRIM et $P_1 \in$ UNIT;
(ii) $\mathscr{T}(P) = \{ \mathscr{Z} := \mathscr{P}_1 \mid \mathscr{Z} \in \mathscr{T}(z), \mathscr{P}_1 \in \mathscr{T}(P_1) \}$;
(iii) $\mathrm{SH}''_{4,1} = \{$ *poss affectation* $z\ P_1 \equiv$ *poss* $z \mid z \in$ PRIM, $P_1 \in$ UNIT $\}$.

La valeur d'une affectation est celle de sa destination.

(iv) Ici encore, l'absence de valeurs multiples en GOL simplifie la définition d'une affectation. Le mécanisme introduit en [6] (§ 8.3.1.2) dont l'un des rôles essentiels consiste à traiter les bornes des valeurs multiples (notamment flexibles) est inutile ici. Ainsi l'exécution de P comprend :
— l'élaboration collatérale de z et E;
— la vérification d'un certain nombre de cohérences [6] (§ 8.3.1.2, *c*), pas 1);
— la modification de la fonction repère.

décembre 1976.

Ainsi

$$\mathscr{M}(P) = \begin{cases} \tilde{P} = m(\tilde{z}, \tilde{P}_1).\text{affect}(\textit{poss}\ z, \textit{poss}\ P_1); \\ \mathscr{M}(z); \\ \mathscr{M}(P_1). \end{cases}$$

L'opération mélange traduit l'élaboration collatérale de z et P_1, le schéma affect (§ 3.6.4, *b*) exprime les deux étapes suivantes de l'affectation.

6.2.5. *Relations d'identité*

(i) $P = \textit{relid}\ P_1\ P_2$ où $P_1, P_2 \in \text{TERT}$;
(ii) $\mathscr{T}(P) = \{ \mathscr{P}_1 : = : \mathscr{P}_2 \mid \mathscr{P}_i \in \mathscr{T}(P_i)$ pour $i = 1, 2 \}$;
(iii) $\text{SH}''_{5.1} = \{ \textit{poss relid}\ P_1\ P_2 \equiv \textit{cond poss}\ P_1\ \textit{poss}\ P_2\ \textit{poss vrai poss faux} \mid P_1, P_2 \in \text{TERT} \}$;
(iv) $\begin{cases} \tilde{P} = m(\tilde{P}_1, \tilde{P}_2), \\ \mathscr{M}(P_i)\ i = 1,2. \end{cases}$

On traduit [6] (§ 8.3.3.2) l'élaboration collatérale de P_1 et P_2.

6.2.6. *Sélections*

(i) $P = \textit{de}\ x\ P_1$ où $x \in L_0^{\text{id}}$ et $P_1 \in \text{SEC}$;
(ii) $\mathscr{T}(P) = \{ x\ \textit{de}\ \mathscr{P}_1 \mid \mathscr{P}_1 \in \mathscr{T}(P_1) \}$;
(iii) $\text{SH}''_{6.1} = \{ \textit{poss de}\ x\ P_1 \equiv ch_x\ \textit{poss}\ P_1 \mid P_1 \in \text{SEC} \}$.

Remarquons que la caractérisation, dans notre formation, de $ch_x\ u$ lorsque $u \in \text{NOM}$ (schéma d'axiomes $\text{SH}'_{1.1}$) évite de traiter distinctement le cas où *poss* P_1 est un nom.

(iv) $\begin{cases} \tilde{P} = \tilde{P}_1.\text{selec}(P_1), \\ \mathscr{M}(P_1). \end{cases}$

selec est le schéma de modification élémentaire défini au paragraphe 3.6.4 (*d*) exprimant que la valeur de P_1 doit être différente de *nil* [6] (§ 8.5.2.1).

6.2.7. *Générateurs*

(i) $P = \textit{genloc}\ \underline{\mu}\ p$ où $\underline{\mu} \in L_0^{\text{decl}}$ et $p \in \text{PORTÉE}$;
(ii) $\mathscr{T}(P) = \{ \underline{\textit{loc}}\ \underline{\mu} \}$;
(iii) $\text{SH}''_{6.1} = \{ \textit{portée poss genloc}\ \underline{\mu p} \equiv p, \textit{portée genloc}\ \underline{\mu p} \equiv p \mid \underline{\mu} \in L_0^{\text{decl}}, p \in \text{PORTÉE} \}$.

En Algol 68, l'élaboration de ce générateur se ramène essentiellement à celle de son déclareur effectif μ. La définition de l'élaboration d'un déclareur effectif permet de créer un nom (de mode repère de μ) ainsi que tous les « sous-noms associés »; elle est récursive pour définir les descripteurs éventuels de ces sous-noms.

En GOL nous ne rencontrons pas ces problèmes et nous nous contentons de « créer » un nom en précisant sa portée; l'« obtention » des sous-noms résulte immédiatement, lors de la première affectation, des schémas d'axiomes

$SH'_{1.1}$ et $SH'_{1.2}$. De plus on définit *portée P* pour permettre une caractérisation convenable de la traduction d'un programme (§ 6.2.12). En effet on exigera, par exemple, que si $\mathscr{P}$ est la traduction de P, la portée de la plus petite région qui le contienne dans le programme concret soit p.

(iv) $\tilde{P}$ = e, e est le mot vide de $\mathscr{M}od^{\infty}$.

De même

(i) P = *genglob* $\underline{\mu}$ où $\underline{\mu} \in L_0^{decl}$,

(ii) $\mathscr{T}(P) = \{ \underline{tas}\ \underline{\mu} \}$;

(iii) $SH''_{7,2} = \{ portée\ poss\ genglob\ \mu \equiv \theta,\ portée\ genglob\ \underline{\mu} \equiv \theta \mid \underline{\mu} \in L_0^{id} \}$;

(iv) $\tilde{P}$ = e.

6.2.8. *Notations de valeurs simples et identificateurs*

(i) $P \in L_0^{ent} \cup L_0^{bool} \cup L_0^{id} \cup \{ \underline{nil} \}$;

(ii) $\mathscr{T}(P) = P$;

(iii) Si $P \in L_0^{ent} \cup L_0^{bool}$, sa valeur associée n'est pas définie par un axiome mais devrait l'être au niveau d'une interprétation (de même qu'en [6] on ne caractérise pas les objets internes associés aux notations de réels, d'entiers...). Remarquons cependant que l'on pourrait axiomatiser les objets internes qui sont possédés par les notations de valeurs simples à l'aide des axiomes classiques de l'arithmétique et de la logique. Nous nous contenterons ici de préciser :

$$SH''_{8,1} = \{ poss\ \underline{vrai} \not\equiv poss\ \underline{faux} \},$$

$$SH''_{8,2} = \{ portée\ poss\ P \equiv \theta \mid P \in L_0^{ent} \cup L_0^{bool} \}.$$

De même *poss* $\underline{nil}$ est caractérisé par l'axiome

$$SH''_{8,3} = \{ portée\ poss\ \underline{nil} \equiv \theta \}.$$

Enfin si $P \in L_0^{id}$, *poss* P est défini par une déclaration d'identité et donc aucun axiome d'accès ne le caractérise.

(iv) $\tilde{P}$ = e.

6.2.9. *Notations de routines*

(i) P = *notproc* $(L, \underline{\mu}, P_1, q, p)$ où $L \in$ PARAM; $\underline{\mu} \in$ GENRE; $P_1 \in$ UNIT; $q, p \in$ PORTÉE.

Rappelons que q précise (la portée qui est) la région définie par la notation P [6] (§ 4.1.1) et que p représente la portée de la routine possédée par P [6] (§ 2.2.4.2).

(ii) L est un paramètre formel, une liste collatérale, une liste sérielle ou un paramètre réduit à λ; μ est soit λ, soit un déclareur. Ainsi

- si $L = \lambda$ et $\underline{\mu} = \lambda$ alors $\mathscr{T}(P) = \mathscr{T}(P_1)$;
- si $L = \lambda$ et $\underline{\mu} \in L_0^{decl}$ alors $\mathscr{T}(P) = \{ \underline{\mu} : \mathscr{P}_1 \mid \mathscr{P}_1 \in \mathscr{T}(P_1) \}$;

- si $L = listcoll_n\ parfor\ \mu_1\ z_1\ parfor\ \mu_2\ z_2 \ldots parfor\ \mu_n\ z_n$ et $\mu = \lambda$ alors :

$$\mathscr{T}(P) = \{(\mu_1 z_1, \mu_2 z_2, \ldots, \mu_n z_n) : \mathscr{P}_1 \mid \mathscr{P}_1 \in \mathscr{T}(P_1)\};$$

- si L a la même forme que ci-dessus et $\mu \in L_0^{decl}$ alors :

$$\mathscr{T}(P) = \{(\mu_1 z_1, \mu_2 z_2, \ldots, \mu_n z_n)\mu : \mathscr{P}_1 \mid \mathscr{P}_1 \in \mathscr{T}(P_1)\}.$$

On définit de manière analogue la traduction de P lorsque la liste des paramètres est sérielle ou lorsqu'elle est réduite à un élément.

(iii) Une notation de routine P « possède une routine qui est une même suite de symboles qu'une certaine proposition fermée » P' [6] (§ 5.4.2).

poss P est défini par le schéma d'axiomes

$$\begin{aligned}
SH''_{9,1} = \{ & P \equiv notproc(L, \mu, P_1, q, p) \supset \\
& (L \equiv \lambda \supset poss\,P \equiv P_1) \wedge \\
& (L \equiv parfor\,\mu_1\,z_1 \supset poss\,P \equiv parent(serie(q, listser_2 \\
& \quad (decid\,\mu_1\,z_1 \sim_1, P_1)))) \wedge \\
& (L \equiv listcoll_n\,parfor\,\mu_1\,z_1 \ldots parfor\,\mu_n\,z_n \supset poss\,P \equiv \\
& \quad parent(serie(q, listser_2(listcoll_n\,decid\,\mu_1\,z_1 \sim_1 \\
& \quad \ldots decid\,\mu_n\,z_n \sim_n, P_1)))) \wedge \\
& (L \equiv listser_n\,parfor\,\mu_1\,z_1 \ldots parfor\,\mu_n\,z_n \supset poss\,P \equiv \\
& parent(serie(q, listser_2(listser_n\,decid\,\mu_1\,z_1 \sim_1 \ldots \\
& decid\,\mu_n\,z_n \sim_n, P_1)))) \wedge \\
& portée\,P \equiv q \wedge \\
& portée\,poss\,P \equiv p \mid P \in \text{NOTPROC};\ L \in \text{PARAM};\ \mu \in \text{GENRE}; \\
& \qquad P_1 \in \text{UNIT};\ q, p \in \text{PORTÉE};\ \mu_i \in L_0^{decl} \\
& \qquad z_i \in L_0^{id}\}
\end{aligned}$$

Appelons P' la proposition possédée par P.

Ces axiomes expriment que, selon la forme de L, P' est soit P_1, soit une proposition fermée avec en tête une déclaration collatérale $\mu_1 z_1 = \sim_1, \ldots, \mu_n z_n = \sim_n$ (resp. une déclaration sérielle $\mu_1 z_1 = \sim_1; \ldots; \mu_n z_n = \sim_n$) où les $\sim_i$ sont des identificateurs qui n'apparaissent que dans les routines. Les $\sim_i$ sont numérotés pour permettre de les associer simplement aux paramètres effectifs correspondants lors d'un appel.

D'autre part la formule *portée* $P \equiv q$ permet le « contrôle des portées » lors de la traduction d'un programme (§ 6.2.12).

L'élaboration (resp. la valeur) d'un appel de procédure est l'élaboration (resp. la valeur) de la proposition déduite de P' en remplaçant les symboles $\sim_i$ par les paramètres effectifs correspondants [6] (§ 8.6.2.2). En GOL le problème de transmission des paramètres est résolu, conformément à l'esprit de [6], à l'aide de « substitutions » (exprimées par le symbole fonctionnel binaire *sub*.

(iv) $\mathscr{M}(P) = \begin{cases} \tilde{P} = \wedge, \\ \mathscr{M}(P') \text{ (où } P' \text{ est la proposition possédée par } P). \end{cases}$

On exprime ainsi que l'élaboration de P ne correspond à aucune action et on place le système $\mathscr{M}(P')$ dans le système $\mathscr{M}(P)$, ce qui permet d'introduire l'ensemble de calculs P' qui sera utilisé dans les appels de la procédure.

6.2.10. *Appels de procédures*

(i) A = *appel* G L où $G \in$ PRIM et $L \in$ PAREF;

(ii) • si $L = \lambda$ alors $\mathscr{T}(A) = \mathscr{T}(G)$,

• si $L = listcoll_n \; E_1 \ldots E_n$ alors $\mathscr{T}(A) = \{ \mathscr{G}(\mathscr{E}_1, \mathscr{E}_2, \ldots, \mathscr{E}_n) \mid \mathscr{G} \in \mathscr{T}(G);\ \mathscr{E}_i \in \mathscr{T}(E_i)\ \ i = 1 \ldots n \}$,

• si $L = listser_n \; E_1 \ldots E_n$ alors $\mathscr{T}(A) = \{ \mathscr{G}(\mathscr{E}_1; \mathscr{E}_2; \ldots; \mathscr{E}_n) \mid \mathscr{G} \in \mathscr{T}(\mathrm{G});\ \mathscr{E}_i \in \mathscr{T}(\mathrm{E}_i)\ \ i = 1 \ldots n \}$;

(iii) L'élaboration de A est définie en [6] (§ 8.6.2.1). Elle consiste essentiellement en :

– l'élaboration de G qui « fournit » une notation de routine (iv);

– la *protection* de la routine P' obtenue ci-dessus (§ 6.2.9). Il s'agit de traiter les identificateurs locaux qui doivent être nouveaux à chaque appel (iii′);

– la substitution du paramètre effectif E_i au symbole $\sim_i$ pour $i = 1, \ldots, n$ dans le cas de procédure à n paramètres (iii″).

– l'élaboration de la proposition fermée obtenue à la suite des étapes précédentes (iv).

(iii′) *Traitement des identificateurs locaux à la procédure.*

A la suite de l'élaboration $\tilde{G}$ du primaire G, l'information courante (i. e. celle qui est l'image de l'information de départ par la modification interprétation du calcul se terminant par $\tilde{G}$) contient un théorème de la forme *poss* $G \equiv P'$ (avec les notations du paragraphe 6.2.9).

P' est une proposition fermée contenant, outre les $\sim_i$, un certain nombre d'identificateurs locaux parmi lesquels les paramètres formels. Pour exprimer qu'à chaque appel A correspond de nouveaux identificateurs locaux, on transforme ces identificateurs à l'aide du symbole fonctionnel *sub*.

Le schéma fonctionnel associé à *poss A* se déduira du schéma *poss P′* en « substituant » à tout identificateur local x, l'élément *sub A x* de EXT.

On exige alors :

– que le schéma fonctionnel *sub A x* soit différent (formellement) de tout autre élément de EXT si x est local;

– que *sub A x* soit égal (formellement) à x si x est une constante ou un identificateur local.

Ce que l'on exprime par les axiomes suivants, dans lesquels $L_0^{P'}$ représente l'ensemble des identificateurs locaux à P'.

$$SH''_{10,1} = \{ poss\,G \equiv P' \supset sub\,A\,x \not\equiv y \mid G \in PRIM,\ P' \in FERMÉE;\ A = appel\,GL \in APPEL;\ x \in L_0^{P'};\ y \in EXT \text{ avec } y \neq sub\,A\,x \},$$

$$SH''_{10,2} = \{ poss\,G \equiv P' \supset sub\,A\,x \equiv x \mid G \in PRIM;\ P' \in FERMÉE;\ A = appel\,GL \in APPEL;\ x \in (EXT - (L_0^{P'} \cup \{ \sim_i \mid_i \geqq 1 \})) \}.$$

[Le cas de *sub* $A \sim_i$ est traité en (iii″)].

On exige également l'injectivité de *sub* :

$$SH''_{10,3} = \{ sub\,A\,x \equiv sub\,B\,y \supset x \equiv y \wedge A \equiv B \mid x,\ y \in L_0^{id};\ A,\ B \in APPEL \}.$$

Il faut également définir *sub* A u pour tout schéma fonctionnel u; en particulier on souhaite que l'axiome *poss* $A \equiv$ *sub* A *poss* P' associe à *poss* A le schéma déduit de *poss* P' en substituant *sub* $A\ x$ à tout identificateur x. C'est le rôle des axiomes

$$SH''_{10,4} = \{ sub\,A\,f\,u_1 \ldots u_n \equiv f\,sub\,A\,u_1 \ldots sub\,A\,u_n \mid f\,u_1 \ldots u_n \in \Sigma;\ A \in APPEL \}.$$

(iii″) *Traitement des paramètres effectifs.*

Il s'agit de remplacer les symboles $\sim_i$ de P' (s'ils existent) par les paramètres effectifs *correspondants* E_i. On utilise encore le symbole *sub* :

$$SH''_{10,5} = \{ sub\,A \sim_i \equiv E_i \mid 1 \leqq i \leqq n;\ A = appel\,GL;\ L \in LISTCOLL \cup LISTSER \cup PARFOR \text{ a } n \text{ éléments } E_1, \ldots, E_n \}.$$

On peut alors préciser la valeur de l'appel

$$SH''_{10,6} = \{ poss\,A \equiv sub\,A\ poss\ poss\,G \mid A = appel\,GL \in APPEL \}.$$

(iv) L'opérateur τ_A (§ 5.3.2, d) associé au schéma $SH''_{10.5}$ permet de « remplacer » $\sim_i$ par E_i dans les paramètres des schémas de modifications élémentaires constituant $\tilde{P}'$: $\tilde{A}$ sera défini à partir de $\tau_A(\tilde{P}')$.

D'autre part $\tilde{\sim}_i$ apparaît dans l'ensemble de calculs associé à $\tilde{P}'$, il doit être remplacé par $\tilde{E}_i$ dans l'ensemble de calculs $\tilde{A}$. Pour cela on complète la définition de l'opérateur τ_A en posant

$$\tau_A(\tilde{\sim}_i) = \tilde{E}_i \qquad \text{pour} \quad i = 1, \ldots, n.$$

Enfin il faut exprimer que, si *poss* $G \equiv P'$, alors l'appel $\tilde{A}$ contient l'ensemble de calculs $\tau_A(\tilde{P}')$. C'est ce que permet la fonction $\tilde{\psi}$ définie ci-dessous :

Il s'agit de préciser ũ pour tout schéma fonctionnel u formellement égal à un élément Q de PH; on souhaite alors que ũ $= \tilde{Q}$. Pour cela on introduit un schéma d'ensembles de calculs (§ 5.3.2 remarque 3), c'est-à-dire une appli-

cation $\tilde{\psi}$ de l'ensemble S des schémas fonctionnels dans l'ensemble des ensembles de calculs. $\tilde{\psi}$ est définie, pour tout programme Pr, par

$$\tilde{\psi}(\mathrm{u}) = \bigcup_{\substack{P' \text{ routine} \\ \text{du programme Pr}}} j(\mathrm{u}, P').\tilde{P}'; \qquad \mathrm{u} \in \mathrm{EXT}.$$

Cette équation se trouve dans le système associé à Pr (§ 6.2.12). Elle signifie que, si une information I contient le théorème $\mathrm{u} \equiv P'$, $\tilde{\psi}(\mathrm{u})$ est un ensemble de calculs dont un seul $(j(\mathrm{u}, P').\tilde{P}')$ conduit à une information consistante.

On peut alors définir $\mathscr{M}(A)$:

$$\begin{cases} \tilde{A} = \tilde{G}\tau_A(\tilde{\psi}(poss\, G)), \\ \tau_A(\tilde{\sim}_i) = \tilde{E}_i \qquad \text{pour} \quad i = 1, \ldots, n, \\ \mathscr{M}(G), \\ \mathscr{M}(E_i) \qquad i = 1, \ldots, n. \end{cases}$$

[En particulier, $\tilde{\psi}(poss\ G) = \tilde{\psi}(P')$ contient le calcul $\tilde{P}'$ si P' est la routine possédée par G.]

6.2.11. *Déclarations collatérales*

(i) $D = listcoll_n\, D_1\, D_2 \ldots D_n$ où $D_i \in \mathrm{DECID}$ pour $i = 1, \ldots, n$;

(ii) $\mathscr{T}(D) = \{(\mathscr{D}_1, \mathscr{D}_2, \ldots, \mathscr{D}_n) \mid \mathscr{D}_i \in \mathscr{T}(D_i) \text{ pour } i = 1, \ldots, n\}$;

(iii) aucun axiome ne caractérise D;

(iv)
$$\mathscr{M}(D)\begin{cases} \tilde{D} = \underset{1 \leqq i \leqq n}{m}(\tilde{D}_i), \\ \mathscr{M}(D_i), \qquad i = 1, \ldots, n. \end{cases}$$

6.2.12. *Programmes*

(i) $P = prog\ P_1$ où $P_1 \in \mathrm{FERMÉE} \cup \mathrm{COLL} \cup \mathrm{COND}$;

(ii) $\mathscr{T}(P)$ est définie à partir de $\mathscr{T}(P_1)$ en tenant compte des portées : $\mathscr{T}(P) = \mathscr{T}(P_1)$ si, pour toute phrase E de P (i. e. tout schéma fonctionnel contenu dans P) à laquelle est associée une certaine portée (élément de SER, de NOTPROC ou de GEN) la traduction concrète correspondante [qui est une occurrence $\mathscr{E}$ d'une certaine phrase de $\mathscr{T}(P_1)$] a la même portée dans $\mathscr{T}(P_1)$. Dans le cas contraire $\mathscr{T}(P) = \emptyset$. Précisons cette condition :

Il est possible de définir (*cf.* [6]) une fonction ξ_1 qui associe à un couple $(\mathscr{P}, \mathscr{E})$ où $\mathscr{P}$ est un programme de GOL et $\mathscr{E}$ l'occurrence d'une certaine région de GOL (proposition sérielle ou notation de routine) dans $\mathscr{P}$, la portée définie par $\mathscr{E}$ dans $\mathscr{P}$ (élément de PORTÉE).

De la même manière, on peut introduire une fonction ξ_2 associant au couple $(\mathscr{P}, \mathscr{E})$ où $\mathscr{E}$ est l'occurrence d'un générateur ou d'une notation de routine dans le programme $\mathscr{P}$, la portée de $\mathscr{E}$ dans $\mathscr{P}$ (dans le cas d'une notation de routine c'est la portée de la routine que possède $\mathscr{E}$).

Enfin on peut définir une fonction δ qui associe à tout schéma fonctionnel E contenu dans P_1 (nous écrirons $E \subset P_1$), l'occurrence $\mathscr{E}$ de $\mathscr{T}(E)$ qui lui correspond dans $\mathscr{T}(P_1)$.

Alors :

$$\mathscr{T}(P) = \{\mathscr{P} \in \mathscr{T}(P_1) \mid \forall E \subset P_1,\ E \in \text{SER},\ \textit{portée}\, E \equiv \xi_1(\mathscr{P}, \delta(E)) \in T$$
$$\text{et } \forall E \subset P_1,\ E = \textit{notproc}(L, \underline{\mu}, P_1, q, p) \in \text{NOTPROC}$$
$$q \equiv \xi_1(\mathscr{P}, \delta(E)) \in T \text{ et}$$
$$p \equiv \xi_2(\mathscr{P}, \delta(E)) \in T$$
$$\forall E \subset P_1,\ E \in \text{GEN},\ \textit{portée}\ E \equiv \xi_2(\mathscr{P}, \delta(E)) \in T\}$$

(T est l'ensemble des théorèmes de la S.I. de GOL).

(iii) Aucun axiome d'accès ne correspond à P : un programme ne possède pas de valeur.

(iv)
$$\begin{cases} \tilde{P} = \tilde{P}_1, \\ \tilde{\psi}(u) = \bigcup\limits_{\substack{P' \text{ routine} \\ \text{de } P}} j(u, P').\tilde{P}' \text{ pour } u \in \text{EXT (en fait pour } u \text{ de la} \\ \quad \text{forme } \textit{poss}\ G \text{ où } G \text{ possède une routine}), \\ \mathscr{M}(P_1). \end{cases}$$

6.2.13. *Propositions fermées*

(i) $P = \textit{parent}\ P_1$ où $P_1 \in \text{SER}$;
(ii) $\mathscr{T}(P) = \{\, \underline{\textit{début}}\ \mathscr{P}_1\ \underline{\textit{fin}} \mid \mathscr{P}_1 \in \mathscr{T}(P_1)\,\}$;
(iii) $\text{SH}_{13.1} = \{\, \textit{poss parent}\ P_1 \equiv \textit{poss}\ P_1 \mid P_1 \in \text{SER}\,\}$;
(iv)
$$\begin{cases} \tilde{P} = \tilde{P_1}, \\ \mathscr{M}(P_1). \end{cases}$$

6.2.14. *Propositions collatérales*

(i) $P = \textit{coll}\ (\textit{listcoll}_n\ P_1\ P_2\ \ldots\ P_n, \mu)$ où $P_i \in \text{UNIT}$ $(1 \leqq i \leqq n)$ et $\mu \in \text{GENRE}$;

(ii) $\mathscr{T}(P) = \{\, (\mathscr{P}_1, \mathscr{P}_2, \ldots, \mathscr{P}_n) \mid \mathscr{P}_i \in \mathscr{T}(P_i)\ \ 1 \leqq i \leqq n\,\}$;

(iii) La valeur que possède P n'est définie que si μ est de la forme $\underline{\textit{struct}}\ (\underline{\mu}_1\, x_1, \ldots, \underline{\mu}_n\, x_n)$, c'est alors un objet structuré :

$$\text{SH}''_{14.1} = \{\, \underline{\mu} \equiv \textit{struct}(\underline{\mu}_1\, x_1, \ldots, \underline{\mu}_n\, x_n) \supset$$
$$ch_{x_1}\, \textit{poss coll}\ (\textit{listcoll}_n\, P_1 \ldots P_n, \underline{\mu}) \equiv \textit{poss}\, P_1 \wedge \ldots \wedge$$
$$ch_{x_n}\, \textit{poss coll}\ (\textit{listcoll}_n\, P_1 \ldots P_n, \underline{\mu}) \equiv \textit{poss}\, P_n \mid$$
$$\underline{\mu} \in \text{GENRE};\ \underline{\mu}_i \in L_0^{\text{decl}},\ x_i \in L_0^{\text{id}},\ P_i \in \text{UNIT}$$
$$\text{pour } i = 1, \ldots, n\,\};$$

(iv) $\begin{cases} \tilde{P} = \underset{1 \leqq i \leqq n}{m} (\tilde{P}_i), \\ \mathscr{M}(P_i)\ i = 1, \ldots, n. \end{cases}$

6.2.15. *Propositions conditionnelles*

(i) P = *conditionnelle* $B\ P_1\ P_2$ où $B,\ P_1,\ P_2 \in \mathrm{SER}$;

(ii) $\mathscr{T}(P) = \{$ *si* $\mathscr{B}$ *alors* $\mathscr{P}_1$ *sinon* $\mathscr{P}_2$ *fsi* $\mid \mathscr{B} \in \mathscr{T}(B);\ \mathscr{P}_i \in \mathscr{T}(P_i)$ pour $i = 1, 2\}$;

(iii) $\mathrm{SH}_{15,1} = \{$ *poss conditionnelle* $B\ P_1\ P_2 \equiv$ *cond poss* B *poss vrai poss* P_1 *poss* $P_2 \mid B,\ P_1,\ P_2 \in \mathrm{SER}\}$;

(iv) $\begin{cases} \tilde{P} = \tilde{B}.(j(\textit{poss } B, \textit{ poss vrai}).\tilde{P}_1 \cup \bar{j}(\textit{poss } B, \textit{ poss vrai}).\tilde{P}_2), \\ \mathscr{M}(B), \\ \mathscr{M}(P_i)\ i = 1, 2. \end{cases}$

6.2.16 *Propositions sérielles*

(i) P = *serie* $(p,\ \textit{listser}_n\ P_1 \ldots P_n)$ où

$$p \in \text{PORTÉE} \quad \text{et} \quad P_i \in \text{UNIT}\ (1 \leqq i \leqq n).$$

En réalité, et pour traiter le problème de la structure de bloc, nous sommes conduits à restreindre l'ensemble des propositions sérielles.

Soit SER′ le sous-ensemble de SER défini par :

Pour toute phrase $P \in \mathrm{SER}'$, pour tout identificateur $x \in L_0^{\mathrm{id}}$, il existe au plus une déclaration de x dans P. Dans toute cette section 6 nous considérons en fait SER′ plutôt que SER.

Cette condition ne peut pas s'exprimer simplement dans le système à point fixe Σ, il y aurait cependant une autre manière de traiter le problème en utilisant le symbole *sub* dans le cas des propositions sérielles (de manière similaire à l'usage qui en est fait dans les appels de procédures (§ 6.2.10); c'est la démarche de [6].

(ii) $\mathscr{T}(P) = \{\mathscr{P}_1;\ \mathscr{P}_2; \ldots;\ \mathscr{P}_n \mid \mathscr{P}_i \in \mathscr{T}(P_i)$ pour $i = 1, \ldots, n\}$;

(iii) $\mathrm{SH}_{16,1} = \{$ *ppq p portée poss* $P_n \equiv$ *poss vrai* $\supset$ *poss serie* $(p,\ \textit{listser}_n\ P_1 \ldots P_n) \equiv$ *poss* $E_n \mid$ $p \in \text{PORTÉE};\ P_i \in \text{UNIT}\ (1 \leqq i \leqq n)\}$.

(On exprime que la valeur de P est celle de P_n, sous réserve que la portée de P_n soit supérieure à celle de P) :

$$\mathrm{SH}''_{16,2} = \{\textit{portée serie}(p,\ \textit{listser}_n P_1 \ldots P_n) \equiv p \mid p \in \text{PORTÉE};\ P_i \in \text{UNIT}(1 \leqq i \leqq n)\}$$

(utilisé lors de la traduction d'un programme complet).

(iv) $\begin{cases} \tilde{P} = \tilde{P}_1.\tilde{P}_2 \ldots \tilde{P}_n, \\ \mathscr{M}(P_i)\ i = 1, \ldots, n. \end{cases}$

décembre 1976.

7. CONCLUSION

Les composantes de notre méthode de formalisation de la sémantique d'un langage $\mathscr{L}$, dont on connaît la syntaxe concrète, sont donc :

définition de la fonction sémantique :
- définition d'un langage pivot $\mathscr{L}_0$ (sous forme de schémas fonctionnels);
- définition des objets internes (schémas fonctionnels) et de leurs propriétés (schémas d'axiomes propres);
- définition d'un ensemble de modifications élémentaires associées à certaines phases de $\mathscr{L}_0$;
- association de leurs valeurs internes à certaines phrases de $\mathscr{L}_0$ par un ensemble d'axiomes d'accès;
- association d'un ensemble de calculs à toute phrase de $\mathscr{L}_0$ par un système à point fixe;
- définition d'une fonction de traduction concrète.

définition de la S.I. du langage

Dans l'introduction, nous avons présenté plusieurs motivations à une définition précise de la sémantique d'un langage; en particulier nous avons remarqué que la formalisation peut dépendre des buts poursuivis. C'est ainsi qu'apparaît un conflit entre la précision et la simplicité de la description d'un langage.

L'intérêt de notre méthode provient de l'utilisation de deux points de vue formalisant deux réalités informatiques complémentaires :

– point de vue statique (ou encore dénotationnel ou logique) : axiomes, modifications;

– point de vue dynamique (ou opérationnel) : calculs.

On peut ainsi penser que ce type de définition permettra d'atteindre la plupart des buts visés :

– elle est certainement utilisable par l'implémenteur car elle est basée sur la notion de calculs et la théorie des S.I. ([5], [19]) permet d'introduire la notion de représentation d'une S.I. par une autre, cette dernière pouvant être une structure de mémoire. La définition de cette représentation, ainsi que celle d'une traduction abstraite forment la définition d'une implémentation;

– elle doit permettre de construire des méthodes de preuves de correction, de terminaison ou de définir des notions d'équivalences concernant des langages évolués existants : l'interprétation d'un calcul fini comme une modification conduit à la notion de sémantique dénotationnelle.

Cependant pour rendre la méthode plus utilisable, notamment par les programmeurs, il conviendrait d'introduire un langage plus commode pour

exprimer les diverses parties d'une définition sémantique et en particulier celle des systèmes de calculs.

Je remercie Messieurs C. Pair et E. Engeler pour leurs suggestions et critiques pertinentes concernant cet article, ainsi que J. L. Rémy pour son travail sur les structures d'information.

EXTRAIT DE LA BIBLIOGRAPHIE
(*cf.* partie I)

5. J.-P. FINANCE, *Contribution à la Formalisation de la Sémantique d'un Langage de Programmation.* Application à Algol 68, Thèse de 3e Cycle, Université de Nancy 1, 1974.
6. GROUPE ALGOL DE L'AFCET, *Définition du Langage Algorithmique Algol* 68 (traduction), Hermann, 1972.
19. J. L. RÉMY, *Structure d'Information, Formalisation des Notions d'Accès et de Modification d'une Donnée*, Thèse de 3e Cycle, Université de Nancy 1, 1974.
23. A. VAN WINJNGAARDEN (*ed.*), B. J. MAILLOUX, J. E. L. PECK et C. H. A. KOSTER, *Report on the Algorithmic Language Algo* 68, Mathematisch Centrum, Amsterdam, MR 101, 1969.

décembre 1976.