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° R2 (1976), p. 5-32

<http://www.numdam.org/item?id=ITA_1976__10_2_5_0>

R.A.I.R.O. Informatique Théorique
(vol. 10, n° 8, août 1976, p. 5 à 32)

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

Première partie

par J.-P. FINANCE [1]

Communiqué par E. Engeler.

Résumé. — *Cet article présente une méthode de formalisation de la sémantique basée sur les notions de structure d'information et de calculs. Une structure d'information est un cadre permettant de définir les concepts d'information (qui formalise l'idée intuitive d'état de mémoire) et de modification élémentaire d'une information (qui permet de rendre compte des instructions élémentaires d'un langage). Une information est un ensemble de théorèmes d'un certain système formel associé au langage. Un calcul est une suite de modifications élémentaires; il traduit une élaboration d'un programme (sémantique opérationnelle) et peut être interprété, s'il est fini, comme une modification « composée » (sémantique dénotationnelle). On illustre cette méthode sur un sous-ensemble d'Algol 68.*

1. INTRODUCTION

Preuves de programmes, équivalence de programmes, synthèse de programmes... Autant de problèmes préoccupant les informaticiens qui ont actuellement le souci d'écrire (et de donner les moyens d'écrire) des programmes « corrects », c'est-à-dire des programmes qui « font » ce que leurs auteurs attendent d'eux. Une telle « définition » demande à être précisée : qu'entend-t-on par « ce que fait un programme »? C'est toute la question de la sémantique des langages de programmation : comment définir précisément le *sens* d'un programme?

La réponse à cette question doit permettre :

a) au théoricien de résoudre les problèmes cités plus haut et en particulier celui des preuves de correction de programmes;

b) à l'implémenteur de connaître sans ambiguïté le langage sur lequel il travaille;

c) au programmeur de maîtriser parfaitement le langage qu'il utilise en évitant ainsi la déplorable pratique actuelle qui consiste à vérifier ce que spécifie un langage par des passages successifs sur machine.

(*) Reçu mai 1975 (version remaniée reçue le 23 février 1976).

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

Il semble cependant que certains de ces objectifs soient difficilement conciliables sinon contradictoires : peut-on donner une définition formelle complète (*b*) permettant des preuves (*a*), qui soit simple et lisible par l'homme (*c*) ? Un moyen de répondre à cette objection, proposé par Hoare et Lauer [9] consiste à donner plusieurs définitions « équivalentes » et « complémentaires » (en un sens à préciser) de la sémantique d'un langage, chacune d'elles concernant un point de vue particulier : preuves, utilisation, implémentation.

Intuitivement, définir la sémantique d'un langage c'est *associer* un *sens* aux différents programmes de ce langage, on dit encore lui associer une *valeur sémantique*. Le problème est donc double : il faut à la fois décrire les valeurs sémantiques et l'association d'une valeur à un programme.

Dans cet article, nous introduisons une méthode de définition de la sémantique d'un langage de programmation reposant sur le concept de structure d'information (qui formalise la notion d'état de mémoire) et dans laquelle on sépare les notions « statiques » (liées aux propriétés des objets manipulés) des notions « dynamiques » (qui correspondent aux différentes compositions des calculs).

On peut penser que le cadre formel utilisé permette d'atteindre les objectifs mentionnés ci-dessus; cependant, il reste encore à trouver un langage plus agréable pour exprimer les différentes composantes de la sémantique.

Dans cette première partie, nous présentons rapidement et intuitivement au paragraphe 2 [noté (§ 2)] les idées à la base de notre approche, puis nous formalisons les concepts de structure d'information (§ 3) et de langage pivot (§ 4) en les illustrant sur un sous ensemble d'Algol 68 : GOL.

Dans la deuxième partie, on précise la notion de calcul (§ 5), ce qui permet de définir la sémantique de GOL (§ 6).

2. COMPOSANTES D'UNE DÉFINITION DE LA SÉMANTIQUE

Brièvement, un programme exprime des *traitements* portant sur des *valeurs*. Précisons ces deux notions.

2.1. Structure d'information

En première approximation, l'ensemble des valeurs accessibles à un instant donné de l'exécution d'un programme peut être considéré comme une suite finie de nombres : c'est la notion de *vecteur d'état* introduite par McCarthy [14] pour rendre compte de l'idée intuitive d'état de mémoire.

Cependant, les informations manipulées par un programme peuvent être beaucoup plus complexes que de simples nombres (tableaux, pointeurs, listes...); une formalisation de la sémantique qui rend vraiment compte des langages évolués sans les ramener au niveau machine nécessite donc une théorie des structures d'informations (en abrégé S.I.) [16, 19].

Intuitivement une S.I. permet de rendre compte des caractéristiques des données sur lesquelles, par exemple, travaille un langage de programmation. Brièvement une *donnée* sera formée d'un ensemble d'objets et de fonctions (encore appelées *accès*) liant ces objets.

Exemple 1 : A la suite de l'élaboration du début de programme Algol 68 :

début réel $x = 3.2$; *rep réel* $y =$ *loc réel* $:= 4.1$; *compl* $z = (x, y)$.

La donnée D obtenue peut être schématisée par la figure 1 : possède, repère, re, im sont des accès unaires (c'est-à-dire à une variable); x, y et z peuvent être considérés comme des accès 0-aires (plus généralement il en sera ainsi pour tout accès désignant un objet d'une donnée tel que possède (y), re (possède (z)) etc.).

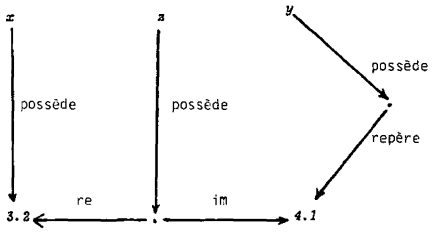

Figure 1.
Exemple de données.

L'élaboration d'une proposition dans la suite de ce programme provoquera l'introduction de nouveaux accès ou la modification d'anciens accès : nous introduisons ainsi la notion de *modification* de la S.I. qui fait passer d'une donnée à une autre.

Exemple 2 : Appelons affect (y, 5.4) la modification associée à l'affectation $y := 5.4$; l'image de D (exemple précédent) par cette modification peut être représentée par la figure 2 :

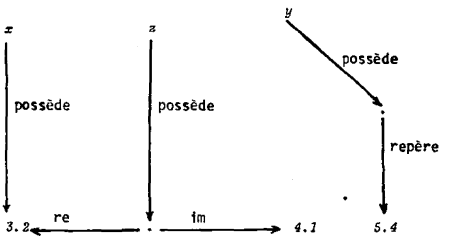

Figure 2.
Donnée de la figure 1 après modification.

On peut dire que cette modification est, en première approximation, la signification de la phrase $y := 5.4$. En réalité, en Algol 68, cette phrase

a aussi une valeur : l'objet possède (y). La signification de $y := 5.4$ est donc constituée à la fois de la modification affect (y, 5.4) et de l'accès 0-aire possède (y).

De même l'expression $x - y \times z$ a une signification formée de la modification identité et de l'accès 0-aire moins (x, mul (y, z)) obtenu en composant les accès moins, mul, x, y, et z.

Une modification telle que affect (y, 5.4) sera qualifiée d'élémentaire en ce sens qu'elle est associée à une phrase élémentaire du langage. Toute S.I. comporte un ensemble $\mathcal{M}$od de modifications élémentaires à partir desquelles sont définies, par composition, les modifications de la structure.

2.2. Calculs

Un traitement décrit par un programme peut se concevoir comme la suite d'états, i. e. de données, par lesquels passe le calculateur pendant l'exécution du programme. Une telle suite peut s'appeler un *calcul*; parallèlement à une théorie des structures d'information (et s'appuyant sur elle) une théorie des calculs est donc nécessaire [3, 17].

Plus précisément, ici, il est naturel de définir un calcul comme étant une suite (finie ou infinie) de modifications élémentaires de la S.I. faisant passer d'un état initial à un état final (si la suite est finie).

Exemple 3 : En reprenant l'exemple 1 ci-dessus et en notant ident (a, b) la modification élémentaire associée à la déclaration d'identité $\mu\, a = b$ où μ est un déclareur de mode quelconque, nous représenterons l'élaboration du début de programme par le calcul :

$$\text{ident } (x, 3.2).\text{ident } (y, \textit{loc réel}).\text{affect } (y, 4.1).\text{ident } (z, (x, y)).$$

On peut considérer ce calcul comme la valeur sémantique « opérationnelle » de la phrase considérée.

On peut aussi interpréter cette suite « formelle » comme une modification en interprétant la concaténation comme la composition des modifications élémentaires (dans l'ordre de leur écriture). On obtient ainsi une définition « dénotationnelle » de la signification d'un programme comme une modification faisant passer d'une donnée initiale à une donnée finale. Le premier point de vue incluant en quelque sorte le second, nous définirons la valeur sémantique d'un programme comme étant un calcul, ou plus précisément un ensemble de calculs : en effet à tout programme sont associées en général plusieurs élaborations possibles.

2.3. Valeur et fonction sémantique

En regroupant les notions introduites aux paragraphes précédents, nous définissons la valeur sémantique d'une phrase d'un langage de programmation comme un couple formé d'un ensemble de calculs (qui peut se réduire à une

modification élémentaire) et d'un accès 0-aire de la S.I. (valeur de la phrase au sens d'Algol 68). Cet accès n'est pas défini si la phrase n'a pas de valeur (c'est le cas d'un programme ou d'une déclaration).

Caractérisons maintenant l'association de sa valeur sémantique à toute phrase, c'est-à-dire précisons le mode de définition de la fonction sémantique :

L'ensemble de calculs associé à une phrase dépend de ceux qui sont associés aux phrases qui la composent. On désire indiquer, pour chaque type de phrase, comment est construit cet ensemble. Cela conduit à obtenir l'ensemble de calculs d'un programme par un système d'égalités, ou plus exactement, si le langage admet la récursivité, par un système à point fixe (système de calculs).

De la même manière, la valeur d'une phrase dépend de celles de certaines de ses phrases composantes; elle est définie par un système d'égalités entre accès 0-aires (axiomes d'accès).

2.4. Langage pivot

L'étude précédente, et en particulier la définition récursive des systèmes associés à une phrase, invite à travailler sur l'arbre syntaxique associé à un programme plutôt que sur la chaîne de caractères qui en forme de texte. Cet arbre permet de s'affranchir de la manière dont sont écrites les phrases (symboles de ponctuation, mots réservés...) mais surtout il exprime que la signification d'une phrase ne dépend pas que d'elle-même, mais d'un certain contexte contenant des renseignements « globaux » (portées, modes...). De tels renseignements peuvent être placés dans l'arbre par des mécanismes tels que les doubles grammaires de Van Wijngaarden [23] ou les attributs de Knuth [10].

Cette distinction entre représentation arborescente et représentation linéaire a été proposée par McCarthy [14] et suivie par les auteurs de la méthode de Vienne [13] qui opposent la notion de syntaxe abstraite à celle de syntaxe concrète.

Ainsi, pour définir la sémantique d'un langage de programmation $\mathscr{L}$ nous commençons par définir un *langage pivot* $\mathscr{L}_0$ (dont les « phrases » sont des « arbres ») et une application (traduction concrète) de $\mathscr{L}_0$ dans $\mathfrak{P}(A^*)$ où A est l'alphabet de $\mathscr{L}$. Ainsi chaque phrase de $\mathscr{L}_0$ possède zéro, une ou plusieurs représentations « concrètes ». Remarquons que c'est la manière dont certains linguistes tels que Chomsky [4] conçoivent la définition d'une langue naturelle en définissant la structure de surface (concrète) et la sémantique à partir de la structure profonde (abstraite) du langage.

La sémantique du langage sera alors définie à partir de son langage pivot (*fig.* 3).

août 1976.

REMARQUE : La construction d'un compilateur exige la recherche d'un algorithme définissant la fonction qui peut être considérée intuitivement comme la relation réciproque de la traduction concrète (traduction abstraite).

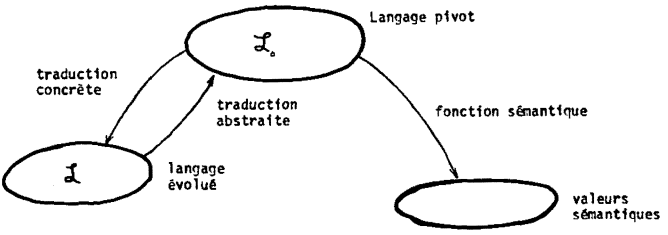

Figure 3.
Langage pivot et sémantique.

2.5. Conclusion

En résumé, pour nous, formaliser la sémantique d'un langage de programmation consiste à définir :

a) un ensemble de valeurs sémantiques :

— accès;

— calculs.

Cet ensemble se déduit de la structure d'information du langage (qui caractérise accès et modifications élémentaires); cette notion est précisée au (§ 3) :

b) un langage pivot et un procédé de traduction [envisagé au (§ 4)];

c) une fonction sémantique caractérisée par l'association à chaque phrase du langage pivot :

— d'un système à point fixe définissant un ensemble de calculs de la S.I. (§ 5);

— d'axiomes définissant un accès de la S.I. (§ 4).

2.6. Comparaison avec d'autres méthodes

Parmi les principales méthodes de définition formelle de la sémantique d'un langage évolué on peut distinguer :

— les méthodes interprétatives ([13, 2]);

— les méthodes compilatoires et fonctionnelles ([12, 14, 15, 1, 3, 22]);

— les méthodes axiomatiques ([7, 8]).

La méthode que nous proposons ici se rapproche de la méthode de Vienne [13] en ce qu'elle utilise explicitement la notion de S.I. Cependant, plutôt que de définir un interprète abstrait, elle associe à un programme un

ensemble de calculs représentant toutes les élaborations possibles. Comme les méthodes fonctionnelles elle utilise la notion de point fixe développée en particulier par Scott [20] pour définir les valeurs sémantiques; elle en est cependant éloignée par l'utilisation de calculs définis à partir d'une S.I.

2.7. Le langage GOL

Dans la suite nous précisons notre formalisation et nous l'illustrons sur un sous-langage d'Algol 68 que nous appellerons GOL.

La syntaxe de GOL est décrite en annexe 1; indiquons simplement ici que ce langage ne contient pas de réels, de caractères, de valeurs multiples, de déclarations et d'unions de modes, de sémaphores ni d'entrées/sorties. De plus GOL est un sous-ensemble du langage strict d'Algol 68 : aucune extension (contraction d'écriture, instruction d'itération...) n'est admise. Enfin GOL n'admet pas d'instruction de saut et donc aucun de ses programmes ne contient d'étiquette.

Une présentation plus détaillée de notre méthode est donnée en [5] où l'on définit la sémantique d'Algol 68 complet (sauf les sémaphores et les entrées/sorties).

3. STRUCTURE D'INFORMATION

3.1. Donnée

Le rapport Algol 68 ([6, 23]) « explique la signification d'un programme en faisant appel à un calculateur hypothétique » qui « traite un ensemble d'objets entre lesquels, à tout moment, certaines relations peuvent être vraies ». Cet ensemble d'objets et de relations représente l'état de mémoire, à un instant donné de l'exécution du programme, du « calculateur hypothétique » d'Algol 68. [L'état de mémoire de la « machine Algol 68 » après élaboration du programme de l'exemple 1 (§ 2.1) est schématisé par la figure 1].

Nous formalisons la notion d'état de mémoire par celle de *donnée* qui est constituée d'objets élémentaires (appartenant à certains ensembles : identificateurs, noms, valeurs structurées...) liés par des fonctions d'accès (possède, repère...). L'opération essentielle sur ces fonctions est la composition qui, à partir d'accès élémentaires, construit des accès plus élaborés. C'est ce que traduit la définition suivante.

DÉFINITION 1 : Une *donnée* est un $(m+1)$-uplet $(E_1, ..., E_m, \bar{L})$ où E_i $(i = 1, ..., m)$ est un ensemble d'objets élémentaires; $\bar{L}$ est un ensemble de fonctions (partielles) définies dans des ensembles de la forme $E_{j_1} \times ... \times E_{j_q}$ $(q \geqq 0$ et $1 \leqq j_i \leqq m)$, à valeurs dans l'un des $E_i (1 \leqq i \leqq m)$; une telle fonction s'appellera accès élémentaire à q variables, une fonction à 0 variables étant confondue avec sa valeur.

août 1976.

Une donnée ainsi définie est un objet « statique »; elle n'a d'intérêt que dans la mesure où l'on peut lui appliquer des traitements. Ainsi dans l'exemple 1 (§ 2.1.) on veut pouvoir définir ce qu'est l'affectation de *5.4.* au nom possédé par *y*, ce qu'est la déclaration d'identité *réel* $t = 2.5$. De telles transformations (nous dirons *modifications*) consistent à « changer » les fonctions repère et possède; elles associent à une donnée d'un certain « type » une donnée du même type. Un type est, de manière intuitive, une classe de données ayant même entier m et des ensembles $\bar{L}$ « semblables ».

Nous regrouperons ces notions de type et de modifications sous le terme de *structure d'information* : c'est un cadre dans lequel on peut exprimer certaines propriétés caractérisant certains objets.

3.2. Structure d'information

La notion de donnée, si elle est naturelle, ne permet pas d'exprimer agréablement ce qu'est une modification : un « changement de fonction d'accès » n'est pas simple à exprimer. Aussi est-on conduit à introduire des objets plus « formels » en remplaçant les fonctions par des symboles fonctionnels, la notion de schéma fonctionnel rendant compte de la composition des fonctions. On remplace ainsi la notion de donnée par celle *d'information* : ce sera un ensemble de théorèmes portant sur des schémas fonctionnels.

Exemple 1 : L'information représentant l'état de mémoire après l'élaboration du début de programme de l'exemple 1 (§ 2.1) contient, entre autres, les théorèmes :

$$re\ poss\ z \equiv 3.2$$
$$rep\ poss\ y \equiv im\ poss\ z$$

où *re*, *im*, *rep*, *poss* sont des symboles fonctionnels interprétés comme les fonctions re, im, repère, possède.

Une structure d'information est donc d'abord un cadre pour ces théorèmes. Ce sera un système formel $\mathscr{F} = (\text{Alph}, F, X, R)$ [21] exprimant les propriétés générales communes à toutes les informations du type (Alph est l'alphabet de $\mathscr{F}$, F l'ensemble des formules, X l'ensemble des axiomes et R l'ensemble des règles d'inférence). Pour obtenir une structure d'information, nous ajouterons ultérieurement un ensemble $\mathscr{M}od$ de modifications élémentaires.

Dans la suite on définit successivement les différents composants de $\mathscr{F}$.

3.3. Alphabet et ensemble de formules du système formel d'une S. I.

3.3.1. *L'alphabet Alph contient :*

— un ensemble L de symboles fonctionnels;
— le symbole $\equiv$ (égalité formelle);
— les symboles $\neg$, $\supset$, (,) (connecteurs logiques et parenthèses).

3.3.2. *L'ensemble des formules* F est un sous-ensemble de Alph* construit en plusieurs étapes.

On définit successivement :

— un sous-ensemble S de L^* (ensemble des schémas fonctionnels sur $L[11]$);

— un sous-ensemble A de $S \equiv S$ (ensemble des formules atomiques);

— l'ensemble F solution unique de l'équation à point fixe

$$F = A \cup \neg F \cup (F \supset F);$$

a) Pour définir *l'ensemble S des schémas fonctionnels* nous sommes conduits à :

— donner un entier m;

— associer à chaque élément f de L deux entiers q et k tels que $q \geqq 0$ et $1 \leqq k \leqq m$, ainsi qu'un ensemble pl (f) de $(q+1)$-uplets $(j_1, ..., j_q, k)$ avec $1 \leqq j_i \leqq m$ pour $1 \leqq i \leqq q$.

NOTATIONS : pl (f) est appelé *profil de f* et q est *son arité*; Lq est l'ensemble des symboles fonctionnels q-aires $(q \geqq 0)$ (Intuitivement, pour interpréter les symboles de L on doit introduire m ensembles $E_1, ..., E_m$ et interpréter f comme une fonction de $\bigcup E_{j_1} \times ... \times E_{j_q}$ dans E_k, la réunion étant prise sur les $(j_1, ..., j_q, k) \in \text{pl}(f)$). En tenant compte des profils dans la concaténation des symboles fonctionnels nous sommes conduits à :

DÉFINITION 2 : L'ensemble S des schémas fonctionnels compatibles avec les profils est défini par

$$S = \bigcup_{1 \leqq k \leqq m} S_k$$

où $(S_1, \ldots, S_m)$ est la solution unique du système Σ :

$$S_k = \bigcup_{q \geqq 0} \{ f S_{j_1} S_{j_2} \ldots S_{j_q} \mid f \in L_q; (j_1, j_2, \ldots, j_q, k) \in \text{pl}(f) \}$$

pour $1 \leqq k \leqq m$.

REMARQUE : Il est équivalent d'écrire le système Σ définissant les S_k ou de définir m et pl. C'est Σ que nous définissons dans le cas de la S.I. de GOL.

b) *L'ensemble A des formules atomiques* est défini par

$$A = \bigcup_{1 \leqq k \leqq m} S_k \equiv S_k$$

(on ne compare deux schémas

$$f \mathrm{u}_1 \ldots \mathrm{u}_q \qquad \text{et} \qquad f' \mathrm{u}_1 \ldots \mathrm{u}'_{q'}$$

avec

$$(j_1, \ldots, j_q, k) \in \text{pl}(f) \quad \text{et} \quad (j'_q, \ldots, j'_{q'}, k') \in \text{pl}(f') \quad \text{que si } k = k')$$

août 1976.

NOTATION : Afin d'abréger l'écriture de certaines formules nous introduirons les connecteurs logiques $\vee$ et $\wedge$ avec leur sens habituel et nous adopterons les conventions habituelles de suppression de parenthèses. En particulier, l'absence de parenthèse dans une formule ne contenant qu'un seul type de connecteur, correspond à un ordre de priorité de gauche à droite. Enfin pour toutes formules atomiques u et v, $u \not\equiv v$ signifiera $\neg u \equiv v$.

3.3.3. *Exemple de GOL :*

En GOL, comme en Algol 68 [6, § 2.2] on distingue les objets externes (accessibles au programmeur) des objets internes. Plus précisément, pour rendre compte du fait que certaines phrases du langage pivot GOL_0 (identificateurs, constantes, propositions) possèdent une valeur, nous sommes conduits à les introduire dans la S.I. en les distinguant des autres objets (internes). Pour cela nous définissons l'ensemble S des schémas fonctionnels de la structure d'information de GOL en deux étapes :

— au cours de la première (dans cette section 3) nous précisons l'ensemble L' des symboles fonctionnels permettant, par composition entre eux et les éléments d'un ensemble PH (qui est, intuitivement, l'ensemble des phrases GOL_0) d'obtenir les schémas fonctionnels interprétables comme des objets internes;

— ensuite (§ 4) nous définissons PH comme l'ensemble des phrases de GOL_0. Nous verrons alors que $m = 29$ et nous pourrons caractériser complètement les schémas fonctionnels de GOL.

a) Soit donc $L' = \bigcup_{q \in \{0,1,2,4\}} L'_q$ l'ensemble des symboles fonctionnels de la S.I. de GOL qui ne concernent pas les phrases. Définissons successivement les différents ensembles L'_q :

— $L'_0 = \{\theta\}$; θ est le symbole 0-aire interprétable comme la portée d'un programme complet;

— L'_1 est formé de :

• $poss^i$ $(1 \leqq i \leqq 7)$ symboles qui peuvent être interprétés comme la « fonction possède »; il y a plusieurs tels symboles (d'où l'indice i) pour rendre compte des différents « domaines d'arrivée » possibles,

• rep^i $(2 \leqq i \leqq 6)$ représentent la « fonction repère »,

• ch^i_x $(2 \leqq i \leqq 6$; $x \in L_0^{id}$ (sous-ensemble de PH formé des identificateurs)) sélecteurs de champ x de toute valeur structurée,

• *portée* qui associe à toute valeur interne sa portée,

• *succ* et *desc* qui permettent de définir les portées à partir de la portée primitive θ;

— L'_2 contient :

• *inf* fonction à valeurs booléennes rendant compte de la relation d'ordre « plus petit que » sur les entiers,

• *moins* qui associe à 2 entiers leur différence,

• *ppq* qui permet de définir une relation d'ordre partiel sur l'ensemble des portées,

• sub^i $(1 \leqq i \leqq 6)$ qui permet de formaliser les appels de routines;

— L'_4 ne contient que les symboles $cond^i$ $(3 \leqq i \leqq 7)$ qui permettent de définir la valeur d'une proposition conditionnelle;

b) Définissons maintenant l'ensemble Σ' des équations du système Σ qui caractérisent les schémas fonctionnels interprétables comme des objets internes. L'ensemble $\Sigma'' = \Sigma - \Sigma'$ des équations caractérisant les phrases de GOL_0 (éléments de PH) est défini en (§ 4.2.2).

Dans toute la définition de Σ nous préférons utiliser une notation plus intuitive que S_k pour désigner les schémas de type k $(1 \leqq k \leqq 29)$, par exemple NOM sera préféré à S_2 pour désigner les schémas fonctionnels interprétables comme des noms.

Précisons Σ' et reportons à la suite les commentaires portant sur les diverses équations :

(1) EXT = PH $\cup$ sub^1 EXT EXT $\cup$ $poss^1$ EXT

(2) NOM = $poss^2$ EXT $\cup$ rep^2 NOM $\cup$ [$\bigcup_{x \in L_0^{id}}$ ch_x^2 (NOM $\cup$ STRU)] $\cup$ sub^2 EXT NOM

(3) STRU = $poss^3$ EXT $\cup$ rep^3 NOM $\cup$ ($\bigcup_{x \in L_0^{id}}$ ch_x^3 STRU) $\cup$ sub^3 EXT STRU $\cup$ $cond^3$ VALEUR $^{(2)}$ STRU VALEUR $\cup$ $cond^3$ VALEUR $^{(3)}$ STRU

(4) ENT = $poss^4$ EXT $\cup$ rep^4 NOM $\cup$ ($\bigcup_{x \in L_0^{id}}$ ch_x^4 STRU) $\cup$ sub^4 EXT ENT $\cup$ *moins* ENT ENT $\cup$ $cond^4$ VALEUR$^{(2)}$ ENT VALEUR $\cup$ $cond^4$ VALEUR$^{(3)}$ ENT

(5) BOOL = $poss^5$ EXT $\cup$ rep^5 NOM $\cup$ ($\bigcup_{x \in L_0^{id}}$ ch_x^5 STRU) $\cup$ sub^5 EXT BOOL $\cup$ *ppq* PORTÉE PORTÉE $\cup$ *inf* ENT ENT $\cup$ $cond^5$ VALEUR$^{(2)}$ BOOL VALEUR $\cup$ $cond^5$ VALEUR$^{(3)}$ BOOL

(6) PROC = EXT $\cup$ $poss^6$ EXT $\cup$ rep^6 NOM $\cup$ ($\bigcup_{x \in L_0^{id}}$ ch_x^6 STRU) $\cup$ sub^6 EXT PROC $\cup$ $cond^6$ VALEUR$^{(2)}$ PROC VALEUR $\cup$ $cond^6$ VALEUR$^{(3)}$ PROC

(7) NOM 1 = NOM $\cup$ $poss^7$ *nil* $\cup$ $cond^7$ VALEUR$^{(2)}$ NOM 1 VALEUR $\cup$ $cond^7$ VALEUR$^{(3)}$ NOM 1

(8) PORTÉE = $\{\theta\}$ $\cup$ PORTÉE 1 $\cup$ *portée* VALEUR $\cup$ *portée* PH

(9) PORTÉE 1 = *succ* PORTÉE 1 $\cup$ *desc* PORTÉE

où VALEUR = ENT $\cup$ BOOL $\cup$ NOM 1 $\cup$ PROC $\cup$ STRU

août 1976.

Ainsi

$$rep^2\ poss^2\ x,$$

$$moins\ ch_x^4\ rep^3\ rep^2\ poss^2\ y\ poss^4\ 5,$$

$$ppq\ portée\ ch_x^3\ poss^3\ y\ succ\ desc\ \theta$$

sont des exemples de schémas fonctionnels définis par Σ (x, y, 5 étant des éléments de PH). Pour améliorer la lisibilité il nous arrivera d'introduire des parenthèses dans de tels schémas.

Commentaires : 1° Les objets internes sont obtenus à partir d'objets externes par composition d'accès tels que *poss*, *rep* ... C'est la raison pour laquelle l'ensemble EXT figure dans Σ'. L'ensemble des objets externes « formels » contient, outre les phrases éléments de PH, des schémas fonctionnels permettant de rendre compte des procédures :

sub^1 et $poss^1$ permettent respectivement de rendre compte des variables locales à une procédure et de l'objet possédé par une notation de procédure (§ 6.2.9). Ils ont pour profils (1, 1, 1) (*) et (1, 1).

2° L'équation (2) définit les schémas fonctionnels interprétables comme des noms. On exprime ainsi qu'un nom peut être possédé par un objet externe, repéré par un autre nom ou composant d'une valeur structurée. Les schémas ch_x^2 NOM permettront d'exprimer [6, § 2.2.3.5 *b*)] que tout nom de valeur structurée permet d'accéder à des sous-noms (voir le schéma d'axiomes $SH_{1,1}$). Le dernier ensemble de schémas fonctionnels permet de rendre compte des appels de procédures. Remarquons que le profil de ch_x^2 est $\{(2,2), (3,2)\}$. Dorénavant nous ne préciserons plus explicitement ces profils.

3° Les autres équations permettent de définir les schémas fonctionnels interprétables respectivement comme des structures, des booléens, des procédures, des noms, des portées (on définit en particulier la portée d'une valeur interne, ainsi que la portée de certaines phrases). VALEUR $^{(i)}$ représente la concaténation de i occurences de VALEUR. D'autre part l'introduction de cet ensemble n'est qu'une simple abréviation d'écriture, c'est la raison pour laquelle son équation ne fait pas partie de Σ (elle n'est pas numérotée). Notons enfin que *nil* est un élément de L_0'' défini en (§ 4.2.1).

REMARQUES : GOL ne contient pas de déclarations de modes, aussi les tests sur les modes figurant dans les définitions des différentes phrases élémentaires (déclarations d'identité, affectations...) peuvent-ils s'exprimer de manière purement syntaxique. Ainsi, contrairement à [5], nous n'introduisons pas « l'ensemble des modes » dans la S.I.;

(*) Lorsque pl (f) ne contient qu'un $(q + 1)$ - uplet on convient d'omettre les accolades, ainsi (1, 1, 1) représente { (1, 1, 1) }.

— nous n'introduisons pas non plus la notion d'exemplaire de valeur : cette notion est surtout utile lors de la définition de l'affectation [6, § 8.3.1.2.], essentiellement pour permettre l'ajustement des bornes flexibles. Ici la définition de l'affectation sera très simple (§ 3.8.4. et 6.2.4);

— la distinction entre NOM et NOM1 permet d'exprimer le fait *nil* que ne repère aucune valeur [6, § 2.2.3.5 *a*].

c) L'ensemble des formules atomiques de GOL est donné par :

$$A = \bigcup_{1 \leqq k \leqq 29} S_k \equiv S_k \qquad \text{(avec } S_1 = \text{EXT}, S_2 = \text{NOM etc.)}.$$

Dans la suite, et pour alléger les notations, nous confondrons les symboles fonctionnels analogues en supprimant les indices qui distinguent $poss^1$, $poss^2 \ldots$; rep^2, rep^3 etc.

Ainsi, *rep poss* $x \equiv$ *moins poss3 possy*; $(u \equiv v \supset cond\ u\ v\ x\ y \equiv x)$, $[(u_1 \equiv u_2 \supset u_1 \equiv u_3) \supset u_2 \equiv u_3]$ sont des exemples de formules de GOL.

3.4. Axiomes et règles d'inférence du système formel d'une S.I.

3.4.1. *L'ensemble X des axiomes* est un sous ensemble de F

On distingue :

— des axiomes logiques, valables pour toute structure d'information : nous noterons X_l leur ensemble;

— des axiomes propres à la structure considérée, soit X_p leur ensemble.

Les axiomes logiques sont :

— Les axiomes du calcul des propositions, regroupés en trois schémas :

$$SL_1 = \{\varphi \supset (\psi \supset \varphi) \mid \varphi, \psi \in F\},$$
$$SL_2 = \{(\varphi \supset (\psi \supset \rho)) \supset ((\varphi \supset \psi) \supset (\varphi \supset \rho)) \mid \varphi, \psi, \rho \in F\},$$
$$SL_3 = \{(\neg \varphi \supset \neg \psi) \supset (\psi \supset \varphi) \mid \varphi, \psi \in F\};$$

— les axiomes caractérisant l'égalité formelle :

$$SL_4 = \{u \equiv u \mid u \in S\},$$
$$SL_5 = \{u_1 \equiv v_1 \supset \ldots \supset u_n \equiv v_n \supset f u_1 \ldots u_n \equiv f v_1 \ldots v_n \mid \text{pour } 1 \leqq i \leqq n,\ u_i \equiv v_i \in A \text{ et } f u_1 \ldots u_n \equiv f v_1 \ldots v_n \in A\},$$
$$SL_6 = \{u_1 \equiv u_2 \supset u_1 \equiv u_3 \supset u_2 \equiv u_3 \mid u_i \equiv u_j \in A \text{ si } 1 \leqq i < j \leqq 3\}.$$

août 1976.

3.4.2. *L'ensemble* R des règles d'inférence est formé seulement de la règle de détachement ou « Modus Ponens » :

$$\varphi, \varphi \supset \psi \vdash \psi.$$

REMARQUE : Il ressort des définitions précédentes que le système formel $\mathscr{F}$ d'une S.I. est caractérisé par un quadruplet (L, m, pl, X_p) ou par le triplet (L, Σ, X_p).

3.4.3. *Exemple de GOL :*

Définissons ici le sous ensemble X'_p (relatif aux objets internes de l'ensemble X_p des axiomes propres de GOL.

Ses éléments sont regroupés en schémas d'axiomes notés SH'.

a) Schémas d'axiomes relatifs aux noms :

$$\text{SH}'_{1.1} = \{ rep\ ch_x \text{u} \equiv ch_x\ rep \text{u} \mid x \in L_0^{\text{id}}; \text{u} \in \text{NOM} \},$$
$$\text{SH}'_{1.2} = \{ \textit{portée}\ ch_x \text{u} \equiv \textit{portée}\ \text{u} \mid \text{u} \in \text{NOM} \}.$$

Ces deux axiomes expriment [6, § 2.2.3.5 *b*]. En particulier $\text{SH}'_{1.1}$ signifie, en terme d'interprétation du système formel $\mathscr{F}$, que le diagramme suivant est commutatif (*fig.* 4).

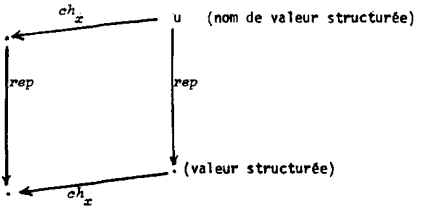

Figure 4.
Commutativité de ch_x et *rep*.

Ce type de propriété est indépendant de la donnée considérée : il ne dépend que du langage de programmation étudié. Ce sont de telles propriétés que l'on exprime par les axiomes propres.

b) Schémas d'axiomes caractérisant le symbole cond :

cond permet de formaliser les propositions conditionnelles. Il est défini par :

$$\text{SH}'_{2.1} = \{ \text{u} \equiv \text{v} \supset cond\ \text{uvw}_1\ \text{w}_2 \equiv \text{w}_1 \mid \text{u, v, w}_1, \text{w}_2 \in \text{VALEUR} \},$$
$$\text{SH}'_{2.2} = \{ \text{u} \not\equiv \text{v} \supset cond\ \text{uvw}_1\ \text{w}_2 \equiv \text{w}_2 \mid \text{u, v, w}_1, \text{w}_2 \in \text{VALEUR} \};$$

c) Schémas d'axiomes relatifs aux portées :

L'axiomatisation de « l'ensemble des portées » ne sera pas développée ici; elle l'est complètement en [5, § 4.1.5.6]. Ces axiomes permettent que

l'ensemble $\{1, 11, 12, \ldots, 111, \ldots, 1111, \ldots\}$ des suites d'entiers strictement positifs, ordonné par la relation : « être facteur gauche de » soit une interprétation de (PORTÉE, *ppq*).

Précisons simplement le lien entre la portée d'un champ d'une structure et la portée de la structure :

$$\mathrm{SH}'_{3\,1} = \{\, \textit{portée}\ \textit{ch}_x \mathrm{u} \equiv \textit{portée}\, \mathrm{u} \mid x \in L_0^{\mathrm{id}};\ \mathrm{u} \in \mathrm{STRU} \,\}.$$

Enfin en (4) et (6) nous compléterons la définition de X_p en précisant l'ensemble X''_p des axiomes caractérisant les accès associés aux phrases (ainsi $X_p = X'_p \cup X''_p$).

3.5. Informations de la structure

DÉFINITION 3 : Une *information* de la structure est un sous-ensemble de F contenant les théorèmes du système formel et stable par application de la règle du modus ponens. I est donc une information de la structure si et seulement si :

(i) $X \subset I$;

(ii) pour tous φ, ψ de F, $\varphi \in I$ et $\varphi \supset \psi \in I$ implique $\psi \in I$.

Une telle information formalise ici la notion d'état de mémoire.

Exemple 2 : Considérons le début de programme :

début ent x = 3; rep réel y = loc réel : = 4.1;
struct (rep réel xl, ent x2) z = (y, x);

L'information I obtenue après élaboration de ce morceau de programme est caractérisée, en plus des axiomes propres de la structure GOL et des axiomes logiques, par les axiomes (§ 3.6.4 et § 6.2) :

$$\textit{poss}\ x \equiv \textit{poss}\ 3 \qquad \textit{ch}_{x2}\ \textit{poss}\ z \equiv \textit{poss}\ x$$
$$\textit{rep}\ \textit{poss}\ y \equiv \textit{poss}\ 4.1 \qquad \textit{portée}\ \textit{poss}\ y \equiv \theta$$
$$\textit{ch}_{x1}\ \textit{poss}\ z \equiv \textit{poss}\ y$$

Alors $\textit{rep}\ \textit{ch}_{x_1}\ \textit{poss}\ z \equiv \textit{poss}\ 4.1$; $\textit{ch}_{x_2}\ \textit{poss}\ z \equiv \textit{poss}\ 3$ sont des exemples d'éléments de I.

L'ensemble T des théorèmes d'une S.I. et l'ensemble F des formules appartiennent à l'ensemble $\mathscr{I}$ des informations de la structure. Il est immédiat de voir que $\mathscr{I}$ est un treillis complet pour la relation d'inclusion, d'élément minimal T et d'élément maximal F.

Une étude des propriétés de ce treillis est esquissée en [5] et plus développée en [19]. En particulier on y introduit les notions de consistance et de complétude au sens de la logique mathématique [21]. Rappelons brièvement qu'une information I est dite consistante si elle diffère de F; qu'elle est complète

si elle est maximale dans l'ensemble des informations consistantes. La consistance (plus précisément l'inconsistance) est fondamentale dans la suite de notre formalisation (§ 3.6.3). Enfin les notions d'interprétation et de réalisation (généralisées pour tenir compte des profils) sont des outils puissants pour l'étude des i

3.6. Modifications élémentaires d'une S.I.

La notion d'état de mémoire étant formalisée par celle d'information d'une S.I., il s'agit de rendre compte du passage d'un état au « suivant ».

Par exemple, la déclaration *ent* $x = 3$ a pour rôle d'ajouter à l'ensemble des axiomes de l'information courante I l'axiome *poss* $x \equiv$ *poss* 3, donc d'« étendre » I en augmentant le nombre de ses théorèmes. On est ainsi conduit à définir la notion d'*extension* d'une S.I. par une autre. Cette formalisation est présentée en [5] et nous nous contenterons ici d'une idée intuitive.

3.6.1. *Modifications d'une S.I.*

DÉFINITION 4 : Si $\mathscr{I}$ est l'ensemble des informations d'une S.I., une modification de cette S.I. est une application de $\mathscr{I}$ dans $\mathscr{I}$.

Une S.I. est définie par la donnée d'un quadruplet (L, m, pl, X_p) déterminant son système formel, et par un ensemble de modifications dites *modifications élémentaires* de la S.I. On indique ainsi quelles modifications élémentaires sont acceptables dans le cadre de la S.I. considérée.

Ainsi toute S.I. est un couple ($\mathscr{F}$, $\mathscr{M}$od) formé d'un système formel et d'un ensemble de modifications élémentaires. Les seules modifications de cette S.I. que nous considérerons seront les modifications élémentaires, et les composées d'un certain nombre de modifications élémentaires; nous noterons $\widehat{\mathscr{M}\text{od}}$ leur ensemble.

3.6.2. *Définition d'une modification élémentaire*

Exemple 3 : Pour définir la modification ident (x, *poss* 3) formalisant la déclaration *ent* $x = 3$ on introduit la S.I. $\mathscr{S}_1$ obtenue à partir de la S.I. $\mathscr{S}$ en remplaçant *poss* par *poss'*, puis la S.I. $\mathscr{S}_2$ « contenant » $\mathscr{S}$ et $\mathscr{S}_1$. Plus précisément, l'ensemble de symboles fonctionnels L_2 de $\mathscr{S}_2$ est donnée par :

$$L_2 = L \cup \{ poss' \}$$

$\mathscr{S}_2$ traduit la cohabitation de $\mathscr{S}$ et $\mathscr{S}_1$. Le nouvel axiome précisé par ident (x, *poss* 3) est donc *poss'* $x \equiv$ *poss* 3.

Examinons comment cet axiome définit une modification de $\mathscr{S}$.

Soit I une information de $\mathscr{S}$, on considère successivement :

— l'information I_2, extension de I dans la structure $\mathscr{S}_2$, notée $\mathscr{E}_2(I)$; (I_2 est la plus petite information de $\mathscr{S}_2$, contenant I);

— l'information I_1, restriction de I_2 dans la structure $\mathscr{S}_1$ notée $\mathscr{R}_1(I_2)$ (par définition $I_1 = I_2 \cap F_1$ où F_1 est l'ensemble des formules de $\mathscr{S}_1$);

— l'information I'_1, obtenue à partir de I_1 en remplaçant *poss'* par *poss.* I'_1 est une information de $\mathscr{S}$, image de I par la modification définie par les axiomes écrits précédemment.

Revenons maintenant au cadre général.

DÉFINITION 5 : Soit $\mathscr{S}$ une S.I. dont le système formel est défini par le quadruplet (L, m, pl, X_p) et soit σ une bijection de l'ensemble d'accès L sur un ensemble L_1 dont la restriction à $L \cap L_1$ est l'identité. On définit une fonction profil pl_1 dans L_1 par :

$$\mathrm{pl}_1(\sigma(f)) = \mathrm{pl}(f) \qquad \text{pour} \quad f \in L$$

σ se prolonge naturellement en une application de l'ensemble des formules F sur un ensemble F_1 : F_1 est l'ensemble des formules de la structure $\mathscr{S}_1$ définie par L_1, m, pl_1 et $X_{p_1} = \sigma(X_p)$.

Soit $\mathscr{S}_2$ la structure définie par $L \cup L_1$, l'entier m, la fonction profil égale à pl sur L et à pl_1 sur L_1, ainsi qu'un ensemble d'axiomes contenant X_p et X_{p_1} :

$$X_{p_2} = X_p \cup X_{p_1} \cup Y.$$

La *modification élémentaire* π définie par σ et l'ensemble d'axiomes Y est :

$$\pi = \sigma^{-1} \circ \mathscr{R}_1 \circ \mathscr{E}_2,$$

où $\mathscr{R}_1$ et $\mathscr{E}_2$ ont la même signification que dans l'exemple précédent (*fig.* 5).

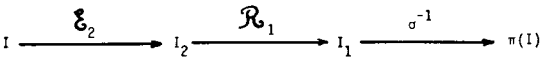

Figure 5.
Définition de π.

Convention : Une modification élémentaire π est définie par la donnée :

— d'une bijection σ_π ;

— d'un ensemble d'axiomes Y_π.

Dans la suite, nous conviendrons de ne définir σ_π que sur les symboles fonctionnels effectivement modifiés (éléments de L-L_1); elle se prolongera implicitement en l'identité sur les autres symboles.

août 1976.

3.6.3. *Schémas de modifications*

Dans l'exemple précédent, π dépend de x et 3; nous avons en fait défini un schéma de modifications à deux paramètres : ident.

Une structure d'information sera caractérisée par un système formel $\mathscr{F}$ et un nombre fini de schémas de modifications élémentaires. Les modifications élémentaires sont des applications et leur composition permet de définir celle des schémas de modifications.

3.6.4. *Exemple de GOL*

Définissons les différents schémas de modifications élémentaires de la S.I. de GOL.

a) ident :

En résumant ce qui précède, ident est un schémas à deux paramètres x et v qui ne modifie que le seul symbole *poss* :

$$\sigma_{\text{ident}(x,\,v)}(poss) = poss'$$

ident est défini par l'ensemble d'axiomes :

$$Y_{\text{ident}(x,\,v)} = \{\, poss'\, x \equiv \text{v} \,\} \cup \{\, poss'\, y \equiv poss\, y \mid y \in \text{EXT} - \{x\} \,\}$$

REMARQUE : La définition formelle d'une déclaration d'identité est beaucoup plus complexe dans le cas d'Algol 68 [5, § 6.2.3] à cause des valeurs multiples et des déclarations de modes. Il faut en effet rendre compte des tests de cohérence portant sur les modes alors qu'ici ces tests sont purement syntaxiques.

b) affect :

C'est un schéma de modifications élémentaires à deux paramètres u, v qui permet de définir l'affectation en GOL (§ 6.2.4). Il permet de formaliser :

— que le nom u auquel on affecte v doit être différent de *nil* (plus précisément de *poss nil* qui est le nom possédé par la notation *nil* $\in$ PH);

— que la portée de u doit être inférieure à celle de v;

— que le seul symbole modifié est *rep*.

Ainsi

$$\sigma_{\text{affect}(u,\,v)}(rep) = rep'$$

et

$$\begin{aligned} Y_{\text{affect}(u,\,v)} = {} & \{\, \text{u} \not\equiv poss\ nil \,\} \cup \\ & \{\, ppq\ portée\ \text{u}\ portée\ \text{v} \equiv poss\ vrai \,\} \cup \\ & \{\, rep'\, \text{u} \equiv \text{v} \,\} \cup \\ & \{\, rep'\, \text{w} \equiv rep\, \text{w} \mid \text{w} \in \text{NOM} - \{\text{u}\} \,\}. \end{aligned}$$

Alors si u est formellement égal à *poss nil* (ou si la portée de u est formellement supérieure à celle de v) dans une information I, l'image de I par affect (u, v) est l'information inconsistante (c'est l'ensemble F des formules).

On exprime ainsi le « non défini » au sens d'Algol 68. Remarquons également que la formalisation de l'affectation est plus complexe dans le cas d'Algol 68 complet [5, § 6.3.2].

c) *Les schémas de modifications élémentaires* j *et* $\bar{j}$.

j (resp. $\bar{j}$) permet de « réduire » le domaine de définition d'une modification qui est l'interprétation d'un calcul. Il est utilisé en particulier dans la définition du schéma de calculs associé à une proposition conditionnelle (§ 6.2.15). j (resp. $\bar{j}$) est un schéma à 2 paramètres.

DÉFINITION DE j : j est construit de telle sorte que j (u,v) soit l'identité sur l'ensemble des informations I contenant le théorème $u \equiv v$ $(u, v \in S)$ et ne soit « pas défini » sur l'ensemble des informations contenant le théorème $u \not\equiv v$:

— $\sigma_{j(u,v)}$ est l'identité sur L;
— $Y_{j(u,v)} = \{ u \equiv v \}$.

Si I est une information complète deux cas peuvent se produire :

— $u \equiv v \in I$ et alors $j(u, v)\ (I) = I$;
— $u \not\equiv v \in I$ et alors $j(u, v)\ (I) = F$ (*information inconsistante*).

Si I est consistante sans être complète, $u \equiv v$ et $u \not\equiv v$ peuvent ne pas appartenir à I, alors $j(u, v)\ (I)$ est l'extension de I obtenue par adjonction de l'axiome $u \equiv v$.

Intuitivement tout calcul commençant par j (u, v) s'interprète comme une modification « définie » seulement sur les informations I telles que $u \not\equiv v \notin I$.

DÉFINITION DE $\bar{j}$: Avec des remarques analogues aux précédentes :

a) $\sigma_{\bar{j}(u,v)}$ est l'identité sur L;
b) $Y_{\bar{j}(u,v)} = \{ u \not\equiv v \}$.

Ainsi la notion d'inconsistance est fréquemment utilisée, soit pour exprimer le « non défini », soit pour « éliminer » certains calculs associés à un programme (lorsqu'à la suite d'un test un tel calcul correspond à la branche de l'organigramme d'un programme que l'on ne suit pas).

d) selec :

C'est un schéma de modification à un paramètre qui permet d'exprimer que la valeur possédée par certaines expressions ne peut être *nil*. Ce schéma

ne modifie aucun symbole ($\sigma_{\text{select}(E)}$ est l'identité sur L); il ajoute simplement l'axiome

$$Y_{\text{select}(E)} = \{ poss\, E \not\equiv poss\ nil \}.$$

4. LANGAGE PIVOT ET AXIOMES D'ACCÈS

4.1. Introduction

Rappelons que la notion de language pivot permet essentiellement de représenter tout programme sous une forme qui met en évidence la nature syntaxique des différentes phrases qui le composent et permet d'introduire localement certains renseignements globaux (tels que portée d'un nom en Algol 68, domaine de validité d'une déclaration d'identité...).

En bref, le langage pivot possède des propriétés telles que la forme d'un quelconque de ses programmes soit adaptée à une définition aisée de la sémantique. Par exemple en [5], comme dans la méthode de Vienne [13], la forme d'un programme pivot est une ramification (Pair et Quere [18]), ce qui est naturel dès qu'on s'aperçoit que la signification d'une phrase ne dépend que de ses composantes.

D'autre part nous avons déjà remarqué qu'il est nécessaire de rendre compte de la valeur d'une expression. Par exemple pour définir la modification associée à l'affectation GOL :

$$x := si\ b\ alors\ E_1\ sinon\ E_2\ fsi,$$

il sera nécessaire d'introduire un objet de la S.I. représentant la valeur de la proposition conditionnelle. Plus précisément, comme cette modification sera définie sur la phrase GOL_0 associée à l'affectation, c'est la valeur de cette phrase GOL_0 qui devra se trouver dans la structure d'information.

Nous sommes ainsi conduits à faire posséder une valeur, c'est-à-dire un accès 0-aire de la S.I., à certaines phrases du langage pivot. Pour cela nous introduirons systématiquement toutes les phrases pivot dans la S.I. et alors, pour unifier complètement la description, ces phrases seront des schémas fonctionnels (qui sont des objets peu différents des ramifications).

Dans la suite de cette section, nous précisons ces idées en définissant le langage pivot GOL_0, puis nous introduisons la notion d'axiomes d'accès qui permet d'associer à certaines phrases leurs valeurs.

4.2. Exemple de GOL : le langage pivot GOL_0

4.2.1. *Symboles fonctionnels de GOL_0*

Soit $L'' = \bigcup_{q \geq 1} L''_q$ l'ensemble des symboles fonctionnels à partir desquels sont construits les schémas fonctionnels qui sont les phrases de GOL_0.

a) $L''_0 = L_0^{ent} \cup L_0^{bool} \cup L_0^{id} \cup \{ nil \} \cup \{ \lambda \} \cup L_0^{decl}$ où :

— L_0^{ent} est l'ensemble des symboles 0-aires qui sont interprétables comme (nous dirons brièvement « qui sont ») les entiers;

— $L_0^{bool} = \{ vrai; faux \}$;

— L_0^{id} est l'ensemble des symboles qui sont les identificateurs. Il contient en particulier des symboles $\sim_i$ pour $i \geqq 1$ qui permettent de rendre compte de notations de procédures (§ 6.2.9);

— *nil* est défini en [6, § 2.2.3.5 *a*];

— λ permet de rendre compte de la notion de proposition de genre neutre ainsi que des procédures sans paramètre ou sans résultat. Sa traduction concrète sera le mot vide de Alg* où Alg est l'alphabet de GOL;

— L_0^{decl} est l'ensemble des déclarateurs de modes utilisés en GOL_0. (*voir* annexe 1).

b) Les autres éléments de L'' sont précisés dans l'annexe 2. Avec les définitions de (§ 3.3.3) on peut alors présenter complètement l'ensemble L des symboles fonctionnels de la S.I. de GOL.

$$L = \bigcup_{q \geqq 0} L_q \qquad \text{avec} \quad L_q = L'_q \cup L''_q \quad \text{pour} \quad q = 0, 1, 2, 4$$

et

$$L_n = L''_n \qquad \text{pour} \quad n = 3 \quad \text{ou} \quad n \geqq 5.$$

4.2.2. *Schémas fonctionnels :*

Ils sont définis par le système d'équations Σ'' défini ci-dessous (les diverses équations sont commentées dans la suite)

(10) PROG = *prog* PAREN
PAREN = FERMÉE ∪ COLL ∪ COND

(11) FERMÉE = *parent* SER

(12) COLL = $\bigcup_{n \geqq 2}$ *coll listcoll*$_n^{12}$ UNIT$^{(n)}$ GENRE
GENRE = $\{ \lambda \} \cup L_0^{decl}$

(13) COND = *conditionnelle* SER SER SER

(14) SER = *série* PORTÉE [($\bigcup_{n \geqq 2}$ *listser*$_n^{14}$ TETE$^{(n-1)}$ UNIT) ∪ UNIT]
TETE = UNIT ∪ DECL
DECL = DECOLL ∪ DECID

(15) DECOLL = $\bigcup_{n \geqq 2}$ *listcoll*$_n^{15}$ DECID$^{(n)}$

(16) DECID = *decid* L_0^{decl} L_0^{id} UNIT

(17) UNIT = TERT ∪ CONF ∪ *derep* UNIT
TERT = SEC ∪ FORM
SEC = PRIM ∪ COH
PRIM = BASE ∪ PAREN
CONF = AFF ∪ RELID

août 1976.

(18) FORM = *sigmoins* SEC SEC ∪ *inférieur* SEC SEC ∪ *égal* SEC SEC
COH = SEL ∪ GEN
BASE = $L_0^{ent} \cup L_0^{nonl} \cup L_0^{id} \cup$ NOTPROC ∪ APPEL

(19) AFF = *affectation* TERT UNIT

(20) RELID = *relid* TERT TERT

(21) SEL = *de* L_0^{id} SEC

(22) GEN = *genloc* L_0^{procl} PORTÉE ∪ *genglob* L_0^{decl}

(23) NOTPROC = *notproc* PARAM GENRE UNIT PORTÉE[2]
PARAM = { λ } ∪ LISTCOLL ∪ LISTSER ∪ PARFOR

(24) LISTCOLL = $\bigcup_{n \geqq 2}$ *listcoll* $_n^{24}$ PARFOR$^{(n)}$

(25) LISTSER = $\bigcup_{n \geqq 2}$ *listser* $_n^{25}$ PARFOR$^{(n)}$

(26) PARFOR = *parfor* L_0^{decl} L_0^{id}

(27) APPEL = *appel* PRIM PAREF
PAREF = { λ } ∪ SUITCOLL ∪ SUITSER ∪ UNIT

(28) SUITCOLL = $\bigcup_{n \geqq 2}$ *listcoll* $_n^{28}$ UNIT$^{(n)}$

(29) SUITSER = $\bigcup_{n \geqq 2}$ *listser* $_n^{29}$ UNIT$^{(n)}$

PH = SER ∪ UNIT ∪ DECL

Ainsi *prog* (*parent* (*serie* (θ, *lister*$_2^{14}$ (*decid* (*rep ent*, *x*, *genglob ent*), *affectation* (*x*, 3))))) est un exemple de programme GOL_0, dans lequel on a introduit des parenthèses et des virgules pour améliorer la compréhension.

Commentaires : Seules les équations numérotées sont des éléments de Σ, les autres n'ont pour rôle que de permettre des abréviations d'écriture.

La numérotation commence à partir de 10 : les 9 premières équations sont les éléments de Σ'.

— (10) définit les programmes;

— (11) définit les propositions parenthésées à partir des propositions sérielles;

— (12) définit les propositions collatérales à l'aide des propositions unitaires et des symboles *coll* et *listcoll*$_n$. En particulier le genre d'une telle proposition peut être neutre, c'est ce qui explique la présence de λ. Rappelons que UNIT$^{(n)}$ représente l'ensemble des mots de longueur *n* sur UNIT;

— (13) caractérise les propositions conditionnelles;

— (14) se rapporte aux propositions sérielles. Remarquons l'introduction explicite de la portée que définit cette région [6, § 4.1.1 *e*)];

— (15) et (16) définissent respectivement les déclarations collatérales et d'identité;

— (17) permet de définir les propositions unitaires. Une telle proposition peut être, en particulier, un élément de *derep* UNIT, ce qui rend compte de la modification syntaxique d'Algol 68 « dereperer ».

Les équations suivantes définissent les différentes propositions unitaires en donnant une description analogue à celle de [6].

Remarquons simplement que dans l'équation (22), les schémas formalisant les générateurs locaux ont une composante qui est une portée.

D'autre part, dans l'équation (23) définissant les notations de procédures, PARAM représente la liste des paramètres formels (qui peut être vide, collatérale ou sérielle), GENRE définit le mode du résultat, le premier symbole de PORTÉE représente la portée de la région qui est cette notation, le deuxième représente la portée de cette notation dans le programme complet. L'introduction de λ dans PARAM permet d'exprimer la modification syntaxique d'Algol 68 « procedurer » :

$$notproc(\lambda, ent, moins(x, y), q, \theta)$$

est une notation de procédure sans paramètre, à résultat entier de portée q, qui possède une routine de portée θ.

De manière analogue, les paramètres effectifs d'un appel (27) peuvent être absents, ce qui rend compte de la modification syntaxique « deprocedurer » :

$$appel(moins(x, y), \lambda)$$

représente l'appel d'une procédure sans paramètre.

4.2.3. *Problèmes de traduction*

Le langage GOL peut être défini syntaxiquement par une sous-grammaire de la double grammaire d'Algol 68.

Sa traduction concrète est une application $\mathscr{T}$ de l'ensemble PH des phrases de GOL_0 dans l'ensemble $\mathfrak{P}(Alg^*)$ où Alg est l'alphabet de GOL.

Exemples :

$$\mathscr{T}\ (coll\ (listcoll_4\ (x, y, z, t), struct\ (ent\ x_1, ent\ x_2, bool\ y_1, bool\ y_2))) = \{(x, y, z, t)\}.$$

$$\mathscr{T}\ (notproc\ (listcoll_2\ (parfor\ (ent, x), parfor\ (bool, b)), ent, conditionnelle\ (b, x, sigmoins0x), q, \theta)) = \{(ent\ x, bool\ b)\ ent : si\ b\ alors\ x\ sinon\ 0-x\ fsi\}.$$

La définition de $\mathscr{T}$ est donnée en (§ 6) où l'on indique, pour chaque type de phrase de GOL_0, quelles sont ses images dans $\mathfrak{P}(Alg^*)$. Nous verrons qu'en fait, pour toute phrase P de PH, $\mathscr{T}(P)$ sera soit l'ensemble vide, soit un ensemble réduit à un élément.

août 1976.

Exemple :

$\mathcal{T}$ (*prog parent serie* (θ, *listser*$_2$ (*decid* (*rep ent*, *x*, *genglob ent*), *parent serie* (θ, *affectation* (*x*, 3))))) = $\varnothing$,

car la portée de la proposition sérielle la plus interne est θ; il faudrait qu'elle soit *desc* θ pour que la traduction concrète soit :

$$\{\textit{début rep ent } x = \textit{tas ent, début } x := 3 \textit{ fin fin}\}.$$

De plus les images de certaines phrases de GOL_0 peuvent ne pas appartenir à GOL. Par exemple l'élément de :

$$\mathcal{T} \text{ (}\textit{conditionnelle (serie } (\theta, \textit{ sigmoins } (x, y)), \text{ u, v)) =}$$
$$\{\textit{si } x - y \textit{ alors } \text{u } \textit{sinon } \text{v } \textit{fsi}\}$$

n'appartient pas à GOL si on impose (par exemple par une double grammaire) que le mode de $x - y$ soit booleen.

Ce qui résulte du fait que nous n'exprimons pas dans GOL_0 un certain nombre de contraintes syntaxiques qui existent dans GOL. Nous définirons cependant la sémantique d'une phrase quelconque de GOL_0 en sachant que les seules phrases qui auront un intérêt seront celles qui ont une image dans GOL. Nous n'étudierons pas ici le problème de la caractérisation de la relation réciproque de $\mathcal{T}$, définie sur GOL, c'est-à-dire de la traduction abstraite. Une telle définition serait une composante d'une définition complète d'un compilateur de GOL.

REMARQUE : Il serait possible d'introduire les contraintes syntaxiques des programmes concrets dans les programmes pivots, en compliquant le mode de définition de ces derniers. $\mathcal{T}$ serait alors une application de PH dans l'ensemble des parties de GOL. Ce serait une méthode permettant de définir, à partir du langage pivot, conjointement syntaxe concrète et sémantique.

4.3. Axiomes d'accès

4.3.1. *Définition*

Exemple : Pour obtenir la valeur de la proposition conditionnelle de GOL_0 :

$$\textit{conditionnelle } (B, E_1, E_2) \quad \text{où} \quad B, E_1, E_2 \in SER$$

nous sommes conduits à lui associer l'axiome :

$$\textit{poss conditionnelle } (B, E_1, E_2) \equiv \textit{cond } (\textit{poss } B, \textit{poss vrai},$$
$$\textit{poss } E_1, \textit{poss } E_2).$$

Plus généralement, à chaque type de phrase du langage pivot est adjoint un ensemble de schémas d'axiomes qui définit l'accès associé à la phrase ou précise ses propriétés. De tels axiomes sont appelés *axiomes d'accès*, leur ensemble X''_p est inclu dans l'ensemble X_p des axiomes propres de la S.I. du langage.

4.3.2. *Axiomes d'accès de GOL*

Dans le cas de GOL, X''_p contient en particulier :

$$\mathrm{SH}''_{0.1} = \{ x \not\equiv y \mid x, y \in L_0, x \neq y \}$$

qui, associé au schéma logique SL_4, précise que les égalités formelles et ensemblistes sont confondues pour les éléments de L_0.

Les autres axiomes d'accès de GOL seront introduits à la section 6 où nous indiquons conjointement, pour chaque type de phrase de GOL_0 :

— traduction concrète;

— schémas d'axiomes d'accès;

— système de calculs [*voir* (§ 5)].

4.4. Structure d'information de GOL

Résumons la définition de la S.I. de GOL présentée dans cette section et dans la précédente :

• l'ensemble L des symboles fonctionnels est la réunion de L' (relatif aux objets internes) et de L'' (relatif au langage pivot);

• l'ensemble S des schémas fonctionnels est défini par le système à point fixe $\Sigma = \Sigma' \cup \Sigma''$, Σ' caractérisant les objets internes et Σ'' les phrases de GOL_0 [ou, ce qui revient au même, par le couple (m, pl)];

• l'ensemble X_p des axiomes propres est formé de X'_p (relatif aux objets internes) et X''_p (relatif aux accès associés aux phrases).

ANNEXE 1

DÉFINITION SYNTAXIQUE DE GOL

La grammaire définissant la syntaxe de GOL est décrite à l'aide de la notation de Backus en convenant que :

{ notion }$^{0/1}$ (resp. { notion }$^{+}$; { notion }*) signifie que notion est écrit zéro ou une fois (resp. un nombre quelconque non nul de fois; un nombre quelconque, éventuellement nul, de fois)

⟨ programme ⟩ : : = ⟨ prop. parenthésée ⟩
⟨ prop. parenthésée ⟩ : : = ⟨ prop. fermée ⟩ | ⟨ prop. collatérale ⟩ | ⟨ prop. conditionnelle ⟩
⟨ prop. fermée ⟩ : : = *début* ⟨ prop. sérielle ⟩ *fin*
⟨ prop. collatérale ⟩ : : = (⟨ liste de prop. unitaires ⟩)
⟨ liste de prop. unitaires ⟩ : : = ⟨ prop. unitaire ⟩ { , ⟨ prop. unitaire ⟩ }$^{+}$
⟨ prop. conditionnelle ⟩ : : = *si* ⟨ prop. sérielle ⟩ *alors* ⟨ prop. sérielle ⟩ *sinon* ⟨ prop. sérielle ⟩ *fsi*
⟨ prop. sérielle ⟩ : : = { ⟨ simple ⟩ {; ⟨ simple ⟩ }*; }$^{0/1}$ ⟨ prop. unitaire ⟩
⟨ simple ⟩ : : = ⟨ prop. unitaire ⟩ , ⟨ déclaration ⟩
⟨ déclaration ⟩ : : = ⟨ décl. collatérale ⟩ | ⟨ décl. d'identité ⟩
⟨ décl. collatérale ⟩ : : = ⟨ décl. d'identité ⟩ { , ⟨ décl. d'identité ⟩ }$^{+}$
⟨ décl. d'identité ⟩ : : = ⟨ déclareur ⟩ ⟨ identificateur ⟩ = ⟨ prop. unitaire ⟩
⟨ prop. unitaire ⟩ : : = ⟨ tertiaire ⟩ | ⟨ confrontation ⟩
⟨ tertiaire ⟩ : : = ⟨ secondaire ⟩ | ⟨ formule ⟩
⟨ secondaire ⟩ : : = ⟨ primaire ⟩ | ⟨ cohésion ⟩
⟨ primaire ⟩ : : = ⟨ base ⟩ | ⟨ prop. parenthésée ⟩
⟨ confrontation ⟩ : : = ⟨ affectation ⟩ | ⟨ relation d'identité ⟩
⟨ formule ⟩ : : = ⟨ secondaire ⟩ ⟨ opérateur ⟩ ⟨ secondaire ⟩
⟨ cohésion ⟩ : : = ⟨ sélection ⟩ | ⟨ générateur ⟩
⟨ base ⟩ : : = ⟨ identificateur ⟩ | ⟨ notation ⟩ | ⟨ appel ⟩ | *nil*
⟨ affectation ⟩ : : = ⟨ tertiaire ⟩ : = ⟨ prop. unitaire ⟩
⟨ relation d'identité ⟩ : : = ⟨ tertiaire ⟩ : = : ⟨ tertiaire ⟩
⟨ opérateur ⟩ : : = − | < | =
⟨ sélection ⟩ : : = ⟨ identificateur ⟩ *de* ⟨ secondaire ⟩
⟨ générateur ⟩ : : = *loc* ⟨ déclareur ⟩ | *tas* ⟨ déclareur ⟩
⟨ notation ⟩ : : = ⟨ notation d'entier ⟩ | ⟨ notation de booléen ⟩ | ⟨ notation de procédure ⟩
⟨ notation d'entier ⟩ : : = { ⟨ chiffre ⟩ }$^{+}$
⟨ notation de booléen ⟩ : : = *vrai* | *faux*
⟨ notation de procédure ⟩ : : = (⟨ paramètres formels ⟩) { ⟨ forceur ⟩ | ⟨ forceur neutre ⟩ } | ⟨ forceur ⟩ | ⟨ prop. unitaire ⟩
⟨ forceur ⟩ : : = ⟨ déclareur ⟩ : ⟨ prop. unitaire ⟩
⟨ forceur neutre ⟩ : : = : ⟨ prop. unitaire ⟩
⟨ paramètres formels ⟩ : : = ⟨ parm. formel ⟩ { ⟨ poingule ⟩ ⟨ parm. formel ⟩ }*
⟨ parm. formel ⟩ : : = ⟨ déclareur ⟩ ⟨ identificateur ⟩
⟨ poingule ⟩ : : = , | ;
⟨ appel ⟩ : : = ⟨ primaire ⟩ (⟨ parm. effectif ⟩ { ⟨ poingule ⟩ ⟨ parm. effectif ⟩ }*) | ⟨ primaire ⟩
⟨ parm. effectif ⟩ : : = ⟨ prop. unitaire ⟩
⟨ déclareur ⟩ : : = *ent* | *bool* | ⟨ décl. struct ⟩ | ⟨ décl. proc ⟩ | *rep* ⟨ déclareur ⟩
⟨ décl. struct ⟩ : : = *struct* (⟨ champ ⟩ {, ⟨ champ ⟩ }*)
⟨ champ ⟩ : : = ⟨ déclareur ⟩ ⟨ identificateur ⟩
⟨ décl. proc ⟩ : : = *proc* { ⟨ décl. de paramètres ⟩ }$^{0/1}$ { ⟨ déclareur ⟩ }$^{0/1}$
⟨ décl. de paramètres ⟩ : : = ⟨ déclareur ⟩ { ⟨ poingule ⟩ ⟨ déclareur ⟩ }*

ANNEXE 2

Symboles fonctionnels permettant de définir les phases de GOL

Symbole	Arité	Notion syntaxique exprimée	Remarques
derep	1	modification syntaxique « dereperer »	
genglob	1	générateurs globaux	
parent	1	Propositions parenthésées	
prog	1	programmes	
série	2	portée associée à une proposition sérielle	
coll	2	genre associé à une proposition collatéralle	
affectation	2	affectation	
sigmoins	2	opérateur « — »	
inférieur	2	relation « < »	
égal	2	relation « = »	
relid	2	relation d'identité « : = : »	
de	2	sélection	
genloc	2	générateurs locaux	
parfor	2	paramètres formels d'une notation de procédure	
decid	3	déclaration d'identité	
conditionnelle	3	propositions conditionnelles	
notproc	5	notations de procédures	
listcoll $^{i}_{n}$	$n \geq 2$	liste collatérale à n éléments	$i \in \{ 12, 15, 24, 28 \}$
lister $^{i}_{n}$	$n \geq 2$	liste sérielle à n éléments	$i \in \{ 14, 25, 29 \}$

août 1976.

BIBLIOGRAPHIE

1. J. W. DE BAKKER, *Fixed Point in Programming Theory*, Lecture Notes for the Advanced Course on the Foundations of Computer Science, Amsterdam, 1974.
2. J. W. DE BAKKER, *Formal Definition of Programming Languages*, Mathematical Center Tracts, Mathematisch Centrum Amsterdam, vol. 16, 1967.
3. A. BLIKLE, *An Algebraic Approach to Semantics of Programs*, Advanced Course on Semantics of Programming Languages, Saarbrücken, 1974.
4. N. CHOMSKY, *Aspects of the Theory of Syntax*, The M.I.T. Press, Cambridge, Mass., 1965.
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.
7. C. A. R. HOARE, *An Axiomatic Basis for Computer Programming*, Comm. A. C. M., 12, 1969, p. 576-583.
8. C. A. R. HOARE, N. WIRTH, *An Axiomatic Definition of the Programming Language Pascal*, Acta Informatica, 2, 1973, p. 335-355.
9. C. A. R. HOARE, P. E. LAUER, *Consistent and Complementary Formal Theories of the Semantics of Programming Languages*, Acta Informatica, 3, 1974, p. 135-153.
10. D. E. KNUTH, *Semantics of Context Free Languages*, Math. Systems Theory, 2, 1968, p. 127-145.
11. G. KREISEL, J. L. KRIVINE, *Éléments de Logique Mathématique*, Théorie des Modèles, Dunod, Paris, 1966.
12. P. J. LANDIN, *A Correspondence Between Algol 60 and Church's Lambda Notation*, Comm. A. C. M., 8, 1965, p. 89-101.
13. P. LUCAS, P. LAUER, H. STIGLEITNER, *Method and Notation for the Formal Definition of Programming Languages*, IBM Laboratory Vienna, Technical Report TR 25.087, 1968.
14. J. MCCARTHY, *Towards a Mathematical Science of Computation*, Information Processing (POPPLEWELL ed.), Proceedings of I.F.I.P. Congress 1962, North-Holland, Amsterdam, 1963, p. 21-28.
15. Z. MANNA, *The Correctness of Programs*, J. Comp. Syst. Sci., 3, 1969, p. 119-127.
16. C. PAIR, *Formalization of the Notions of Data, Information and Information Structure, in* Data Base Management Systems, KLIMBIE-KOFFEMAN (ed.), North-Holland, 1974, p. 149-167.
17. C. PAIR, *Calculs, ensembles de calculs, équivalence de programmes, in* Symposia Mathematica, Rome, Academic Press, XV, 1975, p. 35-53.
18. C. PAIR, A. QUERE, *Définition et Études des Bilangages Réguliers*, Information and Control, 13, 1968, p. 565-593.
19. J. L. REMY, *Structure d'Information, Formalisation des Notions d'Accès et de Modification d'une Donnée*, Thèse de 3e Cycle, Université de Nancy 1, 1974.
20. D. SCOTT, *Continuous Lattices*, Oxford Mono PRG-7, Oxford University, 1972.
21. J. R. SHOENFIELD, *Mathematical Logic*, Addison-Wesley, 1967.
22. C. STRACHEY, D. SCOTT, Oxford Mono PRG-6, Oxford University, 1971.
23. A. VAN WINJGAARDEN (ed.), B. J. MAILLOUX, J. E. L. PECK, C. H. A. KOSTER, *Report on the Algorithmic Language Algol 68*, Mathematisch Centrum, Amsterdam MR 101, 1969.