UNIVERSITÉ DE NANCY FACULTÉ DES SCIENCES

# DEFINITION DE LA SYNTAXE DES LANGAGES DE PROGRAMMATION

-=-=-=-=-

# THÈSE



pour l'obtention du

DOCTORAT de SPECIALITE MATHEMATIQUES (3ème CYCLE)

Soutenue devant le Jury le 8 mai 1970

par

Jean-Marie LECLAIRE

-=-=-=-=-

Jury : Mr LEGRAS Président
Mr PAIR Examinateurs
Mr BOUSSARD

UNIVERSITE DE NANCY FACULTE DES SCIENCES



# DEFINITION DE LA SYNTAXE DES LANGAGES DE PROGRAMMATION

---

par Jean-Marie LECLAIRE

Université de Nancy — Faculté des Sciences

ANNEE SCOLAIRE 1969/1970

DOYEN : M. AUBRY

ASSESSEUR : M. GAY

Doyens honoraires : MM. CORNUBERT - ROUBAULT.

Professeurs honoraires : MM. RAYBAUD - LAFFITTE - LERAY - JOLY - LAPORTE - EICHBORN - CAPELLE - GODEMENT - L. SCHWARTZ - DIEUDONNE - DE MALLEMAN - LONGCHAMBON - LETORT - DODE - GAUTHIER - GOUDET - OLMER - CORNUBERT - CHAPELLE - GUERIN - WAHL - DUBREIL.

Maîtres de Conférences honoraires : MM. LIENHART - PIERRET - Mlle MATHIEU.

## PROFESSEURS

| | | | | |
|---|---|---|---|---|
| MM. | ROUBAULT | Géologie | GAYET | Physiologie |
| | VEILLET | Biologie animale | HADNI | Physique |
| | BARRIOL | Chimie théorique | ✱BASTICK | Chimie |
| | BIZETTE | Physique | DUCHAUFOUR | Pédologie |
| | GUILLIEN | Electronique | GARNIER | Agronomie |
| | LEGRAS | Mécanique rationnelle | NEEL | Chimie organique industrielle |
| | BOLFA | Minéralogie | BERNARD | Géologie appliquée |
| | NICLAUSE | Chimie | ✱CHAMPIER | Physique |
| | FAIVRE | Physique appliquée | ✱GAY | Chimie biologique |
| | AUBRY | Chimie minérale | STEPHAN | Zoologie |
| | COPPENS | Radiogéologie | ✱CONDE | Zoologie |
| | DUVAL | Chimie | ✱WERNER | Botanique |
| | FRUHLING | Physique | EYMARD | Calcul différentiel et intégral |
| | HILLY | Géologie | | |
| | LE GOFF | Génie chimique | LEVISALLES | Chimie organique |
| | SUHNER | Physique expérimentale | FELDEN | Physique |
| | CHAPON | Chimie biologique | ✱GOSSE | Mécanique physique |
| | HEROLD | Chimie minérale industrielle | ✱DAVOINE | Physique (ENSMIN) |
| | SCHWARTZ B. | Exploitation minière | HORN | Physique (1° cycle) |
| | MALAPRADE | Chimie | ✱ROCCI | Géologie |
| | MANGENOT | Botanique | ✱Mme LUMER | Mathématiques |
| | | | DELPUECH | Chimie physique |

| | |
|---|---|
| N... | Chimie biologique |
| N... | Mécanique appliquée |
| N... | Analyse supérieure |
| N... | Méthodes mathématiques de la physique |
| N... | Mécanique rationnelle |

(✱) Professeur titulaire à titre personnel

## PROFESSEURS SANS CHAIRE

| | | | |
|---|---|---|---|
| BASTICK | Chimie P. C. Epinal | BALESDENT | Thermodynamique, Chimie appliquée (ENSIC) |
| GUDEFIN | Physique | | |
| VILLAUME | Psychophysiologie | BLAZY | Minéralogie Appliquée (ENSG) |
| FRENTZ | Biologie Animale | | |
| MARI | Chimie (ISIN) | JANOT | Physique PC Epinal |
| AUROUZE | Géologie | CACHAN | Entomologie Appliquée (ENSA) |
| DEVIOT | Physique du solide | | |
| FLECHON | Physique PC. | JACQUIN | Pédologie et Chimie agricol |
| e HUET | Mathématiques CBG | MAINARD | Physique M. P. |
| VIGNES | Metallurgie | MARTIN | Chimie P. C. |
| PAIR | Mathématiques appliquées | DEPAIX | Probabilitéset Statistiques |
| PAULMIER | Mécanique Expérimentale | PROTAS | Minéralogie |
| RIVAIL | Chimie Appliquée (Cuces) | | |

## MAITRES DE CONFERENCES

| | | | |
|---|---|---|---|
| JOZEFOWICZ | Physico-Chimie | GERL | Physique (ENSEM) |
| VILLERMAUX | Génie Chimique | ROQUES | Chimie Minérale |
| METCHE | Biochimie appliquée (Brasserie) | FERRIER | Mathématiques |
| | | GILORMINI | Mécanique (ISIN) |
| BAUMANN | Physique 1er Cycle | N... | Mécanique des fluides (ISIN) |
| DURAND | Physique | N... | Mathématiques |
| GRANGE | Physique (ISIN) | N... | Mathématiques P. C. |
| BAVEREZ | Chimie (ENSIC) | N... | Mathématiques C. B. G. |
| CHAMBON | Exploitation Minière (Mines) | N... | Physiologie Animale |
| | | N... | Mathématiques M. P. |
| HUSSON | Physique (ENSEM) | N... | Chimie Organique |
| WEISSLINGER | Physique | N... | Chimie (ENSIC) |
| | | N... | Mathématiques |

## CHARGES D'ENSEIGNEMENTS

MM. AMARIGLIO - CŒURE - DAVRAINVILLE - GIRARDEAU - HILY - SCHREIBER
NOVERRAZ - OVAERT - RUYER - WEBER.

Que Monsieur le Professeur LEGRAS, Directeur de l'Institut Universitaire de Calcul Automatique, trouve ici l'expression de ma profonde gratitude pour l'enseignement qui m'a été dispensé à l'Institut, et, l'honneur qu'il me fait en présidant le Jury.

Ce travail a été effectué sous la direction de Monsieur le Professeur PAIR à qui j'exprime mes plus vifs remerciements pour la formation qu'il m'a donnée et pour les conseils qu'il n'a cessé de m'apporter tout au long de mon travail.

Je remercie Monsieur BOUSSARD, Professeur à la Faculté des Sciences de Grenoble pour l'honneur qu'il me fait en acceptant de participer au Jury.

Je tiens également à remercier toute l'équipe de l'Institut de Calcul Automatique et particulièrement Mademoiselle D. MARCHAND qui a réalisé matériellement ce travail.

# SOMMAIRE

# INTRODUCTION

La plupart des langages de programmation sont définis par une grammaire de Chomsky [GROSS et LENTIN] engendrant un langage $L_0$, ainsi que des règles complémentaires, généralement rédigées en anglais, permettant de déterminer un sous-ensemble L de $L_0$, qui est l'ensemble des programmes corrects. Ces règles traitent en particulier des déclarations des variables et de l'homogéneité des expressions. Ce sont des règles de contexte ; les unes sont globales ou relatives à une structure de bloc, les autres sont locales elles traitent des expressions.

Pour Algol 68 une autre méthode a été envisagée [A. VAN WIJNGAARDEN]. L'auteur donne une grammaire permettant d'engendrer des programmes où les phrases respectent la règle d'homogéneité. Cependant tous les programmes engendrés par cette grammaire ne sont pas des programmes "propres" "d" Algol 68 car ils doivent pour cela répondre à des conditions de contextes globales, définies par des règles rédigées en anglais.

Nous proposons, ici, la méthode suivante pour définir un langage L :

- une grammaire de Chomsky $G_0$ engendre un langage $L_0$ et un bilangage [PAIR - QUERE] $B_0$ ;
- une prédicat récursif [QUERE] reconnait un sous ensemble B de $B_0$ ; L est le sous ensemble de $L_0$ formé des mots, des feuilles, des ramifications de B.

Il s'agit là de la formalisation d'une méthode habituellement utilisée. Définir un langage "trop grand", puis préciser les restrictions à apporter à ce langage. Il semble plus commode de faire porter ces restrictions sur les "marqueurs de phrases" que sur les phrases elles-mêmes, aussi : après avoir donné des notions rapides sur :

- les ramifications ;
- les fonctions récursives de ramifications et en particulier les prédicats récursifs ;

nous donnons une telle définition de syntaxe d'Algol 60 (Chapitre II) ainsi que celle d'un sous-ensemble d'Algol 68 obtenu en excluant les modes infinis et les formats (Chapitre III).

# CHAPITRE I

# GENERALITES

## 1.1 RAMIFICATIONS

Introduction : Les structures arborescentes qui constituent les marqueurs de phrase définis par une grammaire de Chomsky peuvent être décrits comme des arborescences, munies d'une orientation dans le sens de la lecture, dont les nœuds sont étiquetés par des symboles de l'alphabet V réunion de l'alphabet terminal et de l'alphabet non terminal de la grammaire. Il est commode d'accepter plusieurs racines, pour permettre des axiomes non nécessairement réduits à une lettre, et surtout pour faciliter la théorie algébrique qui va être faite.

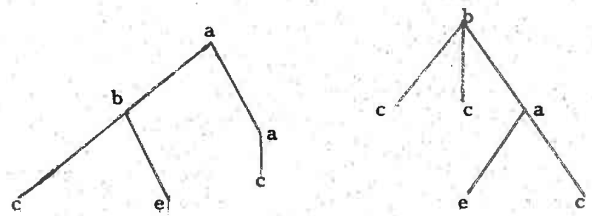



Figure 1

Nous nommerons ramifications sur V les êtres que nous venons de décrire intuitivement et $\hat{V}$ leur ensemble. Nous en donnerons une définition axiomatique après avoir dégagé les propriétés qui permettent de les caractériser.

Deux lois de composition opèrent naturellement sur les ramifications
a) une loi de composition interne (notée +), la concaténation, qui juxtapose deux ramifications à m et n "racines" en une ramification à m+n racines ; ainsi la ramification de la figure 1 est obtenue par concaténation de deux ramifications à une seule racine ; cette loi est associative et on introduit la ramification vide, notée $\wedge$, qui en est élément neutre.
b) Une loi de composition externe (notée $\chi$), ou enracinement, qui à un élément a de V et une ramification r, associe la ramification obtenue en ajoutant à r une racine étiquetée a ; la figure 2, par exemple, schématise $a \chi r$ lorsque r est la ramification de la figure 1.

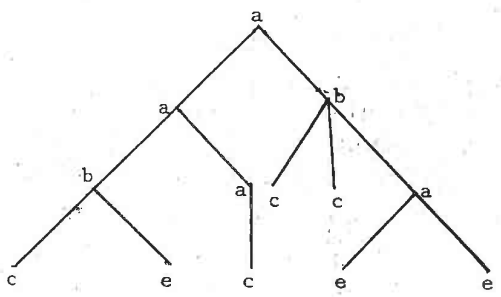



Figure 2

En particulier a $\chi$ $\Lambda$ est la ramification réduite à un seul point étiqueté
on peut identifier a $\chi$ $\Lambda$ et a. Avec cette convention et l'habituelle conve
tion de priorité de $\chi$ sur +, la ramification de la figure 2 s'écrit :

$$a \chi (a \chi (b \chi (c + e) + a \chi c) + b \chi (c + c + a \chi (e + e)))$$

Le mode de construction des ramifications au moyen de ces deux l
est précisé par le fait que toute ramification non vide sur V, r, s'écrit
manière unique

$$r = a \chi s + t \qquad \text{avec} \quad a \in V,\ s \in \hat{V},\ t \in \hat{V}.$$

De plus, les ramifications ont une "taille" finie, que les opération
font croître.

1.1.1 Définition axiomatique de $\hat{V}$:

Définition 1 : On appelle binoïde sur V tout ensemble muni

a) d'une loi de composition interne associative, avec un élément neutr
b) d'une loi de composition externe à opérateurs dans V.

Axiomes

1) $\hat{V}$ est un binoïde sur V ;
2) Soit $r \neq \Lambda$ une ramification de $\hat{V}$ ; il existe $a \in V$, $s \in \hat{V}$, $t \in \hat{V}$, uniques, tels que $r = a \chi s + t$ ;
3) Il existe une application $\nu$ de $\hat{V}$ dans l'ensemble des enti
naturels, telle que :



$$\begin{cases} \nu(\Lambda) = 0, \\ \nu(r+s) > \nu(r) \text{ si } s \neq \Lambda,\ \nu(r+s) > \nu(s) \text{ si } r \neq \Lambda, \\ \nu(a \chi r) > \nu(r) \end{cases}$$

Il résulte de (2) et (3) que $\nu(r)$ est strictement positif lorsque $r \neq \Lambda$.
D'autre part, d'après (2), $a \chi \Lambda = b \chi \Lambda$ entraine $a = b$ : on pourra identifier $a \chi \Lambda$ et a ; ainsi V sera une partie de $\hat{V}$.

On démontre [PAIR - QUERE] que $\hat{V}$ existe et est unique à un isomorphisme près.

Définition 2 : Les éléments de $\hat{V}$ sont appelés ramifications sur V.

Définition 3 : Nous appelons pseudo-arborescence sur V une ramification de $\hat{V}$ à une seule racine. Nous noterons A (V) l'ensemble des pseudo-arborescences sur V

$$A(V) \subset \hat{V}.$$

L'outil essentiel pour l'étude de $\hat{V}$ est l'ensemble des fonctions définies par récurrence.

Proposition 1 Soient un ensemble E, un élément $e_0$ de E, une application $\psi$ de $A(V) \chi \hat{V} \chi E^2$ dans E et pour tout $a \in V$ une application $\psi_a$ de $\hat{V} \chi E$ dans E. Il existe une application f de $\hat{V}$ dans E, et une seule, telle que :

a) $f(\Lambda) = e_0$
b) $(\forall s \in A(\nu),\ t \in \hat{V},\ t \neq \Lambda)\ f(s+t) = \psi(s, t, f(s), f(t))$.
c) $(\forall a \in V,\ s \in \hat{V})\ f(a \chi s) = \psi_a(s, f(s))$.

En effet, d'après les axiomes 2 et 3, ces conditions définissent f, de manière unique, successivement sur les ensembles de ramifications r telles que $\nu(r)$ soit égal à 0 ($r = \Lambda$), 1, 2 etc.

Nous donnons maintenant des exemples de fonctions définies ainsi et utiles dans toute la suite.

Monoïde sur V

Nous appelons monoïde libre sur V l'ensemble des suites finies d'éléments de V (ou mots sur V), nous le notons $V^*$. $V^*$ est muni de la loi de composition interne de concaténation qui est notée, ici par simple juxtaposition.

## 1.2 FONCTIONS USUELLES DE RAMIFICATIONS

### 1.2.1 Mot des racines d'une ramification :

Soit $\rho$, application de $\hat{V}$ dans $V^{\pm}$, définie par :

$$\rho(\Lambda) = \Lambda$$
$$\rho(s+t) = \rho(s)\,\rho(t)$$
$$\rho(a \times s) = a$$

$\rho(r)$ s'appelle mot des racines de r. Pour la ramification de la figure 1 mot des racines et ab.

### 1.2.2 Mot des feuilles :

Soit $\varphi$ application de $\hat{V}$ dans $V^{\pm}$ définie par :

$$\varphi(\Lambda) = \Lambda$$
$$\varphi(s+t) = \varphi(s)\,\varphi(t)$$
$$\varphi(a \pm s) = \text{si } s = \Lambda \text{ alors } a \text{ sinon } \varphi(s)$$

$\varphi(r)$ s'appelle mot des feuilles de r. Pour la ramification de la figure 1 mot des feuilles est cecccee.

### 1.2.3 Première composante d'une ramification

Soit $c_1$ l'application de $\hat{V}$ dans $\hat{V}$, définie par :

$$c_1(\Lambda) = \Lambda$$
$$c_1(s+t) = s$$
$$c_1(A \times s) = A \times s$$

$c_1(r)$ s'appelle première composante de r. Pour la ramification de la figure 1 $c_1(r) = a \times (b \pm (c + e) + a \times e)$.

### 1.2.4 Première queue d'une ramification

Soit $d_1$ l'application de $\hat{V}$ dans $\hat{V}$, définie par :

$$d_1(\Lambda) = \Lambda$$
$$d_1(s+t) = t$$
$$d_1(A \times s) = \Lambda$$

$d_1(r)$ s'appelle première queue de r. Pour la ramification de la figure 1 $d_1(r) = b \times (c + c + a \times (e + c))$.

Propriété 1 $\quad r = c_1(r) + d_1(r)$

en effet d'après l'axiome II $\quad \forall r \in \hat{V} \quad \exists a \in V,\ s, t \in \hat{V}$ uniques tels



$r = a \times s + t$

$c_1(r) = c_1(a \times x + t) = c_1(a \times s) = a \times s$

$d_1(r) = d_1(a \times s + t) = d_1(a \times s) + t = \Lambda$

Propriété 2 $\quad c_1\,c_1(r) = c_1(r)$ et $d_1\,c_1(r) = \Lambda$

Tout $r \in V$ tel que $d_1\,c_1(r) = \Lambda$ est une pseudo-arborescence sur $\hat{V}$.

### 1.2.5 Dernière composante d'une ramification

Soit $c_{-1}$ l'application de V dans V définie par :

$$c_{-1}(\Lambda) = \Lambda$$
$$c_{-1}(s+t) = c_{-1}(t)$$
$$c_{-1}(A \times s) = A \times s$$

$c_{-1}$ s'appelle la dernière composante de la ramification

### 1.2.6 Tête d'une ramification

Soit $d_{-1}$ l'application de $\hat{V}$ dans $\hat{V}$ définie par :

$$d_{-1}(\Lambda) = \Lambda$$
$$d_{-1}(s+t) = s + d_{-1}(t)$$
$$d_{-1}(A \times s) = \Lambda$$

$d_{-1}$ s'appelle la tête de la ramification r.

### 1.2.7 Longueur d'une ramification

Soit l'application de V dans l'ensemble des entiers naturels notée $|\ |$ et définie par :

$|\Lambda| = 0$

$|s+t| = |s| + |t|$

$|a \times s| = 1$

$|r|$ s'appelle la longueur de la ramification r.

Pour la ramification de la figure 1 $\quad |r| = 2$.

### 1.2.8 Ième queue de ramification

Soit l'application dernière queue définie précédemment, nous appelons ième queue d'une ramification ; l'application $d_i$ e $\hat{V}$ dans $\hat{V}$ définie par :

$$d_i(r) = d_1^i(r)$$

on note la composition de deux fonctions par simple juxtaposition ainsi :

$c_1\, d_1^3\,(r)$ représente $c_1\ (d_1\ (d_1\ (d_1\ (r))))$

1.2.9 Ième composante d'une ramification s

Soit $c_1$ et $d_1$ les applications définies précédemment, nous appelons ième composante d'une ramification l'application $c_i$ de $\hat{V}$ dans $\hat{V}$ définie par :

$$c_i\,(r) = c_1\, d_1^{i-1}\,(r).$$

Propriété 3 $\forall r \in \hat{V}$ si $n = |r|$ alors il existe $s_1, s_2, \ldots s_n \in \hat{V}$ tel $r = s_1 + s_2 + \ldots + s_n$ avec $s_i = c_i\,(r)$.
La démonstration par récurrence est basée sur la longueur de r.

1.2.10 Branches d'une ramification

Soit b l'application de $\hat{V}$ dans $\hat{V}$, définie par :

$b\,(\Lambda) = \Lambda$

$b\,(s+t) = b\,(s) + b\,(t)$

$b\,(A \times s) = s$

b (r) s'appelle les branches de la ramification r.

1.2.11 Ième rameau d'une ramification

Soit $c_i'$ l'application de $\hat{V}$ dans $\hat{V}$, définie par :

$$c_i'\,(A \times s) = c_i\,(s).$$

1.2.12 Famille de prédecesseur $b \in V$

Soit $b \in V$, on appelle familles de prédecesseur b dans ramification r les mots de l'ensemble $F_b\,(r)$, où $F_b$ est l'application dans l'ensemble des parties de $V^*$ définie par :

$F_b\,(\Lambda) = \emptyset$

$F_b\,(s+t) = F_b\,(s) \cup F_b\,(t)$

$F_b\,(a \times s) =$ si $a = b$ alors $\{\rho\,(s)\} \cup F_b\,(s)$
sinon $F_b\,(s)$.

1.2.13 Nombre d'occurrences d'une pseudo-arborescence r dans ramification

Soit m l'application de $\hat{V} \times \hat{V}$ dans $\hat{V}$ définie par :

$m\,(r, \Lambda) = \Lambda$



$m\,(r, s_1 + s_2) = m\,(r, s_1) + m\,(r, s_2)$

$m\,(r, A \times s) =$ si $r = A \times s$ alors r sinon $\Lambda$.

Nous appellerons nombre d'occurences l'application de $\hat{V} \times \hat{V}$ dans N définie par

$$n\,(r, s) = |m\,(r, s)| \quad \forall\,(r, s) \in \hat{V} \times \hat{V}.$$

1.3 FONCTIONS RECURSIVES PRIMITIVES ET PREDICATS RECURSIFS PRIMITIFS [QUERE ]

Nous avons cité précédemment quelques fonctions sur $\hat{V}$ ou $\hat{V} \times \hat{V}$, toutes ces fonctions sont des cas particulier de celles que nous allons appeler fonctions récursives primitives.

Notons $\mathcal{F}^k$ l'ensemble de fonctions de $\hat{V}^k$ dans $\hat{V}$ en convenant que $\mathcal{F}^0 = \hat{V}$.

Opérateurs a) Nous appelons composition l'opérateur $\Gamma$, qui quels que soient les entiers naturels k et $\ell$, associe aux fonctions $g_1, \ldots, g_\ell$ de $\mathcal{F}^k$ et g de $\mathcal{F}^\ell$ la fonction $f = \Gamma\,(g, g_1, \ldots, g_\ell)$ de $\mathcal{F}^k$ définie par :

$$(\forall r_1, \ldots, r_k \in \hat{V})\ f\,(r_1, \ldots, r_k) = g\,(g_1\,(r_1, \ldots, r_k),\ g_2\,(r_1, \ldots, r_k),\ \ldots,\ g_\ell\,(r_1, \ldots, r_k)).$$

b) Nous appelons opérateur de récurrence l'opérateur R, qui quel que soit l'entier naturel k associe aux fonctions $g \in \mathcal{F}^k$, $\psi \in \mathcal{F}^{k+4}$, $\psi_a \in \mathcal{F}^{k+2}$ pour tout $a \in V$, la fonction $f = R\,(g, \psi, (\psi_a)\ a \in V)$ de $\mathcal{F}^{k+1}$ définie par :

$(\forall r_1, \ldots, r_k, s \in \hat{V},\ r \in A\,(V), a \in V)$ $f\,(r_1, \ldots, r_k, \Lambda) = g\,(r_1 \ldots r_k)$

$$f\,(r_1, \ldots r_k, r+s) = \psi\,(r_1, \ldots r_k, r, s, f(r_1, \ldots r_k, r), f(r_1, \ldots r_k, s))$$

$$f\,(r_1, \ldots, r_k, a \times s) = \psi_a\,(r_1, \ldots r_k, r, f\,(r_1, \ldots, r_k, s))$$

Fonctions de base :

Nous prenons pour fonctions de base les fonctions suivantes :

1) $\Lambda \in \mathcal{F}^0$

2) $\forall r_1, r_2 \in \hat{V}$ la fonction f telle que $f\,(r_1, r_2) = r_1 + r_2$

3) Pour tout $a \in V$ la fonction $f_a$ définie par $\forall r \in \hat{V}$ $f_a\,(r) = a \times r$.

4) Les fonctions projections c'est-à-dire, pour tout $k > 0$ et tout $1 \le i \le k$ les fonctions $p_i^k$ de k variables définies par :

$$(\forall r_1 \ldots r_k \in \hat{V})\ p_i^k\,(r, \ldots, r_k) = r_i.$$

Définition :

L'ensemble des fonctions récursives primitives (sur $\hat{V}$) est le plu petit ensemble $\mathfrak{R}$ contenant les fonctions de base et stable par les opérateurs de composition et de récurrence.

Dans toute la suite, nous notons 0 un élément particulier de V.

Prédicats

Nous appelons prédicat sur $\hat{V}^k$ une application de $\hat{V}^k$ dans V ne prenant que les deux valeurs $\Lambda$ et 0. Nous interpréterons $\Lambda$ comme la valeur vrai et 0 comme la valeur faux.

Exemple : égal (r, s) = si r = s alors $\Lambda$ sinon 0, on montre dans [QUERE] que :

a) si f et g sont des prédicats récursifs primitifs, il en est de même pour les prédicats non f, f et g, f ou g ; les opérateurs non, et, ou ayant leur signification habituelle.

b) égal est un prédicat récursif primitif

c) si p est un prédicat récursif primitif sur $\hat{V}^k$ et f, g deux fonctions récursives primitives, sur $\hat{V}^k$ la fonction définie par :

$$h(r_1, \ldots, r_k) = \text{si } p(r_1, \ldots, r_k) \text{ alors } f(r_1, \ldots, r_k) \text{ sinon } g(r_1, \ldots, r_k)$$

est aussi récursive primitive.

L'élément 0 de V nous servira d'autre part dans la définition des "prédicats de reconnaissance" pour indiquer l'exitence d'une "erreur" dans le programme étudié.

## 1.4 GRAMMAIRE DE CHOMSKY

Une grammaire de Chomsky est un quadruplet G = (T, N, ::=, X) où

- T est un ensemble fini non vide appelé vocabulaire terminal ;
- N est un ensemble fini non vide, disjoint de T appelé vocabulaire auxiliaire où non terminal.
- ::= est une relation binaire entre N et $V^{\star}$, (V = T $\cup$ N), telle que le nombre de couples en relation (ou règles) soit fini.
- X est un élément distingué dans N, appelé axiome de G.



Le langage engendré par une grammaire de Chomsky G est l'ensemble des mots $\alpha \in T^{\star}$ tels que :

$$X \overset{\star}{\rightarrowtail} \alpha$$

où la relation $\overset{\star}{\rightarrowtail}$ ("dérive de") est la fermeture transitive de la relation $\rightarrowtail$ ("se réecrit") qui est définie par :

$$(\forall \alpha, \beta \in V^{\star}) \quad \alpha \rightarrowtail \beta \Leftrightarrow (\exists A \in N, \lambda \in V^{\star}, \lambda' \in V^{\star}, \gamma \in V^{\star})$$
$$\text{tel que } \alpha = \lambda A \lambda'$$
$$\beta = \lambda \gamma \lambda'$$
$$A ::= \gamma.$$

Les langages engendrés par une grammaire de Chomsky sont appelés langages de Chomsky.

## 1.5 BILANGAGE ENGENDRE PAR UNE GRAMMAIRE

Une ramification sur V est engendrée par la grammaire G lorsque :

a) chacune de ses familles $\alpha$ de prédecesseur a $\in$ V vérifie a::=$\alpha$

b) sont mot des feuilles est un mot sur T

c) sont mot des racines est l'axiome.

On appelle bilangage sur V une partie de $\hat{V}$ et bilangage engendré par une grammaire G l'ensemble des ramifications sur $\hat{V}$ engendré par la grammaire G.

## 1.6 APPLICATION A LA DEFINITION D'UN LANGAGE

Pour définir un langage L nous donnons :

a) une grammaire de Chomsky $G_0$ ;

b) un vocabulaire V contenant celui de $G_0$ ;

c) un prédicat récursif F sur $\hat{V}$.

Soit $B_0$ le bilangage engendré par $G_0$, par définition, L est l'ensemble des mots des feuilles des ramifications r de $B_0$ tel que :

$$F(r) = \Lambda ;$$

$$L = \varphi(\{r \in B_0 \mid F(r) = \Lambda\}).$$

# CHAPITRE II

## DEFINITION DE LA SYNTAXE D'ALGOL 60

Pour définir Algol 60, après avoir donné la grammaire $G_0$ et le vocabulaire V on donne le prédicat de reconnaissance F.

La grammaire $G_0$ est décrite selon la notation de Bachus [BACKUS]. Elle est déduite de celle du rapport Algol 60 [A. C. M. Janv. 63]. On peut constater certaines différences dans la decription de ces deux grammaires. Ces différences ont pour but de faciliter la description du prédicat de reconnaissance.

Le prédicat de reconnaissance est décrit au moyen de trois fonctions récursives, qui sont décrites elles-même au moyen de fonctions plus simples. Certaines n'apparaissent pas, dans leur description, sur une des fonctions récursives primitives, cependant on peut les y ramener toutes [cf QUERE]

## 2.1 GRAMMAIRE $G_0$

### a) Symboles de base

< Symbole de base > ::= < symbole de commentaire > | ;

< symbole de commentaire > ::= < symbole de commentaire final > | fin |

< symbole de commentaire final > ::= < non guillemet > | ' | '

< symbole de chaine > ::= < non guillemet > | ; | fin | sinon

< non guillemet > ::= < lettre > | < chiffre > | < valeur logique > | < délimi
< declareur > | commentaire | debut

< lettre > ::= A | B | C | D | E | F | G | H | I | J | K | L | M | N | Ø | P | Q | R |
U | V | W | X | Y | Z

< chiffre > ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

< valeur logique > ::= vrai | faux

< délimiteur > ::= < opérateur de relation > | < opérateur additif > |
| ± | / | ÷ | ↑ | ≡ | ⊃ | ∨ | ∧ | ¬ | ( | ) | [ | ] |
| allera | si | alors | pour | faire | , | . | $_{10}$ | : | := | ␣ | pas | jusc
| tantque |

<declareur> ::= | rémanent | booléen | entier | réel | tableau | aiguillage |
| procédure | chaine | étiquette | valeur

< opérateur additif > ::= + | -

< opérateur de relation > ::= ≤ | < | = | > | ≥ | ≠

### b) Commentaires - séparateurs

< commentaire > ::= commentaire < texte de commentaire >

< texte de commentaire > ::= < symbole de commentaire > |
< symbole de commentaire > < texte de commentaire >

< commentaire de fin > ::= < symbole de commentaire final > |
< symbole de commentaire final > < commentaire de fin >

< chaîne de lettre > ::= < lettre > | < lettre > < chaine de lettre >

< Séparateur > ::= , | ) < chaine de lettre > (

< point virgule > ::= ; | ; < commentaire > ;

< début > ::= début | début < commentaire >

< fin > ::= fin < commentaire de fin > | fin



### c) Programme

< programme > ::= < bloc > | < instruction composée >

< bloc > ::= < début de bloc > < point virgule > < fin d'instruction composée > |
< étiquette > : < bloc >

< début de bloc > ::= < début > < déclaration > |
< début de bloc > < point virgule > < déclaration >

< instruction composée > ::= < début > < fin d'instruction composée >
< étiquette > : < instruction composée >

< fin d'instruction composée > ::= < instruction > < fin > | < fin >
< instruction > < point virgule > < fin d'instruction composée >

### d) Instructions

< instruction > ::= < instruction inconditionnelle > |
< instruction conditionnelle > |
< instruction Pour >

< instruction inconditionnelle > ::= < instruction de base > |
< instruction composée > |
< bloc >

< instruction de base > ::= < instruction non étiquetée > |
< étiquette > : < instruction de base > |
< étiquette > :

< instruction non étiquetée > ::= < instruction d'affectation > |
< instruction allera > |
< instruction procédure >

< étiquette > ::= < identificateur > | < entier sans signe >

### Instruction d'affectation

< instruction d'affectation > ::= < liste de parties gauches > < expression >

< liste de parties gauches > ::= < partie gauche > := |
< partie gauche > := < liste de parties gauches >

< partie gauche > ::= < identificateur > | < identificateur indicé >

Instruction allera

< instruction allera > ::= allera < expression de désignation >

< expression de désignation > ::= < expression >

Instruction procédure

< instruction procédure > ::= < appel de fonction > | < identificateur >

Instruction conditionnelle

< instruction conditionnelle > ::= < instruction si > |
< instruction si > sinon < instruction > |
< instruction si > sinon |
< proposition si > < instruction pour > |
< étiquette > : < instruction conditionnelle >

< instruction si > ::= < proposition si > |
< proposition si > < instruction inconditionnelle >

< proposition si > ::= si < expression booléenne > alors

Instruction Pour

< instruction pour > ::= < proposition pour > < instruction > |
< proposition pour > |
< étiquette > : < instruction pour >

< proposition pour > ::= Pour < partie gauche > := < liste de pour > faire

< liste de pour > ::= < élément de liste de pour > |
< élément de liste de pour >, < liste de pour >

< élément de liste de pour > ::= < expression arithmétique > |
< expression arithmétique > pas < expression arithmétique
jusqua < expression arithmétique > |
< expression arithmétique > tant que < expression booléen



e) Expressions

< expression booléenne > ::= < expression >

< expression arithmétique > ::= < expression >

< expression de désignation > ::= < expression >

< expression > ::= < expression simple > | < proposition si > < choix >

< choix > ::= < expression simple > sinon < expression >

Expression simple

< expression simple > ::= < implication > | < expression simple > ≡ <implication>

< implication > ::= < terme booléen > | < implication > ⊃ < terme booléen >

< terme booléen > ::= < facteur booléen > | < terme booléen > ∨ <facteur booléen>

< facteur booléen > ::= < secondaire booléen > |
< facteur booléen > ∧ < secondaire booléen >

< secondaire booléen > ::= < relation > | ¬ < relation >

< relation > ::= < expression arithmétique simple > |
< relation booléenne >

< relation booléenne > ::= < expression arithmétique simple >
< opérateur de relation > < expression arithmétique simple >

Expression arithmétique simple

< expression arithmétique simple > ::= < terme > | < opérateur additif > < terme > |
< expression arithmétique simple > <opérateur additif> <terme>

< terme > ::= < facteur > | < terme > * < facteur > |
< terme > / < facteur > | < quotient entier >

< quotient entier > ::= < terme > ÷ < facteur >

< facteur > ::= < primaire > | < facteur > ↑ < primaire >

< primaire > ::= < valeur logique > | < nombre sans signe > |
< identificateur > | < identificateur indicé > |
< appel de procédure > | (< expression >)

Identificateur

< identificateur > ::= < lettre > | < identificateur > < lettre > |
< identificateur > < chiffre >

Identificateur indicé

< identificateur indicé > ::= < identificateur > [< suite d'indices>]
< suite d'indices > ::= < indice > | < indice > , < suite d'indices >
< indice > ::= < expression arithmétique >

Appel de procédure

< appel de procédure > ::= < identificateur > ( < liste de paramètres > )
< liste de paramètres > ::= < paramètre > |
< paramètre > < séparateur > < liste de paramètres
< paramètre > ::= < expression > | < chaine >

Nombre sans signe

< nombre sans signe > ::= < nombre décimal > |
< facteur de cadrage > |
< nombre décimal > < facteur de cadrage >
< nombre décimal > ::= < entier sans signe > | < partie décimale > |
< entier sans signe > < partie décimale >
< facteur de cadrage > ::= $_{10}$ < entier >
< partie décimale > ::= . < entier sans signe >
< entier > ::= + < entier sans signe > | - < entier sans signe > |
< entier sans signe >
< entier sans signe > ::= < chiffre > | < chiffre > < entier sans signe >

Chaine

< chaine > ::= ' < chaine ouverte > ' | ''
< chaine ouverte > ::= < chaine propre > | ' < chaine ouverte > ' |
< chaine ouverte > < chaine ouverte >
< chaine propre > ::= < symbole de chaine > |
< symbole de chaine > < chaine propre >



f) Déclarations

< déclaration > ::= < déclaration simple > |
< déclaration de tableau > |
< déclaration d'aiguillage > |
< déclaration de procédure >

Déclaration simple

< déclaration simple > ::= < type > < suite d'identificateurs >
< suite d'identificateurs > ::= < identificateur > |
< identificateur > , < suite d'identificateurs >
< type > ::= réel | entier | booléen
rémanent réel | rémanent entier | rémanent booléen

Déclaration de tableau

< déclaration de tableau > ::= < type de tableau > < liste de tableaux >
< liste de tableaux > ::= < section de tableau > |
< section de tableau > , < liste de tableaux >
< section de tableau > ::= < suite d'identificateurs > < volume >
< volume > ::= [ < suite de paires de bornes > ]
< suite de paires de bornes > ::= < paire de bornes > |
< paire de borne > , < suite de paires de bornes >
< paire de bornes > ::= < expression arithmétique > : < expression arithmétique >
< type de tableau > ::= tableau | réel tableau | entier tableau |
booléen tableau | rémanent réel tableau |
rémanent entier tableau |
rémanent booléen tableau

Déclaration d'aiguillage

< déclaration d'aiguillage > ::= aiguillage < identificateur > :=
< liste d'aiguillages >
< liste d'aiguillages > ::= < expression de désignation > |
< expression de désignation > , < liste d'aiguillages >

Déclaration de procédure

< déclaration de procédure > ::= < tête de procédure > < partie paramétr
< corps de procédure >

< tête de procédure > ::= < type de procédure > < identificateur de proc

< identificateur de procédure > ::= < identificateur >

< type de procédure > ::= procédure | réel procédure | entier procédure
booléen procédure

< partie paramètrique > ::= ( < liste de paramètres formels >) < partie va
< partie spécification > |
(< liste de paramètres formels >)<partie spécificatio
(< liste de paramètres formels >)

< liste de paramètres formels > ::= < paramètre formel > | < paramètre f
< séparateur > < liste de paramètres formels >

< paramètre formel > ::= < identificateur >

< partie valeur > ::= valeur < suite d'identificateurs > ;

< partie spécification > ::= < spécification > |
< spécification > < partie spécification >

< spécification > ::= < spécificateur > < suite d'identificateurs > ;

< spécificateur > ::= réel | entier | booléen |
réel tableau | entier tableau | booléen tableau
réel procédure | entier procédure | booléen procédu
tableau | procédure | aiguillage | étiquette |
chaine.

< corps de procédure > ::= < instruction >



## 2.2 VOCABULAIRE V :

Le vocabulaire V est la réunion du vocabulaire $V_0$ de la grammaire $G_0$ et de l'ensemble suivant :

{0, libre, ouvert, vide, enquete, rangée, proc, protype, externe}

0 représente la valeur d'échec ou d'erreur.

Les autres éléments que nous ajoutons ont pour but de faciliter la description du prédicat de reconnaissance.

## 2.3 PREDICAT DE RECONNAISSANCE

Introduction :

Le prédicat de reconnaissance a pour but de vérifier si un programme satisfait aux conditions de contexte. Celles-ci sont de deux sortes :

a) les conditions de contexte globales sur les déclarations relatives à chaque bloc.

b) les conditions de contexte locales relatives aux expressions et aux instructions.

Pour vérifier les conditions du type b, il est notamment nécessaire de connaître les types des opérandes, des expressions. Nous aurons donc, deux phases principales, ou deux fonctions principales composant le prédicat récursif F. Une fonction T vérifiera les conditions de contexte globales et remplacera chaque identificateur par son type (nous l'appelons fonction de transport de déclaration) ; puis une fonction L effectuera les vérifications locales sur les expressions et les instructions.

Pour faciliter la description de ces deux fonctions nous les ferons précéder d'une fonction P "de préparation de la ramification", qui modifie notamment les déclarations de tableaux, de procédure et d'aiguillage et qui crée des déclarations d'étiquette.

F est donc le produit de trois fonctions :

$$F = L \,.\, T \,.\, P.$$

2.3.1 Préparation - Fonction P

- Simplifications générales
- création de déclarations d'étiquette
- modification des déclarations de tableau
- modification des déclarations d'aiguillage
- modification des déclarations de procédure.

2.3.1.1 Simplifications générales

Certains symboles ne sont que des délimiteurs permettant d'
lyser un programme. Dans la ramification obtenue par l'analyse, ces
symboles ne supportent plus aucune signification. Pour faciliter la des
tion des fonctions T et L nous allons les supprimer. Nous supprimeron
aussi les symboles non terminaux < point virgule > < début > < fin > et
< séparateur >.

2.3.1.2 Création de déclarations d'étiquette

La rencontre d'une étiquette devant une instruction ou
seule composante d'une instruction (l'instruction vide n'apparait pas da
la grammaire $G_0$) doit être considérée comme une déclaration d'identif
cateur ou d'entier sans signe du type étiquette. Nous remplaçons donc
occurrences d'étiquette par une déclaration simple.

2.3.1.3 Modification des déclarations de tableau

Nous considérons l'identificateur d'un tableau d'élément ré
(entier ou booléen) à une dimension comme un identificateur dont le typ
serait : rangée de réel (rangée d'entier, rangée de booléen) ; l'identific
teur d'un tableau à deux dimensions comme rangée de rangée de réel ..
Nous introduisons de nouveaux types et pour eux de nouveaux déclarate
(fonction δ 2.3.1.3'). D'autre part on transforme une déclaration de ta-
bleau en une suite de déclarations. A chaque section de tableau corresp
une déclaration (cf 2.3.1.3', 2.3.1.6 et 2.3.1.6').

Enfin dans une déclaration de tableau les bornes ne peuvent pas être
primées en fonction de variables, que si celles-ci sont déclarées dans
bloc plus grand. Pour faciliter la vérification de cette condition par la
tion T en (2.3.2.3), nous allons placer chaque paire de bornes sous la
pendance du symbole supplémentaire externe.



2.3.1' DEFINITION DE LA FONCTION P

$P(\Lambda) = \Lambda$

$P(r+s) = P(r) + P(s)$

$P(A \times r) = A \times P(r)$ sauf

2.3.1.1' A = point virgule, début, fin, séparateur, opérateur de relation, opérateur additif, $\star$, /, +, ↑, ≡, ⊃, ∨, ∧, ¬, [, ], (, ), ;, :=, ., $_{10}$, :, , allera, si, alors, pour, faire, pas, jusqua, tantque, sinon et rémanent

$P(A \times r) = \Lambda$

2.3.1.2' A = étiquette

$P(A \times r)$ = déclaration simple $\star$ (étiquette + r)

2.3.1.3' A = déclaration de tableau

$P(A \times r) = \delta(c_1 P(r), d_1 P(r)) + \text{ext}(r)$

A = liste de tableau

$P(A \times r) = P(r)$

A = section de tableau

$P(A \times r) = A \times (d_1 P(r) + c_1 P(r))$

A = suite de paire de bornes

$P(A \times r) =$ rangée $\times P(r)$

A = paire de bornes

$P(r) = \Lambda$.

2.3.1.4 Modification des déclarations d'aiguillage

Nous considérons les identificateurs d'aiguillage comme de identificateurs du type rangée d'étiquette aussi nous modifions les dé rations d'aiguillage de la même façon que pour les déclarations de ta (cf 2.3.1.4')

2.3.1.5 Modification des déclaration de procédure

Nous introduisons aussi de nouveaux types pour les procéd où interviennent, le type du résultat éventuel et, les types des param Nous associons donc à chaque identificateur de procédure un nouveau clareur (fonction $\mu$ 2.3.1.5', 2.3.1.11').

En même temps que nous calculons ce nouveau déclareur, nous ons des déclarations pour les paramètres en (2.3.1.8 et 2.3.1.8' fon $\pi$) vérifiant les conditions suivantes :

a) un paramètre ne peut figurer plus d'une fois en partie spécifi

b) un paramètre ne peut figurer plus d'une fois en partie valeur

c) tout paramètre en partie valeur doit obligatoirement être une tie spécification.

Les déclarations des paramètres sont obtenues à partir de leur cification.

Si un paramètre n'est pas spécifié on lui attribue le type libre.

Si un paramètre est spécifié tableau on le déclare ouvert afin qu puisse correspondre à n'importe quel tableau du même type.

Si un paramètre est spécifié aiguillage on le déclare rangée d'éti

Enfin si un paramètre est spécifié procédure on le déclare proc à paramètres libre .

Pour faciliter les vérifications de contexte sur l'identificateur d procédure (cf 2.3.3.5 a) et les paramètres (cf 2.3.3.1 d) nous créons deux déclarations d'identificateur de procédure.

- La première dont le type est placé sous la dépendance du sym protype, est placée dans un bloc contenant les déclarations des param et le corps de la procédure.

- La seconde dont le type et sous la dépendance du symbole proc placée à l'extérieur du bloc précédent.



2.3.1.4' A = déclaration d'aiguillage

$P(A \times r)$ = déclaration simple $*$ (rangée $*$ étiquette $+ c_2(r)) + c_3(r)$

2.3.1.5' A = déclaration de procédure

$P(A \times r)$ = déclaration simple $\times$ (proc $\times (c_1 P(r) + \mu b c_3 P(r)) + c_2 P(r))$
+ bloc $\times$ (déclaration simple $\times$ (protype $\times$ ($c_1 P(r) + \mu b c_3$
$P(r)) + c_2 P(r)) + d_3 P(r)$).

A = tête de procédure

$P(A \times r) = P(r)$

A = type de procédure

$P(A \times r)$ = si $c_1(r)$ = procédure alors vide sinon $c_1(r)$

A = identificateur de procédure

$P(A \times r) = r$

A = partie paramètrique

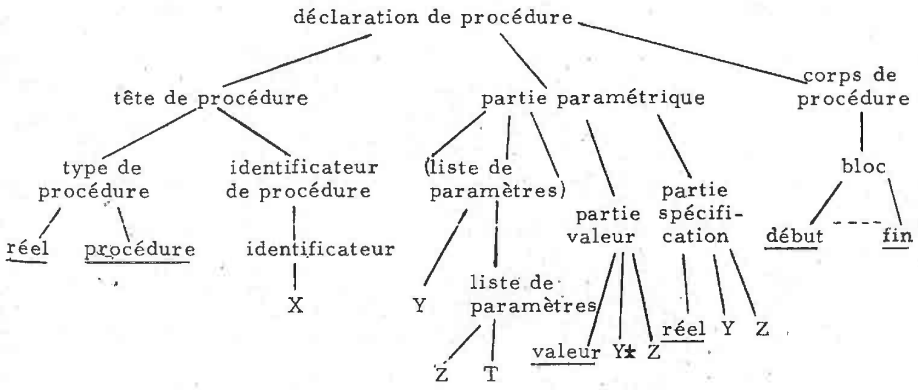
$P(A \times r) = A \times \pi (c_1 P(r), d_1 P(r))$

A = corps de procédure

$P(A \times r)$ = bloc $\times P(r)$

Exemple de transformation de délcaration de procédure (1)

réel procédure X (Y, Z, T) valeur Y, Z ; réel Y, Z ;
début ... fin



2.3.1.6 Modes des identificateurs de tableau

La fonction $\delta$ permet de déterminer le mode des identificateu de tableau.

Par la fonction P, (cf 2.3.1.3'), nous avons inversé le contenu section de tableau, en plaçant devant la suite des identificateurs le no de dimensions de ces tableaux.

La fonction $\delta$ permet de transformer la section de tableau en un déclaration où tous les tableaux sont de même type, de la forme

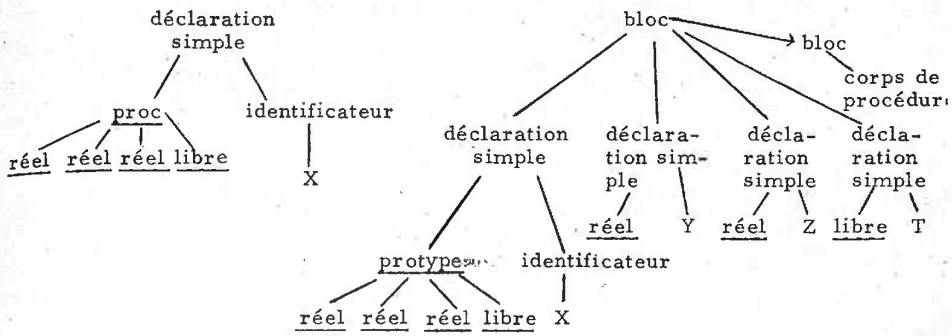
$$\underline{\text{rangée}} \times \underline{\text{rangée}} \times \cdots \times \begin{cases} \underline{\text{réel}} \\ \underline{\text{entier}} \\ \underline{\text{booléen}} \end{cases}$$

2.3.1.7 Séparation des bornes des déclarations de tableau

La fonction ext, extrait chaque paire de bornes contenue dan une déclaration de tableau, en la plaçant sous la dépendance de < exter



Après modification nous obtenons




2.3.1.6' Définition de la fonction $\delta$

$\delta\ (r, \Lambda) = \Lambda$

$\delta\ (r,\ s_1 + s_2) = \delta\ (r,\ s_1) + \delta\ (r,\ s_2)$

$\delta\ (r,\ A \times s) = A \times \delta\ (r, s)$ sauf

A = section de tableau

$\delta\ (r,\ A \times s)$ = déclaration simple $\times\ \delta\ (r, s)$

A = rangée

$\delta\ (r,\ A \times s)$ = si $s = \Lambda$ alors $A \times r$ sinon

$A \times \delta\ (r, s)$

2.3.1.7' Définition de la fonction ext

ext $(\Lambda) = \Lambda$

ext $(r + s)$ = ext $(r)$ + ext $(s)$

ext $(A \times r)$ = ext $(r)$ sauf

A = paire de bornes

ext $(A \times r)$ = externe $\times$ r.

(1) p. 23 - Nous avons pour cet exemple représenté une ramification, en ne conservant que les nœuds essentiels permettant d'illustrer la transformation des déclarations de procédure.

2.3.1.8 Déclaration des paramètres d'une procédure

Au cours de la transformation de la partie paramètrique d'un déclaration de procédure, on crée une déclaration pour chaque paramè trouvé dans la liste des paramètres.

Le type d'un paramètre est déduit de sa spécification éventuelle lo que les conditions de contexte sur les paramètres en partie valeur et pa tie spécification, sont respectées. La fonction $\mathcal{E}$ permet d'effectuer la rification de la condition de contexte citée en 2.5.1.10, et de déduire l type du paramètre.

La fonction $\omega$ recherche si un paramètre figure en partie valeur $[\omega\ (r, c_1\ (s))]$ et en partie spécification $[\omega\ (r,\ c_2(s))]$.

2.3.1.9 Recherche d'un identificateur

La fonction $\omega$ recherche un identificateur :

1) dans une partie spécification : si l'identificateur est trouvé une seul fois , le résultat est la spécification s'il n'est pas trouvé le résultat $\Lambda$ sinon il y a erreur.

2) Dans une partie valeur, si l'identificateur est trouvé une seule fois l résultat est valeur, s'il n'est pas trouvé le résultat est $\Lambda$ , sinon il erreur.

3) Dans une déclaration simple : si l'identificateur est trouvé une seule fois le résultat est le type de la déclaration, s'il n'est pas trouvé l résultat est $\Lambda$, sinon il y a erreur. Ce cas particulier n'est utilisé pour la fonction T.

2.3.1.10 Recherche du type d'un paramètre

Le type d'un paramètre d'une fonction est déterminé selon l règles suivantes

- si un paramètre n'est pas spécifié, son type est libre
- si un paramètre est spécifié au moins deux fois c'est une erreur (d'où la condition $\mathcal{E}\ (r,\ s_1 + s_2) = 0$)



2.3.1.8' Définition de la fonction $\pi$

$\pi\ (\Lambda, s) = \Lambda$

$\pi\ (r_1 \dotplus r_2, s) = \pi\ (r_1, s) \dotplus \pi\ (r_2, s)$

$\pi\ (A \ast r, s) = \pi\ (r, s)$ sauf

A = paramètre

$$\pi\ (A \uparrow r, s) = \begin{cases} \text{si } \omega\ (r,\ c_1(s)) \neq 0 \text{ alors} \\ \text{déclaration simple} \ast (\mathcal{E}(\omega(r, c_1(s)), \omega(r, c_2(s))) \dotplus r) \\ \text{sinon } 0 \end{cases}$$

2.3.1.9' Définition de la fonction $\omega$

$\omega\ (r, \Lambda) = \Lambda$

$\omega\ (r,\ s_1 \dotplus s_2) = \omega\ (r, s_1) \dotplus \omega\ (r, s_2)$

$\omega\ (r,\ A \ast s) = \omega\ (r, s)$ sauf

A = identificateur

$\omega\ (r,\ A \ast s) =$ si $r = s$ alors $s$ sinon $\Lambda$

A = spécification

$\omega\ (r,\ A \ast s) =$ si $|\omega\ (r, c_2(s))| = 1$ alors $c_1\ (s)$
si $\omega\ (r, c_2\ (s)) = \Lambda$ alors $\Lambda$
sinon 0

A = partie valeur

$\omega\ (r,\ A \ast s) =$ si $|\ \omega\ (r, s)\ | = 1$ alors valeur
si $\omega\ (r, s) = \Lambda$ alors $\Lambda$
sinon 0

A = déclaration simple

$\omega\ (r, A \ast s) =$ si $|\omega\ (r, s)| = 1$ et $\rho\omega\ (r, s) =$ type alors $b\ \omega\ (r, s)$
si $\omega\ (r, s) = \Lambda$ alors $\Lambda$ sinon 0

2.3.1.10' Définition de la fonction $\xi$

$\mathcal{E}(r, \Lambda) =$ si $r = \Lambda$ alors libre sinon 0

$\mathcal{E}(r, 0) = 0$

$\mathcal{E}(r,\ s_1 + s_2) = 0$

- si un paramètre est spécifié réel, entier, booléen, étiquette ou chaine son type est alors réel, entier, booléen, étiquette ou chai
- si un paramètre est spécifié tableau son type est ouvert x réel, s est spécifié réel tableau, entier tableau, booléen tableau alors le type est ouvert x réel, ouvert x entier, ouvert x booléen, on utili ouvert pour indiquer que le paramètre doit correspondre à un tab dont on ne connait pas le nombre de dimensions.
- Si un paramètre est spécifié procédure et s'il ne figure pas en pa valeur alors son type est celui d'une procédure à résultat vide et paramètres libres.
- Si un paramètre est spécifié réel procédure (entier procédure ou booléen procédure) alors :

  s'il figure en partie valeur son type est alors celui d'une pro dure a résultat réel (entier ou booléen) et sans paramètre.

  s'il ne figure pas en partie valeur son type est alors celui d'une proc dure à résultat réel (entier ou booléen) et à paramètres libres.
- Si un paramètre est spécifié aiguillage et s'il ne figure pas en par valeur son type est rangée x étiquette.

2.3.1.11 Type des paramètres dans le type de la procédure

Pour obtenir le déclareur d'un identificateur de procédure o rassemble le type de chacun des paramètres.



$\mathcal{E}(r, A \times s)$ = si A = spécificateur alors
si $\rho$ (s) = réel, entier, booléen alors s
si s = étiquette ou chaine alors type s
si s = tableau alors type x ouvert x réel
si $c_2$ (s) = tableau alors type ✱ ouvert x b $c_1$ (s)
si $c_2$ (s) = procédure alors | si r = $\Lambda$ alors type ✱ proc ✱ ($bc_1$(s)+libre
sinon 0
si s = procédure alors | si r = $\Lambda$ alors type ✱ proc ✱ (réel + libre)
sinon 0
si s = aiguillage alors | si r = $\Lambda$ alors type ✱ rangée ✱ étiquette
sinon 0
sinon 0

2.3.1.11' Définition de la fonction $\mu$

$\mu$ ($\Lambda$) = $\Lambda$
$\mu$ (r + s) = $\mu$ (r) + $\mu$ (s)
$\mu$ (A✱ r) = $\mu$ (r)
sauf
A = déclaration simple
$\mu$ (A✱r) = b $c_1$ (r)

## 2.3.2 VERIFICATIONS GLOBALES - TRANSPORT DES DECLARATIONS

- Vérification de l'unicité des déclarations d'identificateur
- cas particulier des étiquettes représentées par un entier
- vérification de la condition sur les bornes d'un tableau.

### 2.3.2.1 Vérification de l'unicité des déclarations d'identificateurs

Pour qu'un programme soit correct il faut qu'il vérifie les deux conditions suivantes :

- Tout identificateur doit être déclaré.
- Un identificateur ne peut être déclaré plus d'une fois dans chaque bloc.

Pour vérifier les conditions, dans chaque bloc, on recherche un identificateur y compris ceux contenus dans les déclarations est déclar

- S'il n'est déclaré qu'une seule fois, on le remplace par son
- S'il est déclaré plusieurs fois c'est une erreur.

Après l'étape 2 il ne reste plus que les identificateurs non déc

### 2.3.2.2. Cas particulier des étiquettes de type entier

Une étiquette peut être représentée par un entier sans signe : il en résulte qu'il peut exister une ambiguïté sur la signification des entie

Pour lever cette ambiguïté on remplacera chaque entier pouvan être étiquette par le symbole enquête. Au cours des vérifications locales (fonction L) par une étude du contexte dans les expressions on détermine si enquête doit devenir étiquette ou entier (cf 2.3.3.2 - 2.3.3.3).

### 2.3.2.3 Vérification sur les bornes d'un tableau

Une borne d'un tableau ne peut être exprimée en fonction de variables que si celles-ci sont déclarées dans un bloc plus grand.

Pour vérifer ces conditions on essaie dans le plus petit bloc co nant la déclaration de remplacer dans chaque borne les variables par leu type ; si un remplacement est possible il y a erreur. Sinon la borne est correcte et on supprime la racine externe afin de permettre des remplac ments dans les bloc plus grands.



Définition de la fonction T

$T(\Lambda) = \Lambda$

$T(r + s) = T(r) + T(s)$

$T(A \chi r) = A \chi\ T(r)$ sauf

A = bloc

$T(A \chi r) = \iota(T(r), T(r))$

Définition de la fonction $\iota$

$\iota(r, \Lambda) = \Lambda$

$\iota(r, s_1 + s_2) = \iota(r, s_1) + \iota(r, s_2)$

$\iota(r, A\ s) = A \chi\ \iota(r, s)$ sauf

2.3.2' A = identificateur

$\iota(r, A \chi s) =$ si $\omega(A \chi s, r) = \Lambda$ alors A s

si $|\omega(A \chi s, r)| = 1$ alors $\omega(A \chi s, r)$

sinon 0

2.3.2.1' A = entier sans signe

$\iota(r, A \chi s) =$ si $\omega(A \chi s, r) = \Lambda$ alors A s

si $|\omega(A \chi s, r)|$ = étiquette alors enquête

sinon 0

2.3.2.2' A = externe

$\iota(r, A \chi s) =$ si $\iota(r, s) = \Lambda$ alors s sinon 0

Définition de la fonction $\iota'$

$\iota'(r, \Lambda) = \Lambda$

$\iota'(r, s_1 + s_2) = \iota'(r, s_1) + \iota'(r, s_2)$

$\iota'(r, A \chi s) = \iota'(r, s)$ sauf

A = identificateur

$\iota'(r, A \chi s) =$ si $\omega(A \chi s, r) = \Lambda$ alors $\Lambda$ sinon 0.

## 2.3.3 VERIFICATION DES CONDITIONS DE CONTEXTE LOCALES

### FONCTION L

Introduction

Nous décrivons ici, une fonction formalisant les règles de contexte relatives aux instructions, aux expressions et aux primaires.

Nous aurons à évaluer le type de primaires et des opérations sur les primaires, le type ou la validité des expressions, et enfin la validité des expressions et des instructions.

Par la fonction T nous avons vérifié si les règles de contexte globales sont satisfaites ; nous n'aurons plus à nous préoccuper ni des déclarations ni de la structure de blocs en général.

2.3.3.1 Type des primaires

- a) Constantes :
  - Le type d'un entier sans signe est entier ;
  - le type d'une partie décimale ou d'un facteur de cadrage est réel ;
  - le type d'un nombre sans signe ou d'un nombre décimal est déterminé par la règle de composition définie par (cf 2.2.3.7) ;
  - le type des valeurs logiques vrai et faux est booléen ;
  - le type des chaines est chaine.
- b) Identificateurs
  - Le type d'un identificateur est celui contenu dans sa déclaration.

Au cours des vérifications globales on a remplacé les identificateurs par leur type. S'il reste maintenant des identificateurs, c'est qu'ils n'ont pas été déclarés ; il y a donc une erreur.

- c) Identificateurs indicés
  - Le type d'un identificateur indicé est celui des éléments du tableau qu'il désigne si la condition suivante est respectée :

un identificateur indicé doit posséder autant d'indices que sa déclaration



## 2.3.3' DEFINITION DE LA FONCTION L

$L(\Lambda) = \Lambda$

$L(r + s) =$ si $L(r) = 0$ ou $L(s) = 0$ alors 0 sinon $L(r) + L(s)$

$L(A \times r) = L(r)$ sauf

A = entier, réel, booléen, chaine, étiquette, enquête, vide libre, ouvert, proc, protype, rangée

$L(A \times r) = A \times L(r)$.

2.3.3.1'

- a) A = entier sans signe

  $L(A \times r) =$ entier

  A = partie décimale ou facteur de cadrage

  $L(A \times r) =$ réel

  A = nombre sans signe ou nombre décimal

  $L(A \times r) = \lambda\ (c_1\ L(r),\ c_2\ L(r))$

  A = vrai, faux

  $L(A \times r) =$ booléen

  A = chaine

  $L(A \times r) =$ chaine

- b) A = identificateur

  $L(A \times r) = 0$

- c) A = identificateur indicé

  $L(A \times r) = rg\ (c_1\ L(r),\ c_2\ L(r))$

  A = indice

  $L(A \times r) =$ si $L(r) =$ entier, ou réel alors $\Lambda$ sinon 0

possède de paires de bornes, ses indices doivent être de type entier ou réel ; cette dernière condition est vérifiée par l'étude des expressions arithmétiques, la fonction rg (cf 2. 3. 3. 6) permet de vérifier la première condition.

- d) Appel de procédure

Le type d'un appel de procédure est celui du résultat de la procédure si la condition suivante est réalisée :

un appel de procédure doit posséder autant de paramètres effectifs que sa déclaration posséde de paramètres formels ; les types des paramètres effectifs doivent être respectivement compatibles aux types des paramètres formels. La fonction $\theta$ (cf 2. 3. 3. 8) vérifie la condition de compatibilité entre les paramètres formels et les paramètres effectifs.

2. 3. 3. 2' Type des expressions simples

- a) Le type du résultat d'une opération est déterminé par la règle de composition définie par la fonction $\lambda$ (cf 2. 3. 3. 7)

De plus :

- b) Si la règle de composition appliquée à un quotient entier fournit le résultat entier alors le quotient entier est du type entier sinon c'est une erreur.

- c) Si la règle de composition appliquée à une relation booléenne fournit le résultat entier ou réel alors le type de la relation booléenne est booléen sinon c'est une erreur.

2. 3. 3. 3 Type d'un choix

Si les opérandes d'un choix sont l'un du type étiquette et l'autre du type étiquette ou enquête alors le type du choix est étiquette, sinon le type du choix est déterminé par la règle de composition.

2. 3. 3. 4 Validité des expressions

- a) Pour qu'une expression arithmétique soit valide il faut que son type soit entier ou réel ou enquête.



A = suite d'indices

$L(A \times r)$ = si $L(r) = 0$ alors 0 sinon rangée $\times L(r)$

-d) A = appel de procédure

$L(A \times r)$ = si $d_1\ c_1\ L(r)$ = libre ou $\theta(d_1\ b\ c_1\ L(r),\ d_1\ L(r)) = \Lambda$ alors $c'_1\ c_1\ L(r)$ sinon 0

2. 3. 3. 2' -a) A = facteur, terme, relation, expression arithmétique simple, secondaire booléen, facteur booléen, implication, expression simple

$L(A \times r) = \lambda\ (c_1\ L(r),\ c_2\ L(r))$

- b) A = quotient entier

$L(A \times r)$ = si $\lambda\ (c_1\ L(r),\ c_2\ L(r))$ = entier alors entier sinon 0

- c) A = relation booléenne

$L(A \times r)$ = si $\lambda\ (c_1\ L(r),\ c_2\ L(r))$ = réel ou entier ou enquête alors booléen sinon 0

2. 3. 3. 3' A = choix

$L(A \times r)$ = si $c_1\ L(r)$ = étiquette ou enquête et $c_2\ L(r)$ étiquette ou $c_1\ L(r)$ = étiquette et $c_2\ L(r)$ = étiquette ou alors étiquette sinon $\lambda\ (c_1 L(r), c_2 L(r))$ enquête

2. 3. 3. 4' -a) A = expression arithmétique

$L(A \times r)$ si $L(r)$ = entier ou réel ou enquête alors $\Lambda$ sinon 0.

- b) Pour qu'une expression booléenne soit valide, il faut que son type soit booléen.

- c) Pour qu'une expression de désignation soit valide, il faut sont type soit étiquette ou enquête.

2.3.3.5 Validité des instructions

A l'exception des instructions d'affectation et instructions procédures toutes les instructions sont exprimées en fonction : d'expression arithmétique, d'expression booléenne ou d'expression de désignation il suffit de déterminer la validité : d'une instruction d'affectation et d'une instruction procédure.

- a) Validité d'une instruction d'affectation

Une instruction d'affectation est valide si les règles suivantes sont respectées.

a) Le type d'une partie gauche est celui de sa variable. Un identificateur de procédure ne peut être en partie gauche qu'à l'intérieur de son corps de procédure.

b) Une liste de parties gauches doit être composée de parties gauches de même type ; son type est celui de ses parties gauches.

c) Une instruction d'affectation est valide si la règle de composition du type de sa liste de parties gauches et de sa partie droite, fournit l'un des trois types entier, réel, booléen

- b) Validité d'une instruction procédure

Pour qu'une instruction procédure soit valide il faut que son type, soit type vide (cas général d'une instruction procédure) ou procédure à résultat vide (cas d'une instruction procédure sans paramètre).

2.3.3.6 Vérification de la condition sur les indices d'un identificateur indicé

Nous nous proposons ici, de vérifier que le nombre d'indices d'un identificateur indicé est égal au nombre de dimensions du tableau dont il indique un représentant.



- b) A = expression booléenne

$L(A \times r)$ = si $L(r)$ = booléen alors $\Lambda$ sinon 0

- c) A = expression de désignation

$L(A \times r)$ = si $L(r)$ = étiquette ou enquête alors $\Lambda$ sinon 0

2.3.3.5'

- a) A = Instruction d'affectation

$L(A \times r)$ = si $\lambda$ ($c_1\ L(r)$, $c_2\ L(r)$) = entier, réel, booléen
alors $\Lambda$ sinon 0

A = partie gauche

$L(A \times r)$ = si $\rho\ L(r)$ = protype alors $c_1'\ L(r)$
si $\rho\ L(r)$ = proc alors 0
sinon $L(r)$

A = liste de parties gauches

$L(A \times r)$ = si $c_1\ L(r) = c_2\ L(r)$ = réel ou entier ou booléen
alors $c_1\ L(r)$ sinon 0

- b) A = instruction procédure

$L(A \times r)$ = si deproc $L(r)$ = vide alors $\Lambda$ sinon 0

2.3.3.6' Définition de la fonction rg

$rg(r, \Lambda)$ = si $\rho(r)$ = rangée alors 0 sinon r

$rg(r, s_1 + s_2) = 0$

$rg(r, a \times s)$ = si r = ouvert alors si A = rangée et s = $\Lambda$
alors b (r) sinon $rg(r, s)$

Le déclareur créé par la fonction P (cf 2.3.1.3) est comparé à une ramification analogue obtenue en "comptant" le nombre d'indices de l'identificateur indicé (cf 2.3.3.1 c). S'il y a égalité on retient le type des éléments du tableau, sinon il y a erreur.

Si la déclaration du tableau est issue d'une spécification alors on recherche simplement le type des éléments du tableau.

2.3.3.7 Règle de composition

La règle de composition doit permettre les opérations sur des opérandes de type réel, entier, booléen ou des procédures sans paramètres à résultat réel, entier ou booléen :

La composition est alors la suivante :

la composition d'un seul opérande donne cet opérande.

Deux opérandes réels donnent un résultat réel.

Deux opérandes l'un étant réel et l'autre entier ou enquête donnent le résultat réel.

si les deux opérandes sont entier ou enquête le résultat est entier.

Deux opérandes booléen donnent un résultat booléen.

Si un opérande au moins est libre le résultat est le second opérande.

Dans tous les autres cas la composition est impossible.

2.3.3.8 Règle de compatibilité entre paramètres formels et paramètres effectifs

Une suite de paramètres effectifs est compatible à une suite de paramètres formels si la compatibilité existe entre paramètres effectifs et paramètres formels de même rang.

Un paramètre effectif est compatible à un paramètre formel dans chacun des cas suivants :

a) le paramètre effectif est de même type que le paramètre formel ;

b) le paramètre formel est de type libre ;

c) le paramètre effectif est de type libre ;



Si A = rangée et $\rho$ (r) = rangée
alors rg (b (r), s)
sinon 0.

2.3.3.7' Définition de la fonction $\lambda$

$\lambda$ (r, A x s) = $\lambda'$ (deproc (r), deproc (A x s))

a) Définition de la fonction deproc

deproc ($\Lambda$) = $\Lambda$

deproc (r+s) = 0

deproc (A x r) = si $\rho$ = proc ou protype et $d_1(r) = \Lambda$ alors r
sinon A x r

b) Définition de la fonction $\lambda'$

$\lambda'$ (r, $\Lambda$) = r

$\lambda'$ (r, $s_1+s_2$) = 0

$\lambda'$ (r, Axs) = si A = entier, réel, enquête et r = réel alors réel
si A = entier, enquête et r = entier ou enquête
alors entier
si A = réel et r = entier, réel, ou enquête alors réel
si A = booléen et r = booléen alors booléen
si A = libre alors r sinon si r = libre alors A x s
sinon 0.

2.3.3.8' Définition de la fonction $\theta$

$\theta$ ($\Lambda$, s) = 0

$\theta$ ($r_1 + r_2$, s) = $\theta$ ($r_1$, $c_1$ (r)) + $\theta$ ($r_2$, $d_1$(s))

$\theta$ (A x r, s) = si $|\rho(s)| > 1$ alors 0
si A x r = s alors $\Lambda$
si A = libre alors $\Lambda$
si r = libre alors $\Lambda$
si A = réel, et s = proc x réel ou protype x réel
A = entier, et s = proc x entier ou protype x entier
A = booléen et s = proc x booléen ou protype x booléen
alors $\Lambda$.

d) le paramètre effectif est une procédure sans paramètre do
le résultat est du type du paramètre formel ;

e) le paramètre effectif est d'un type numérique ou procédur
sans paramètre à résultat numérique et le paramètre form
est d'un type numérique ;

f) le paramètre effectif est de type enquête, le paramètre for
est de type étiquette ;

g) le paramètre effectif est une procédure, le paramètre for
est une procédure à résultat de même type ;

h) le paramètre effectif est un tableau, le paramètre formel
un tableau.



si A = réel et s = entier ou proc × entier, protype × entier
ou proc × (entier + libre)
A = entier et s = réel ou proc × réel, protype × réel
ou proc × (réel + libre)
alors Λ

si A = étiquette et s = enquête alors Λ

si A = proc et $\rho$ (s) = proc ou protype alors 0 (r, $c_1$ (s))

si A = ouvert et $\rho$ (s) = rangée alors 0 (A × r, b (s))
A = ouvert et $\rho$ (s) ≠ rangée alors 0 (r, s)

sinon 0.

# CHAPITRE III

# DEFINITION DE LA SYNTAXE D'UN SOUS-ENSEMBLE D'ALGOL 68

Nous nous proposons de définir la syntaxe du sous-ensemble d'Algol 68 obtenu en enlevant les formats ainsi que les modes infinis ; après avoir donné la grammaire $G_0$ et le vocabulaire V nous donnerons le prédicat de reconnaissance F.

La grammaire $G_0$ est décrite selon la notation de Backus. Elle est déduite du rapport Algol 68 [A. VAN WIJNGAARDEN]. Cette grammaire peut paraître, par endroits, inutilement compliquée ; nous l'avons voulu ainsi afin de faciliter la description du prédicat de reconnaissance.

Le prédicat de reconnaissance est décrit au moyen de 5 fonctions récursives, qui sont, elles-mêmes, décrites au moyen de fonctions plus simples. Il semblerait que la fonction ξ (cf 3.3.3.2') ne puisse pas être décrite par des fonctions récursives primitives. Parmi les autres fonctions, certaines n'apparaissent pas, dans leur description, comme récursives primitives cependant, on peut les y ramener [cf QUERE ].

## 3.1 GRAMMAIRE $G_0$

### 3.1.1 Symboles de base

< symbole de base > ::= < lettre > |< chiffre > |true|false|1|0|.|$ |
= |.|$_{10}$| < opérateur strict > |< symbole de contation > | < symbole de déclaration > |
< marques syntaxiques > | < marques de séquence >|
< skip nil > | < symboles spéciaux> |< symboles particuliers :

< lettre > ::= A | B | C| • • • X |Y|Z

< chiffre > ::= 0|1|2| • • • |9

< opérateur strict > ::= -:=|+:=|*:=|/:=|↑:=|↓:=|+=:|
∧|∨|≠|<|≤|≥|>|/| + | ↑:|
elem| lwb | upb| lws| ups| i| not| down| up |
abs| bin| repr| leng| short| odd| sign| rownd|
re| im| conj| btb| ctb| +| -

< symbole de confrontation > ::= :=| ::| :=:| :≠:|:

< symbole de déclaration > ::= int| real| bool| format| long| struct| ref
flex| either| proc| mode| compl| bits|bytes|
string| sema| file| priority| loc| heap| op

< marques syntaxiques > ::= (|) | begin| end| , | par| '[|]|:| at | if| case| || then| in| else| out| fi| esac| of

< marques de séquence > ::= ;| . | goto

< skip nil > ::= skip | nil

< symboles particuliers > ::= for | from| by| to| while| do| thef| elsf

< symboles spéciaux > ::= "|co| < autre symbole >

< autre symbole > ::= tous les caractères que le programmeur peut souhaiter à l'exclusion de ceux déjà cités.

### 3.1.2 Programme

< programme > ::= (< prélude > ; < programme particulier > ; < sortie > : < postlude >)

< prélude > ::= < parties des déclarations >

< sortie > ::= EXIT

< postlude > ::= < train seriel > | < clause unitaire >| < étiquetage > < clause unitaire >

< programme particulier > ::= < clause parenthésée > | < étiquetage > < clause parenthésée >

< étiquetage > ::= < étiquette > : | < étiquette > : < étiquetage >

< étiquette > ::= < identificateur >

< identificateur > ::= < lettre > | < identificateur > < lettre > | < identificateur > < chiffre >

### 3.1.3 Clause parenthèsée

< clause parenthèsée > ::= < clause fermée > | < clause collatérale > | < clause conditionnelle >

< clause fermée > ::= begin < clause serielle > end | (< clause serielle >)

#### a) Clause serielle

< clause serielle > ::= < queue serielle > | < partie des déclarations > ; < queue serielle >

< queue serielle > ::= < train seriel > | < clause unitaire > | < étiquetage > < clause unitaire > | < train seriel > < compléteur > < queue serielle >



< compléteur > ::= . < étiquette > :

< point virgule > ::= ; < étiquetage >

< train seriel > ::= < clause unitaire > < point virgule > < train seriel > | < étiquetage > < clause unitaire > <point virgule > < train seriel >

< partie des déclarations > ::= < déclaration simple > | < suite de phrases> < déclaration simple >

< suite de phrases> ::= < phrase> ; | <phrase > ; < suite de phrases >

< phrase > ::= < déclaration simple > | < clause unitaire >

< déclaration simple > :: < déclaration unitaire > | < déclaration collatérale >

< déclaration collatérale > ::= < déclaration unitaire > , < déclaration unitaire > | < déclaration unitaire > , < déclaration collatérale >

#### b) Clause conditionnelle

< clause conditionnelle > ::= if < condition > then < clause serielle > else < clause serielle > fi | (< condition > | < clause serielle > |<clause serielle>)| case < condition > in < clause serielle > out < clause serielle > esac

< condition > ::= < clause sérielle >

#### c) Clause collatérale

< clause collatérale > ::= begin < liste collatérale > end | (< liste collatérale >) | par begin < liste collatérale > end | par (< liste collatérale >)

< liste collatérale > ::= < clause unitaire > , < clause unitaire > | < clause unitaire > , < liste collatérale >

### 3.1.4 Clause unitaire

< clause unitaire > ::= < confrontation > | < tertiaire >

#### a) Confrontation

< confrontation > ::= < assignation > | < relation de conformité > ) < relation d'identité > | < moule >

< assignation > ::= < tertiaire > := < clause unitaire >

< relation de conformité > ::= < tertiaire > < marque de conformité >
< tertiaire >

< marque de conformité > ::= :: | ::=

< relation d'identité > ::= < tertiaire > < marque d'identité > < tertiaire >

< marque d'identité > ::= :=: | : ≠ :

< moule > ::= < déclareur effectif > : < clause unitaire >

### 3.1.5 Tertiaires

< tertiaire > ::= < formule > |<secondaire>

a) Formule

< formule > ::= < formule dyadique > | < formule monadique >

< formule dyadique > ::=
< formule dyadique > < opérateur dyadique > < opérande effectif >
< opérande effectif > < opérateur dyadique > < opérande effectif >

< opérande effectif > ::= < formule monadique > | < secondaire >

< formule monadique > ::= < opérateur monadique > < secondaire > |
< opérateur monadique > < formule monadique >

b) Secondaires

< secondaire > ::= < générateur > | < sélection > | < primaire >

< générateur > ::= loc < déclareur effectif > |
< déclareur effectif >

< sélection > ::= < sélecteur > of < secondaire >

< sélecteur > ::= < lettre > | < sélecteur > < lettre > | < sélecteur > < chif

### 3.1.6 Primaires

< primaire > ::= < base > | < clause parenthèsée > | < saut > |
skip | nil

< saut > ::= goto < identificateur > | < identificateur >

a) Bases

< base > ::= < identificateur > | < denotation > | < tranche > )
< appel > | (< moule vide >)



b) Tranche

< tranche > ::= < primaire > [ < liste de sections > ] | < primaire > [ ]

< liste de sections > ::= < section > | < section > , < liste de sections > |
< section > , | , < liste de sections >

< section > ::= < indice > | < zone d'indice >

< indice > ::= < tertiaire >

< zone d'indice > ::= < zone simple > |
< zone simple > at < tertiaire >

< zone simple > ::= : | < tertiaire > : | : < tertiaire > |
< tertiaire > : < tertiaire >

c) Appel

< appel > ::= < primaire > < paramètres effectifs >

< paramètres effectifs > ::= ( < liste de clauses unitaires >)

< liste de clauses unitaires > ::= < clause unitaire > < sepa >
< liste de clauses unitaires > |
< clause unitaire >

< sepa > ::= , | ;

d) Moule vide

< moule vide > ::= : < clause unitaire >

### 3.1.7 Dénotation

< dénotation > ::= < dénotation numérique > | < dénotation booléenne > |
< dénotation de caractère > | < dénotation de bits > |
< dénotation de chaine > | < dénotation de procédure > |

a) Dénotation numérique

< dénotation numérique > ::= < nombre > | long < dénotation numérique >

< nombre > ::= < entier > | < réel >

< entier > ::= < chiffre > | < chiffre > < entier >

< réel > ::= < nombre décimal > | < nombre flottant >

< nombre décimal > ::= . < entier > | < entier > . < entier >

< nombre flottant > ::= < entier > < exposant > |
< nombre décimal > < exposant >

< exposant > ::= 10 < entier > | 10 < signe > < entier > |
e < entier > | e < signe > < entier >

b) Dénotation booléenne

< dénotation booléenne > ::= true | false

c) Dénotation de caractère

< dénotation de caractère > ::= " < élément de chaine > "
< élément de chaine > ::= < lettre > | < chiffre > | . | 10 | 0 | 1 |
( | ) | , | - | ⊥ | i | " " |
< autres éléments de chaine >
< autres éléments de chaine > ::= < autres symboles >

d) Dénotation de bits

< dénotation de bits > ::= < rangée de bits > |
long < dénotation de bits >
< rangée de bits > ::= < flip flop > |
< flip flop > < rangée de bits >
< flip flop > ::= 0 | 1

e) Dénotation de chaine

< dénotation de chaine > ::= " < rangée de caractères > "
< rangée de caractères > ::= < élément de chaine > < élément de chaine > |
< élément de chaine > < rangée de caractères >

f) Dénotation de procédure

< dénotation de procédure > ::= ( < liste de paramètres formels > )
< moulide >
< liste de paramètres formels > ::= < paramètre formel > < sépa >
< liste de paramètres formels > |
< paramètre formel >
< paramètre formel > ::= < déclareur formel > < identificateur >
< moulide > ::= < moule > | < moule vide >

### 3.1.8 Déclarations

< déclaration unitaire > ::= < déclaration d'identité > |
< déclaration de mode > |
< déclaration d'opération > |
< déclaration de priorité >



a) Déclaration d'identité

< déclaration d'identité > ::= < déclareur formel > < identificateur > =
< clause unitaire >

b) Déclaration de mode

< déclaration de mode > ::= mode < indication de mode > =
< déclareur effectif >

c) Déclaration d'opération

< déclaration d'opération > ::= op < empreinte > = < clause unitaire >
< empreinte > ::= < empreinte dyadique > | < empreinte monadique >
< empreinte dyadique > ::= ( < opérandes virtuels dyadiques >)
< résultat > < opérateur dyadique >
< empreinte monadique > ::= ( < opérande virtuel monadique > )
< résultat> < opérateur monadique >
< résultat > ::= < déclareur virtuel >
< opérandes virtuels dyadiques > ::= < déclareur virtuel >, < déclareur virtuel >
< opérande virtuel monadique > ::= < déclareur virtuel >
< opérateur dyadique > ::= < opérateur standard long > |
< autre opérateur dyadique >
< opérateur monadique > ::= < opérateur strict long > |
< autre opérateur monadique >
< opérateur standard long > ::= < opérateur standard > |
long < opérateur standard long >
< opérateur standard > ::= < opérateur strict > | = | ≠
< opérateur strict long > ::= < opérateur strict > |
long < opérateur strict long >
< autres opérateurs dyadiques > ::= < autre symbole >
< autres opérateurs monadiques > ::= < autre symbole >

d) Déclaration de priorité

< déclaration de priorité > ::= priority < opérateur dyadique > = < niveau >
< niveau > ::= 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

### 3.1.9 Déclareurs

< déclareur formel > ::= < déclareur formel de tableau > |
< déclareur formel de structure > |
< déclareur formel de ref > |
< déclareur d'union > |
< déclareur simple >

< déclareur effectif > ::= < déclareur effectif de tableau > |
< déclareur effectif de structure > |
< déclareur effectif de ref > |
< déclareur d'union > |
< déclareur simple >

< déclareur virtuel > ::= < déclareur virtuel de tableau > |
< déclareur virtuel de structure > |
< déclareur virtuel de ref > |
< déclareur d'union > |
< déclareur simple >

< déclareur simple > ::= < déclareur de proc > |
< déclareur standard > |

a) Déclareurs standards

< déclareur standard > ::= < mode standard > | < indication de mode >

< mode standard > ::= bool | char | < numérique >

< numérique > ::= int | real | long < numérique >

< indication de mode > ::= string | file | sema |
< indication longue > | < autre symbole >

< indication longue > ::= compl | bits | bytes |
long < indication longue >

b) Déclareurs de tableau

< déclareur formel de tableau > ::= [< volume formel >] < déclareur formel de structure >
[< volume formel >] < déclareur virtuel de ref >
[< volume formel >] < déclareur d'union >
[< volume formel >] < déclareur simple >



< déclareur effectif de tableau > ::= [< volume effectif >] < déclareur effectif de structure > |
[ < volume effectif >] < déclareur virtuel de ref > |
[< volume effectif >] < déclareur d'union> |
[< volume effectif >] < déclareur simple >

< déclareur virtuel de tableau > ::= [< volume virtuel>] < déclareur virtuel de ref > |
[< volume virtuel >] < déclareur d'union> |
[< volume effectif >] < déclareur simple >

< volume formel > ::= < déclareur de bornes formelles > , < volume formel> |
< déclareur de bornes formelles >

< volume effectif > ::= < déclareur de bornes effectives >, < volume effectif > |
< déclareur de bornes effectives >

< volume virtuel > ::= : | : , < volume virtuel >

< déclareur de bornes formelles > ::= < borne formelle > : < borne formelle > |
: < borne formelle > | < borne formelle > :

< borne formelle > ::= either | < tertiaire > either | flex | < tertiaire > flex |
< tertiaire >

< déclareur de bornes effectives > ::= < borne effective > : < borne effective >

< borne effective > ::= flex | < tertiaire > flex | < tertiaire >

c) Déclareurs de structure

< déclareur formel de structure > ::= struct (< liste de champs formels >)

< déclareur effectif de structure > ::= struct (< liste de champs effectifs >)

< déclareur virtuel de structure > ::= struct (< liste de champs virtuels >)

< liste de champs formels > ::= < champ formel > , < liste de champs formels> |
< champ formel >

< liste de champs effectifs > ::= < champ effectif > , < liste de champs effectifs> |
< champ effectif >

< liste de champs virtuels > ::= < champ virtuel > , <liste de champs virtuels> |
< champ virtuel >

< champ formel > ::= < déclareur formel de structure> < sélecteur > |
< déclareur formel de tableau > < sélecteur > |

< déclareur virtuel de ref > < sélecteur > |
< déclareur d'union > < sélecteur > |
< déclareur simple > < sélecteur >

< champ effectif > ::= < déclareur effectif de structure > < sélecteur > |
< déclareur effectif de tableau > < sélecteur > |
< déclareur virtuel de ref > < sélecteur > |
< déclareur d'union > < sélecteur > |
< déclareur simple > < sélecteur >

< champ virtuel > ::= < déclareur virtuel > < sélecteur >

d) Déclareurs de ref

< déclareur virtuel de ref > ::= ref < déclareur virtuel >

< déclareur effectif de ref > ::= ref < déclareur virtuel >

< déclareur formel de ref > ::= ref < déclareur formel de tableau > |
ref < déclareur formel de structure > |
ref < déclareur virtuel de ref > |
ref < déclareur d'union > |
ref < déclareur simple >

e) Déclareurs d'union

< déclareur d'union > ::= union (< liste de composantes >)

< liste de composantes > ::= < déclareur virtuel > , < déclareur virtuel > |
< déclareur virtuel > , < liste de composantes

f) Déclareurs de proc

< déclareur de proc > ::= < déclareur de proc sans paramètre ni résultat >
< déclareur de proc sans résultat > |
< déclareur de proc sans paramètre > |
< déclareur de proc général >

< déclareur de proc général > ::= proc (< paramètres virtuels >) < résultat virtuel

< paramètres virtuels > ::= < déclareur virtuel > < sepa > < paramètres virtuels > | < déclareur virtuel >

< résultat virtuel > ::= < déclareur virtuel >

< déclareur de proc sans paramètre ni résultat > ::= proc

< déclareur de proc sans résultat > ::= proc (< paramètres virtuels >)

< déclareur de proc sans paramètre > ::= proc < résultat virtuel >



## 3.2 VOCABULAIRE

Le vocabulaire V est la réunion du vocabulaire $V_0$ de la grammaire $G_0$ et de l'ensemble suivant :

{0, void, procl, rowof, borne de tableau, opérandes, op2, empreinte double, empreinte simple, ser, coll, row of 1 }

- 0 représente la valeur d'échec ou d'erreur ; les autres symboles que nous ajoutons ont pour but de faciliter la description du prédicat de reconnaissance F.

## 3.3 PREDICAT DE RECONNAISSANCE

### Introduction

Algol 68 offre la possibilité de traiter des objets dont la nature (le mode) n'est pas fixé de façon rigide. On peut définir : de nouveaux modes ; (généralement exprimés en fonctions des modes de base) et dans ces modes de nouvelles opérations. Afin de conserver une certaine efficacité dans le contrôle d'un programme, on a apporté des restrictions à la définition des modes, à la définition des opérations, et, surtout à leur utilisation. Ces restrictions sont exprimées par des règles de contexte.

Avec le prédicat de reconnaissance, nous vérifions si ces conditions de contexte sont satisfaites. Pour cela on opère de la façon suivante :

a) on explicite chaque déclareur de mode en fonction des modes standard et on associe à chaque opérateur la propriété de l'opération qu'il représente (fonction M) ;

b) on organise, dans les formules, l'enchainement des opérations en fonction de la priorité des opérateurs (fonction E) ;

c) on assimile chaque identificateur à son mode et chaque opérateur aux opérations qu'il représente (fonction T) ;

d) on évalue le mode des clauses et par conséquent la validité d'un programme (Fonction L).

Afin de faciliter la description de la fonction M nous l'avons fait précéder d'une fonction de "préparation de la ramification" (Fonction P).

F est donc le produit de 5 fonctions :

F = L . T. E. M. P.

3.3.1 Préparation - Fonction P

- Simplifications générales
- création de déclarations d'étiquettes
- transport du n° de région dans les déclarations de priorité
- simplification des déclareurs.

3.3.1.1 Simplifications générales

Certains symboles ne sont que délimiteurs, permettant d'analyser un programme. Dans la ramification obtenue par l'analyse ces symboles ne supportent plus aucune signification . Pour faciliter la description des fonctions M, E, T, L nous allons les supprimer (cf 3.3.1.1.').

3.3.1.2 Création de déclarations d'étiquettes

La rencontre d'une étiquette devant une instruction doit être considérée comme une déclaration d'identificateur dont le mode est skip. Nous remplacerons ces occurences d'étiquette par une déclaration (cf 3.3.1.2').

3.3.1.3 Transport du n°de région dans les déclarations de priorité

Dans une formule, la nature de l'opération désignée par un opérateur doit être définie par une déclaration d'opération ; de plus si c'est un opérateur dyadique sa priorité doit être définie par une déclaration de priorité.

Nous appelons "région" le programme ainsi que toute clause sérielle et toute dénotation de procédure ; le n° d'une région est le nombre de régions qui la contiennent. Le programme est une région de numéro 1.

Pour distinguer chacune des déclarations de priorité d'une même indication (cf 3.3.2.1 et 3.3.4.2), nous allons à chaque déclaration de priorité ajouter le numéro de la plus petite région qui la contient.

3.3.1.4 Simplification des déclareurs

L'analyse syntaxique fournit pour les déclareurs des pseudo-arborescences fort compliquées. Chaque déclareur pouvant être composé d'autres déclareurs, on obtient une pseudo-arborescence longue où pratiquement



3.3.1' Définition de la fonction P

A = programme

$P(A \times r) = A \times P'(r, \Lambda)$

Définition de la fonction P'

$P'(\Lambda, s) = \Lambda$

$P'(r_1 + r_2, s) = P'(r_1, s) + P'(r_2, s)$

$P'(A \times r, s) = A \times P'(r, s)$ sauf

3.3.1.1' A = :=, ::, ::=, :=:, :≠:, :, (, ), begin, end, , , par, [, ], at, if, case, |, then, in, else, out, fi, esac, of, ;, ., goto, for, from, by, to, while, do, thef, elsf, ", co, either, flex

$P'(A \times r, s) = \Lambda$

3.3.1.2' A = étiquette, sortie

$P'(A \times r, s) =$ déclaration $\times$ (skip + r)

3.3.1.3' A = déclaration de priorité

$P'(A \times r, s) = A \times (c_2(r) + c'_4(r) + s)$

A = clause serielle, dénotation de procédure

$P'(A \times r, s) = A \times P'(r, s+1)$

3.3.1.4'

seules les feuilles nous intéressent, avec toutefois des renseignements sur la manière de les grouper. Nous allons donc supprimer tous les noeuds ayant un nom de déclareur et utiliser les symboles terminaux struct, union, proc, ref, long pour indiquer comment se groupent les autres symboles.

a) Pour permettre plus tard (cf 3.3.5.6) la vérification de la syntaxe des bornes de tableau nous extrayons (cf 3.3.1.5), des déclareurs formels et effectifs, les bornes que l'on y peut trouver, ainsi que les bornes contenues dans d'autres bornes (cas particulier des bornes qui sont, ou contiennent, des clauses serielles).

b) Nous considérons qu'un tableau de réels à une entrée est du mode row of × real ; qu'un tableau d'entiers à deux entrées est du mode row of × row of × int ... Pour effectuer ces modifications, on compte dans chaque volume (formel, effectif, virtuel) le nombre de dimensions ; on obtient ainsi un certain nombre de symboles row of juxtaposés ; la fonction tab (cf 3.3.1.6) enracine les uns aux autres chacun des symboles row of en plaçant "en feuille", le mode des éléments du tableau.

c) Un déclareur de structure (formel, effectif ou virtuel) doit satisfaire la règle suivante :

un déclareur de structure ne peut contenir deux sélecteurs identiques (cf [V.W] 4.4.3.e).

Nous vérifions si cette condition est réalisée (fonction sel) ; auquel cas on transforme le déclareur, en enracinant ses composantes au symbole struct.

d) On transforme chaque déclareur de ref (formel, effectif ou virtuel) en enracinant ses composantes au symbole ref.



a)

A = déclareur formel, déclareur effectif

$P'(A \times r, s) = P'(r, s) + P'(\text{ext}(r), s)$

A = déclareur virtuel, déclareur simple, déclareur standard, mode standard

$P'(A \times r, s) = P'(r, s)$

b)

A = déclareur formel de tableau
déclareur effectif de tableau
déclareur virtuel de tableau

$P'(A \times r, s) = \text{tab}(P'(r, s))$

A = volume formel, volume effectif, volume virtuel

$P'(A \times r, s) = \underline{\text{row of}} + P'(r, s)$

A = déclareur de bornes formelles
déclareur de bornes effectives

$P'(A \times r, s) = \Lambda$

c)

A = déclareur formel de structure;
déclareur effectif de structure,
déclareur virtuel de structure.

$P'(A \times r, s) = \text{si sel}(d_1\ P'(r, s)) = \Lambda$
alors $\underline{\text{struct}} \times d_1\ P'(r, s)$ sinon 0

A = liste de champs formels
liste de champs effectifs
liste de champs virtuels

$P'(A \times r, s) = P'(r, s)$

d)

A = déclareur virtuel de ref,
déclareur effectif de ref,
déclareur formel de ref,

$P'(A \times r, s) = \underline{\text{ref}} \times P'(r, s)$

e) Un déclareur d'union désigne la réunion logique de plusieurs modes, on enracine chacun de ces modes par le symbole union. On supprime éventuellement le symbole union consécutif en ne gardant que le premier (fonction un).

f) Pour permettre les vérifications de contexte sur les procédures (cf 3.3.5.4), nous allons distinguer les procédures avec paramètres des procédures sans paramètre.

Les déclareurs de procédure sans paramètre sont placés sous la dépendance du symbole proc alors que les déclareurs de procédure avec paramètres, sont placés sous la dépendance du symbole proc 1. De plus on attribue à tout déclareur de procédure sans résultat le "mode résultat" void .

3.3.1.5 Fonction ext

La fonction ext recherche les bornes formelles ou effectives contenues dans les déclareurs. Si ces bornes sont exprimées par des tertiaires ceux-ci sont placés sous la dépendance du symbole supplémentaire borne de tableau.

3.3.1.6 Fonction tab

La fonction tab transforme toute ramification dont la première composante est row of, en une pseudo arborescence de racine row of

3.3.1.7 Fonction sel

La fonction sel vérifie si un déclareur de structure ne contient pas deux sélecteurs identiques.



e) A = déclareur d'union

$P'(A \times r, s) = \underline{union} \times un\ (d_1\ P'(r, s))$

A = liste de composantes

$P'(A \times r, s) = P'(r, s)$

f) A = déclareur de proc général

$P'(A \times r, s) = \underline{proc\ 1} \times d_1\ P'(r, s)$

A = déclareur de proc sans résultat

$P'(A \times r, s) = \underline{proc\ 1} \times (d_1\ P'(r, s) + \underline{void})$

A = déclareur de proc sans paramètre

$P'(A \times r, s) = \underline{proc} \times d_1\ P'(r, s)$

A = déclareur de proc sans paramètre ni résultat

$P'(A \times r, s) = \underline{proc} \times \underline{void}$

A = déclareur de proc, paramètres virtuels, résultat virtuel, sepa

$P'(A \times r, s) = P'(r, s)$

3.3.1.5' $ext(\Lambda) = \Lambda$

$ext(r + s) = ext(r) + ext(s)$

$ext(A \times r) = ext(r)$ sauf

A = borne effective, borne formelle

$ext(A \times r) =$ si $r = \Lambda$ alors $\Lambda$

sinon $\underline{borne\ de\ tableau} \times r$

3.3.1.6' $tab(\Lambda) = \Lambda$

$tab(r+s) =$ si $r = \underline{row\ of}$ alors $\underline{row\ of} \times tab(s)$

sinon $tab(r) + tab(s)$

$tab(A \times r) = A \times r$

3.3.1.7' $sel(\Lambda) = \Lambda$

$sel(s_1 + s_2) =$ si $\rho(s_1) \neq$ selecteur alors $sel(s_2)$

sinon si $n(s_1, s_2) = \Lambda$ alors $sel(s_2)$

sinon 0

$sel(A \times s) = \Lambda$

3.3.1.8 Fonction un :

Lorsqu'un mode uni $\mu$ a pour composante un autre mode uni $\mu'$, $\mu$ doit être directement exprimé en fonction des composantes de $\mu'$ ; la fonction un permet d'effectuer cette décomposition.

un ($\Lambda$) = $\Lambda$
un (r+s) = un (r) + un (s)
un (A $\times$ r) = A $\times$ r sauf
A = union
un (union $\times$ r) = un (r)

### 3.3.2 Transport des déclarations de priorité et des déclarations de mode - Fonction M

Dans toute la suite nous appellerons "épiregion" une région à l'exclusion de toutes les régions qu'elle contient.

Dans chaque épirégion nous avons à vérifier certaines conditions de contexte relatives aux opérateurs et aux modes.

a) La priorité d'un opérateur dyadique ne peut être déclarée plus d'une fois par épirégion (cf [V. W] 4. 2. 2 b)

Cette condition est vérifiée en 3. 3. 2. 1 (cf fonction $\mu$).

b) Dans une épirégion il ne peut exiter deux déclarations d'opérations relatives à un même opérateur, telles que leurs empreintes soient légèrement liées (cf [V. W] 4. 4. 2. c). La vérification de cette condition de contexte est effectuée en même temps que celle sur les identificateurs (cf 3. 3. 4. 1 et 3. 3. 4. 2).

c) Un mode ne peut être déclaré plus d'une fois par épirégion (cf [V. W] 4. 2. 2 b).

Cette condition est vérifiée en 3. 3. 2. 2 (cf fonction $\mu$).

d) Un déclareur d'union ne peut contenir deux modes tels que l'on puisse trouver un troisième qui leur soit fermement compatible (cf [V. W] 4. 4. 3 d).

La vérification de cette condition est effectuée par la fonction L (vérification des conditions de contexte locales), (cf 3. 3. 5. 10).

e) Un déclareur de structure ne peut contenir deux sélecteurs identiques (cf [V. W] 4. 4. 3 c).

La vérification de cette condition a été effectuée par la fonction F (cf 3. 3. 1. 4. c).

#### 3. 3. 2. 1 Transport des déclarations de priorité

Un opérateur dyadique représente une opération dont la priorité est définie par une déclaration de priorité. A chaque opérateur dyadique on associe la priorité de l'opération qu'il représente ainsi que le numéro de la région contenant la déclaration de priorité. On les place sous la dépendance du symbole op 2. Si un opérateur ne possède pas de déclaration de priorité



### 3.3.2' Définition de la fonction M

$M(\Lambda) = \Lambda$

$M(r+s) = M(r) + M(s)$

$M(A \times r) = A \times M(r)$ sauf

A = programme, clause sérielle, dénotation de procédure

$M(A \times r) = A \times \mu(M(r), M(r))$

#### Définition de la fonction $\mu$

$\mu(r, \Lambda) = \Lambda$

$\mu(r, s_1 + s_2) = \mu(r, s_1) + \mu(r, s_2)$

$\mu(r, A \times s) = A \times \mu(r, s)$ sauf

#### 3.3.2.1' A = opérateur dyadique

$\mu(r, A \times s) =$ si $|\zeta(r, A \times s)| = 1$ alors op 2 $\times (s + \zeta(r, A \times s))$

si $\zeta(r, A \times s) = \Lambda$ alors $A \times s$

sinon 0

on le laisse inchangé, en vue de lui affecter une priorité dans une région plus grande. S'il possède plusieurs déclarations de priorité, c'est une erreur (cf fonction $\mu$ 3.3.2.1 a).

3.3.2.2 Transport des déclarations de mode

Une indication de mode représente un mode dont le déclareur figure dans une déclaration de mode. Le but de cette opération est de remplacer partout, même dans les déclareurs, chaque indication de mode par son déclareur. Un déclareur n'est pas toujours exprimé en fonction des modes standards, donc avant de remplacer les indications de modes par leur déclareur, il faut expliciter ces derniers en fonction des modes standards ; pour cela nous avons recours à la fonction $\xi$ (cf 3.3.2.4).

3.3.2.3 Recherche des déclarations de priorité et des déclarations de mode-fonction $\zeta$

La fonction $\zeta$ permet de retrouver dans une région les déclarations de priorité et les déclarations de mode. Elle analyse chacune d'elle afin de reconnaître l'opérateur ou l'indication de mode pour lequel la recherche est effectuée. Si l'opérateur est reconnu, elle sélectionne le niveau de priorité avec le n° de région, si c'est une indication de mode, elle sélectionne le déclareur effectif. Il se peut que des opérateurs ou des indications de mode figurent dans plusieurs déclarations de priorité ou de mode, dans ce cas la fonction $\zeta$ fournit autant de résultats.

3.3.2.4 Vérification sur les modes finis - Fonction $\xi$

Si un déclareur effectif s'exprime en fonction d'indications de mode, pour vérifier que ce déclareur engendre un mode fini, il est nécessaire de remplacer dans ce déclareur chacune des indications de mode par leur déclareur. Ce processus peut se poursuivre indéfiniment si le déclareur engendre un mode infini ; pour remédier à cette difficulté on procède de la façon suivante :

soit $d_0$ le déclareur initial et $m_0$ l'indication de mode correspondante, soit $m_i$ une indication et $d_i$ son déclareur.

Pour chaque indication de mode $m_{i+1}$ rencontrée dans un déclareur $d_i$, on établit une liste des indications déjà rencontrées.



3.3.2.2' A = indication de mode

$\mu (r, A \times s) =$ si $\zeta (r, A \times s) = \Lambda$ alors $A \times s$
si $|\zeta (r, A \times s)| = 1$ alors $\xi (r, \zeta (r, A \times s), s)$
sinon 0

3.3.2.3' Définition de la fonction $\zeta$

$\zeta (\Lambda, s) = \Lambda$
$\zeta (r_1 + r_2, s) = \zeta (r_1, s) + \zeta (r_2, s)$
$\zeta (A \times r, s) = \zeta (r, s)$ sauf
A = déclaration de priorité
$\zeta (A \times r, s) =$ si $c_1 (r) = s$ alors $d_1 (r)$ sinon $\Lambda$
A = déclaration de mode
$\zeta (A \times r, s) =$ si $c_1 (r) = s$ alors $c_2 (r)$ sinon $\Lambda$

3.3.2.4' Définition de la fonction $\xi$

$\xi (r, \Lambda, t) = \Lambda$
$\xi (r, s_1 + s_2, t) = \xi (r, s_1, t) + \xi (r, s_2, t)$
$\xi (r, A \times s, t) = A \times \xi (r, s, t)$ sauf
A = indication de mode
$\xi (r, A \times s, t) =$ si $n (s, t) \neq \Lambda$ alors 0
si $\zeta (r, A \times s) = \Lambda$ alors $A \times s$
si $|\zeta (r, A \times s)| = 1$ alors $\xi (r, \zeta (r, A \times s), s+t)$
sinon 0

Soit $\mathfrak{L}(d_0)$ la liste des indications de mode rencontrées pour le mode $m_0$ ; par convention $\mathfrak{L}(d_0) = \{m_0\}$.
Soit $\mathfrak{L}(d_i)$ la liste des indications de mode rencontrées pour le mode $m_i$, alors $\mathfrak{L}(d_{i+1}) = \mathfrak{L}(d_i) \cup \{m_{i+1}\}$.

Si $\mathfrak{L}(d_i)$ contient déjà l'indication $m_{i+1}$, nous sommes en présence d'un mode infini c'est donc une erreur.

Il se peut que dans la région en cours on ne trouve pas le déclareur $d_i$ du mode $m_i$, dans ce cas l'opération de remplacement est abandonnée à cet endroit, en vue de permettre un remplacement dans une région plus grande.

### 3.3.3 Enchainement des opérations dans une formule - Fonction E

L'analyse syntaxique d'un programme Algol 68 ne tient pas compte de la priorité des opérations. Nous allons, dans les formules, organiser l'enchainement des opérations en fonction des priorités des opérateurs. De plus pour faciliter les vérifications de contexte sur les modes des opérandes effectifs (cf 3.3.5.3 a) nous convertirons les opérations dyadiques de la notation usuelle en notation préfixée.

Considérons la pseudo-arborescence d'une formule

$r = A \times (r_1 + r_2 + r_3)$

où A = formule dyadique

$r_1$ est la pseudo-arborescence de l'opérande gauche

$r_2$ est la pseudo-arborescence de l'opérateur (op 2)

$r_3$ est la pseudo-arborescence de l'opérande droit.

D'après la grammaire $G_0$, $r_1$ peut être une formule ; considérons ce cas particulier.

$r_1 = A \times (r_{11} + r_{12} + r_{13})$ avec $\rho(r_{11})$ = opérande effectif

$r_2$ et $r_{12}$ sont deux opérateurs désignant deux opérations de priorité $pr_2$ et $pr_{12}$. Si $pr_2 \leq pr_{12}$ alors l'opération définie par $r_{12}$ doit être considérée la première ; on obtient le parenthésage suivant :

$((\varphi(r_{11})\ \varphi(r_{12})\ \varphi(r_{13}))\ \varphi(r_2)\ \varphi(r_3))$ où $\varphi$ représente le mot des feuilles (cf 1.2.2). Si $pr_{12} < pr_2$, c'est l'opération définie par $r_2$ qui doit être considérée la première ; on obtient le parenthèsage suivant :

$(\varphi(r_{11})\ \varphi(r_{12})\ (\varphi(r_{13})\ \varphi(r_2)\ \varphi(r_3)))$.

#### 3.3.3.1 Comparaison de la priorité des opérateurs

La fonction E permet de comparer la priorité des opérateurs ; si l'ordre des opérations n'est pas à inverser elle effectue simplement la transformation de notation, sinon par l'intermédiaire de la fonction $\mathcal{E}$, elle inverse les opérations, en effectuant la transformation de notation.

#### 3.3.3.2 Inversion des opérations - fonction $\mathcal{E}$

Possédant un opérateur $\delta$ et son opérande droit, la fonction $\mathcal{E}$ recherche, dans une formule dans laquelle l'enchainement des opérations est effectué, l'opérande gauche pour l'opérateur $\delta$. Elle le trouve lors de



#### 3.3.3'' Définition de la fonction E

$E(\Lambda) = \Lambda$

$E(r + s) = E(r) + E(s)$

$E(A \times r) = A \times E(r)$ sauf

#### 3.3.3.1' A = formule dyadique

$$E(A \times r) = \begin{cases} \text{si } c'_2\ c_2\ E(r) = \Lambda \text{ alors } 0 \text{ sinon} \\ \text{si } \rho\ c_1(r) = \text{formule dyadique} \\ \underline{\text{et}}\ c'_2\ c'_1\ c_1\ E(r) < c'_2\ c_2\ E(r) \\ \text{alors } \mathcal{E}(c_1\ E(r),\ d_1\ E(r)) \\ \text{sinon } A \times (c_2\ E(r) + \underline{\text{opérandes}} \times (c_1\ E(r) + c_3\ E(r))) \end{cases}$$

#### 3.3.3.2' Définition de la fonction $\mathcal{E}$

$\mathcal{E}(\Lambda, s) = 0$

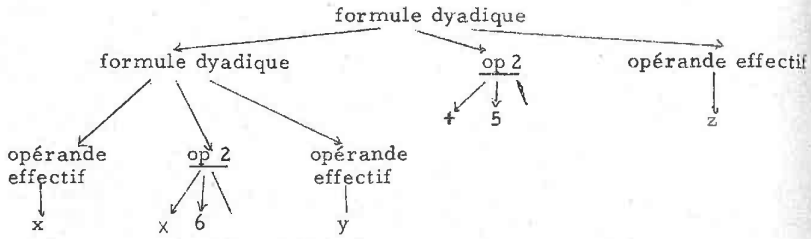
$\mathcal{E}(r_1 + r_2, s) = 0$

$\mathcal{E}(A \times r, s) = A \times \mathcal{E}(r, s)$ sauf

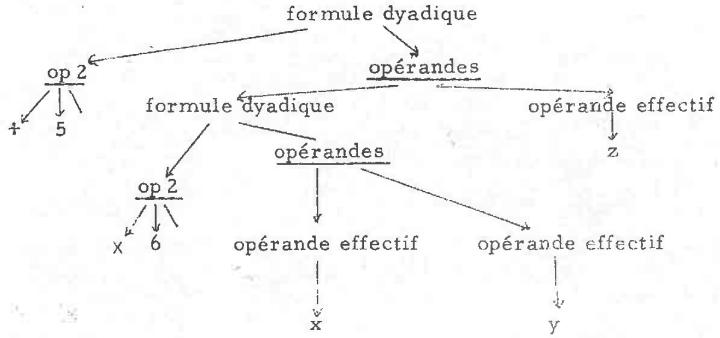
sa rencontre avec un opérateur $\delta'$ de priorité supérieure ou égale à celle de $\delta$ ; cet opérande gauche est le résultat de l'opération définie par $\delta'$ ; s'il n'y a pas d'opérateur de priorité supérieure ou égale l'opérande gauche recherché est alors le dernier opérande effectif trouvé.

Exemple 1 Considérons la formule suivante : x × y + z avec priorité de + = 5 et priorité de × = 6.

La transformation P fournit la pseudo-arborescence suivante :




Par suite de la priorité naturelle de gauche à droite, dans cette formule l'enchainement des opérations existe, on obtiendra donc par E :






A = formule dyadique

$\mathcal{E}(A \times r, s)$ = si $c'_2\ c_1\ (s) \leq c'_2\ c_1\ (r)$
alors $A \times (c_1\ (s) +$ opérandes $\times\ (A \times r + c_2\ (s))$
sinon $A \times (c_1\ (r) + \mathcal{E}(c_2\ (r), r))$

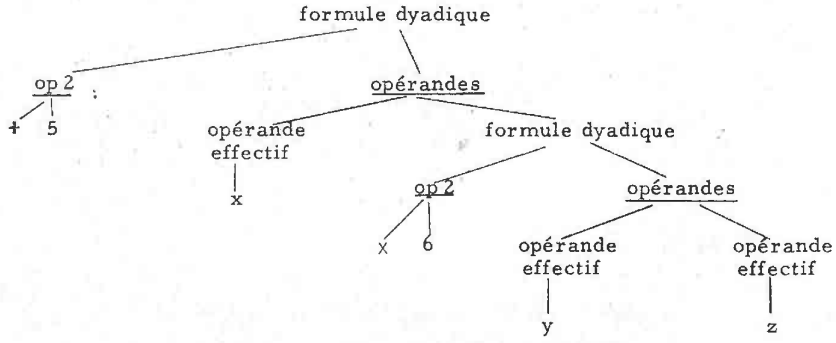
A = opérandes

$\mathcal{E}(A \times r, s) = A \times (c_1\ (r) + \mathcal{E}\ (c_2\ (r), s))$

A = opérande effectif

$\mathcal{E}(A \times r, s)$ = formule dyadique $\times\ (c_1\ (s) +$ opérandes $\times\ (A \times r + c_2(s))$

Exemple 2 Avec la formule suivante x + y × z, par suite d'une modification de l'enchainement des opérations on aurait obtenu :




Note : Ces pseudo-arborescences ont été décrites avec les seuls nœuds intéressants.

### 3.3.4 Transport des déclarations d'identité et des déclarations d'opération - Fonction T

Nous avons à vérifier certaines conditions de contexte sur les identificateurs et les opérations.

a) Un identificateur ne peut être déclaré plus d'une fois par épirégion (cf [ V. W ] 4.2.2. b).

Cette condition est vérifiée en 3.3.4.1.

b) Dans une épirégion, il ne peut exister deux déclarations d'opérations relatives à un même opérateur telles que leurs empreintes soient légèrement liées (cf [ V. W. ] 4.4.2. c).

Cette condition est vérifiée en 3.3.4.2.

#### 3.3.4.1 Transport des déclarations d'identité

Pour chaque identificateur, y compris ceux contenus dans les déclarations, on recherche la ou les déclarations qui lui correspondent (cf fonction $\theta$ 3.3.4.1'). S'il y en a plusieurs c'est une erreur, sinon on remplace l'identificateur par le mode trouve. Si l'identificateur n'a pas été déclaré dans la région, il reste inchangé en vue d'une vérification dans une région plus grande.

#### 3.3.4.2 Transport des déclarations d'opérations dyadiques

Un opérateur dyadique désigne toutes les déclarations d'opération dont la priorité a été déclarée par la même déclaration de priorité.

Nous avons associé précédemment à chaque opérateur dyadique (op 2) sa priorité et le numéro de la région contenant la déclaration de priorité. Pour chaque opérateur dyadique (op 2) on va rechercher les déclarations d'opérations (cf 3.3.4.2' fonction $\theta'$). S'il en existe et si elles vérifient la condition (3.3.4.b) alors on associe à l'opérateur correspondant la liste des empreintes trouvées dans les déclarations, chacune d'elle est placée sous la dépendance du symbole empreinte double.

#### 3.3.4.3 Transport des déclarations d'opérations monadiques

Pour chaque opérateur monadique on recherche les déclarations d'opérations (cf 3.3.4.3' fonction $\theta''$). S'il en existe et si elles vérifient la condition (3.3.4.b), on associe à l'opérateur correspondant la liste des



### 3.3.4' Définition de la fonction T

$T(\Lambda) = \Lambda$

$T(r+s) = T(r) + T(s)$

$T(A \times r) = A \times T(r)$ sauf

A = programme, clause sérielle, dénotation de procédure.

$T(A \times r) = \tau(T(r), T(r))$

Définition de la fonction $\tau$

$\tau(r, \Lambda) = \Lambda$

$\tau(r, s_1 + s_2) = \tau(r, s_1) + \tau(r, s_2)$

$\tau(r, A \times s) = A \times \tau(r, s)$ sauf

#### 3.3.4.1' A = identificateur

$\tau(r, A \times s) =$ si $\theta(r, A \times s) = \Lambda$ alors $A \times s$

si $|\theta(r, A \times s)| = 1$ alors $\theta(r, A \times s)$

sinon 0

#### 3.3.4.2' A = op 2 (1)

$\tau(r, A \times s) = A \times (s + k(\theta'(r, c_1(s) + c_2(s) + c_3(s))))$

Note 1 La vérification sur les opérateurs dyadiques doit porter
- sur le symbole de représentation $c_1(s)$
- sur la priorité de l'opérateur $c_2(s)$
- sur le numéro de la région $c_3(s)$.

#### 3.3.4.3' A = opérateur monadique

$\tau(r, A \times s) = A \times (s + k(\theta''(r, c_1(s))))$

empreintes trouvées dans les déclarations. Chacune d'elle est placée sous la dépendance du symbole empreinte simple.

3.3.4.4 Vérification de la condition des déclarations d'opérations

Deux déclarations d'opérations ont leurs empreintes légèrement liées, si leurs opérandes virtuels (dyadiques ou monadiques) sont légèrement liées.

Deux opérandes virtuels dyadiques sont légèrement liés, si leurs deux opérandes gauches sont de modes légèrement liés, et si leurs deux opérandes droits sont légèrement liés.

Deux opérandes virtuels monadiques sont légèrement liés, si leurs modes sont légèrement liés.

Deux modes sont légèrement liés :

- s'ils sont liés (cf 3.3.5.11)
- si par transformation ferme (cf 3.3.6.1) ils peuvent devenir rangée de modes légèrements liées.



Définition de la fonction $\theta$

$\theta(\Lambda, s) = \Lambda$

$\theta(r_1 + r_2, s) = \theta(r_1, s) + \theta(r_2, s)$

$\theta(A \times r, s) = \theta(r, s)$ sauf

A = déclaration d'identité, paramètre formel, déclaration

$\theta(A \times r, s) =$ si $c_2(r) = s$ alors $c_1(r)$ sinon $\Lambda$

Définition de la fonction $\theta'$

$\theta'(\Lambda, s) = \Lambda$

$\theta'(r_1 + r_2, s) = \theta'(r_1, s)$

$\theta'(A \times r, s) = \theta'(r, s)$ sauf

A = empreinte dyadique

$\theta'(A \times r, s) =$ si $d_2(r) = s$ alors empreinte double $\times (c_1(r) + c_2(r))$ sinon $\Lambda$

Définition de la fonction $\theta''$

$\theta''(\Lambda, s) = \Lambda$

$\theta''(r_1 + r_2, q) = \theta''(r_1, s) + \theta''(r_2, s)$

$\theta''(A \times r, s) = \theta''(r, s)$ sauf

A = empreinte monadique

$\theta''(A \times r, s) =$ si $c_3(r) = s$ alors empreinte simple $\times (c_1(r) + c_2(2))$ sinon $\Lambda$

3.3.4.4' Définition de la fonction k

$k(\Lambda) = \Lambda$

$k(r + s) = k(s)$ et $n\ell\ell(r, s)$

$k(A \times r) = \Lambda$

Modes non légèrement liés - Fonction $n\ell\ell$

$n\ell\ell\ (r, \Lambda) = \Lambda$

$n\ell\ell\ (r, s_1 + s_2) = n\ell\ell\ (r, s_1) + n\ell\ell\ (r, s_2)$

$n\ell\ell\ (r, A \times s)$ = non lié $(r, A \times s)$ sauf

A = empreinte simple

$n\ell\ell\ (r, A \times s) = n\ell\ell\ (c_1\ b\ (r), c_1\ (s))$

A = empreinte double

$n\ell\ell\ (r, A \times s) = n\ell\ell\ (c_1\ b(r), c_1\ (s))$ ou $n\ell\ell\ (c_2\ b\ (r), c_2\ (s))$

A = ref ou proc

$n\ell\ell\ (r, A \times s)$ = non lié $(r, A \times s)$ et $n\ell\ell\ (h\ (r), h\ (s))$

A = row of

$n\ell\ell\ (r, A \times s)$ = si $\rho\ (s)$ = row of et $\rho\ (r)$ = row of

alors $n\ell\ell\ (b\ (r), s)$

si $\rho\ (r)$ = row of alors non lié $(b\ (r), s)$

Définition de la fonction h

$h\ (\Lambda) = \Lambda$

$h\ (r+s) = \Lambda$

$h\ (A \times r) = \Lambda$ sauf

A = ref, proc

$h\ (A \times r)$ = si $\rho\ (r)$ = row of

alors r sinon $h\ (r)$.

### 3.3.5 Vérifications des conditions de contexte locales

Un programme est composé de phrases (les déclarations et les clauses). Pour qu'un programme soit "propre" il faut que ses composa respectent les règles de contexte globales et locales. A l'exception de trois règles suivantes, toutes les règles globales ont été vérifiées :

Tout identificateur, toute indication de mode, toute opération, toute priorité doivent être déclarés (cf [V.W] 4.4.1.b). Il reste à vé fier ces trois règles ainsi que les règles de contexte locales. Pratique ment nous avons à vérifier que les composantes de chaque phrase sont modes compatibles et que de ces modes on peut déduire le mode de la phrase. Les déclarations et certaines clauses constituent un tout en elles mêmes et leur mode n'intervient pas dans la constitution des modes de ses plus grandes, nous vérifierons dans ces cas que les règles de cont sont satisfaites et si elles le sont le résultat de la vérification sera sim plement $\Lambda$ , nous dirons par extension que le mode est $\Lambda$.

Ainsi :

- un programme est propre si le résultat des vérifications sur chacune ses composantes est $\Lambda$.
- Une condition est correcte si son mode (celui de sa clause parenthèse est compatible par contrainte avec le mode bool.

Il existe quatre positions de contrainte : Forte, ferme, faible et molle (cf 3.3.5.). A chacune d'elle correspond une transformation du mode a priori. Une transformation est une succession de modifications élémen res. Une programme particulier est en position forte.

La position de contrainte d'une clause $\mathcal{C}$ n'est pas toujours co a priori.

Elle peut :

a) Ne pas exister, nous dirons alors que la contrainte est nulle et compatibilité se traduit nécessairement par l'égalité.



### 3.3.5' Définition de la fonction L

$L(\Lambda) = \Lambda$

$L(r + s) = L(r) + L(s)$

$\forall A,\ L(A \times r) = 0$ si n $(0,\ L(r)) \neq \Lambda$

sinon :

$L(A \times r) = L(r)$ sauf

A = postlude, programme particulier

$L(A \times r) = \text{STRONG}(L(r),\ \underline{\text{void}})$

b) Dépendre d'une clause $\mathcal{C}'$ dont $\mathcal{C}$ est une composante. La contrainte est dite "descendante" et sa position est celle de $\mathcal{C}'$, dans ce cas particulier le mode a priori sera placé sous la dépendance d'un symbole particulier (ser ou coll) et on dit qu'on a obtenu un mode généralisé.

3.3.5.1 Modes des clauses

a) La position de contrainte d'une clause sérielle ou d'une clause conditionnelle n'est pas connue à l'avance on place les modes de leurs composantes sous la dépendance du symbole ser.

b) Le mode d'un train sériel est celui de sa dernière composante si le mode de sa première composante est fortement compatible au mode void sinon c'est une erreur. (cf 3.3.6.2)

c) Si le mode de la composante d'une condition est fortement compatible au mode void, la condition est correcte ; sinon c'est une erreur.

d) La position de contrainte d'une clause collatérale n'est pas connue à l'avance, on place le mode de ses composantes sous la dépendance du symbole coll.

3.3.5.2 Mode des clauses unitaires

a) Le mode d'une assignation est celui de sa partie gauche lorsque les conditions suivantes sont réalisées :

- la transformation par soft (cf 3.3.6.1) du mode de sa partie gauche est possible et produit le mode $\mu$ ;
- le mode de sa partie droite est fortement compatible à $\mu$ (cf 3.3.6.2) sinon c'est une erreur.

b) Le mode d'une relation de conformité est bool si la condition suivante est réalisée :

- la transformation par soft (cf 3.3.6.1) du mode de sa partie gauche est possible.

c) Le mode d'une relation d'identité est bool si l'une des conditions suivantes est réalisée :

- la transformation par soft (cf 3.3.6.1) du mode de sa partie gauche est possible et le mode de sa partie droite est fortement compatible (cf 3.3.6.2) au mode de sa partie gauche.



3.3.5.1' a) A = clause sérielle, clause conditionnelle

$L(A \times r) = \underline{ser} \times L(r)$

b) A = train sériel

$L(A \times r) =$ si STRONG ($c_1$ L(r), void) alors $d_1$ V (r) sinon (

c) A = condition

$L(A \times r) =$ STRONG (L (r), bool)

d) A = clause collatérale

$L(A \times r) = \underline{coll} \times L(r)$

3.3.5.2' a) A = assignation

$L(A \times r) =$ si STRONG ($c_2$ L(r), b soft $c_1$ L (r)) alors soft ($c_1$ L (r))
sinon 0

b) A = relation de conformité

$L(A \times r) =$ si soft ($c_1$ L (r)) alors bool sinon 0

c) A = relation d'identité

$L(A \times r) =$ si STRONG ($c_2$ L (r), soft $c_1$ L (r)) ou
STRONG ($c_1$ L (r), soft $c_2$ L (r)) alors bool
sinon 0

- la transformation par soft du mode de sa partie droite est possible et le mode de sa partie gauche est fortement compatible au mode de sa partie droite.

d) Le mode d'un moule est celui de sa partie gauche si la condition suivante est réalisée :

- le mode de sa partie droite est fortement compatible (cf 3.3.6.2) au mode de sa partie gauche.

3.3.5.3 Mode des tertiaires

a) Le mode d'une formule est celui du résultat de l'opération qu'elle désigne. Un opérateur peut désigner plusieurs opérations (cf 3.3.4.2). Pour déterminer l'opération désignée il faut comparer le (les) mode (s) de l' (des) opérande (s) effectif (s) : $c_{-1}$ (L (r)) au (x) mode (s) de l' (des) opérande (s) virtuel (s) des opérations possibles : $d_{-1}$ L (r). La fonction res (cf 3.3.5.7) détermine le mode du résultat.

b) Le mode d'une sélection est le mode du champ désigné par le sélecteur (ou le mode référence à ce mode) si les conditions suivantes sont réalisées (cf 3.3.6.1) :

- la transformation par weak (cf 3.3.6.1) sur le mode du secondaire de la sélection est possible ;

- le sélecteur désigne un champ du secondaire.

La fonction select (cf 3.3.5.8) détermine le mode d'une sélection.

c) Le mode d'un générateur est le mode ref $\mu$, où $\mu$ est le mode de son déclareur effectif.

3.3.5.4 Mode des primaires

a) Soit t une tranche et T l'identificateur de tableau désigné par t ; le mode de t est celui d'un sous-tableau de T (ou le mode référence à ce mode), si les conditions suivantes sont réalisées :

- les indices et zones d'indices sont d'un mode fortement compatible au mode int (cf 3.3.6.2) ;

- la somme des indices et zones d'indices ne dépasse pas le nombre de dimensions du tableau ;

- la transformation par weak (cf 3.3.6.1) sur le mode T est possible



d) A = moule

L (A x r) = si STRONG ($c_2$ L(r), $c_1$ L(r)) alors $c_1$ L (r)
sinon 0

3.3.5.3' a) A = formule

L (A x r) = res ($c_{-1}$ L (r), $d_{-1}$ L (r))

A = opérande

L (A x r) = A x L (r)

b) A = sélection

L (A x r) = select ($c_1$ L (r), weak $d_1$ L (r))

A = sélecteur

L (A x r) = A x r

A = champ formel, champ effectif, champ virtuel

L (A x r) = A x L (r)

c) A = générateur

L (A x r) = ref x L (r)

3.3.5.4' a) A = tranche

L (A x r) = rg ($d_1$ L (r), weak $c_1$ L (r))

A = liste de sections

L (A x r) = si $c_1$ L (r) = row of 1 alors row of 1 x $d_1$ L (r)
sinon row of x $d_1$ L (r)

A = indice

L (A x r) = si STRONG (L(r), int) alors row of sinon 0

A = zones d'indices

L (A x r) = si STRONG (ser x L (r), int) alors row of 1 sinon 0

A chaque indice on fait correspondre le symbole row of et à chaque zone d'indice le symbole row of 1, la fonction rg effectue la vérification de ces conditions et détermine le mode d'une tranche.

b) Le mode d'un appel est celui du résultat de la procédure qu'elle désigne si les conditions suivantes sont réalisées :

- les modes des paramètres effectifs sont respectivement fortement compatibles aux modes des paramètres formels (cf 3.3.6.2) ;
- la transformation par firm (cf 3.3.6.1) du mode du primaire est possible (le primaire désigne une procédure avec paramètres).

Sinon il y a erreur.

c) Le mode d'un moule vide est $\Lambda$ si le mode de sa clause unitaire est fortement compatible au mode void (cf 3.3.6.2).

3.3.5.5 Mode des dénotations

a) Les dénotations numériques peuvent être représentées en longueur simple, double, triple...

- Le mode d'un entier (réel) en simple longueur est int (réel).
- Soit un entier (réel) de longueur i, notons $\mu$ son mode, alors le mode d'un entier (réel) de longueur i+1 est long $\times \mu$ = long $^{i} \times$ int (long $^{i} \times$ real)

b) Le mode d'une dénotation booléenne est bool

c) Le mode d'une dénotation de caractère est char

d) Comme les dénotations numériques, les dénotations de bits peuvent être réprésentées en longueur simple, double, triple ..., le mode d'une rangée de bit est bits.

Soit une dénotation de bits de longueur i dont le mode est $\mu$, alors la dénotation de bits de longueur i+1 a pour mode long $\times \mu$ = long $^{i} \times$ bits.

e) Une chaîne est une rangée de caractères elle est du mode row of $\times$ char.



b) A = appel

L (A $\times$ r) = si STRONG (coll $\times$ $d_1$ L(r), $c_1$ firm $c_1$ L (r))
alors $c_2$ firm $c_1$ L (r) sinon 0

A = paramètres virtuels

L (A $\times$ r) = struct $\times$ L (r)

A = empreinte double, empreinte simple

L (A $\times$ r) = A $\times$ L (r)

c) A = moule vide

L (A $\times$ r) = si STRONG (L (r), void) alors void sinon 0

3.3.5.5' a) A = entier

L (A $\times$ r) = int

A = réel

L (A $\times$ r) = real

A = dénotation numérique

L (A $\times$ r) = si $c_1$ (r) = long alors long $\times$ $d_1$ L (r) sinon L (r)

A = int, real, long, bool, char, bits, rowof, ref, struct proc, proc1

L (A $\times$ r) = A $\times$ r

b) A = true, false

L (A $\times$ r) = bool

c) A = dénotation de caractères

L (A $\times$ r) = char

d) A = dénotation de bits

L (A $\times$ r) = si $c_1$ L (r) = long alors long $\times$ $d_1$ L (r) sinon L(r)

A = rangée de bits

L (A $\times$ r) = bits

e) A = dénotation de chaine

L (A $\times$ r) = row of $\times$ char

### 3.3.5.6 Vérification sur les déclarations

a) La fonction de transport de déclarations a remplacé tout identificateur et toute indication de mode par leur mode, et a associé à chaque opérateur les empreintes des opérations qu'il représente. S'il existe encore, des identificateurs et des indications de modes ou s'il existe des opérateurs sans empreintes d'opération, c'est une erreur.

S'il existe des opérateurs dyadiques (op 2) sans priorité, c'est une erreur.

b) Une déclaration d'identité est correcte si la condition suivante est réalisée :

- le mode de la clause unitaire est fortement compatible au mode du déclareur formel (cf 3.3.6.2).

c) Une déclaration d'opération est correcte si la condition suivante est réalisée :

- le mode de sa clause unitaire est égal au mode de son empreinte (cf 3.3.6.3).

d) Le mode d'un déclareur d'union est l'union des modes de ses composantes si la condition suivante est réalisée :

- un mode union de plusieurs modes ne peut contenir deux modes liés (3.3.5.10).

e) Une borne de tableau est correcte si son mode est fortement compatible au mode int (cf 3.3.6.2).

### 3.3.5.7 Mode d'une formule

A chaque opérateur on a associé les empreintes des opérations qu'il peut désigner (cf 3.3.4.2), l'opération désignée effectivement est celle dont la déclaration est contenue dans la plus petite région contenant la formule, et dans laquelle l'empreinte vérifie la condition suivante :

- le (les) mode (s) de l' (des) opérande (s) effectif (s) de la formule est (sont respectivement) fermement compatible (s) au (x) mode (s) de l' (des) opérande (s) virtuel (s) de l'empreinte.

Le mode de la formule est le mode du résultat de la déclaration trouvée.

La fonction res recherche ce résultat.



### 3.3.5.6' a)

A = identificateur, indication de mode

$L(A \times r) = 0$

A = opérateur monadique

$L(A \times r) = \text{si } d_1\ L(r) = \Lambda \text{ alors } 0 \text{ sinon } d_1\ L(r)$

A = op 2

$L(A \times r) = \text{si } d_3(r) = \Lambda \text{ alors } 0 \text{ sinon } d_3\ L(r)$

A = empreinte double, empreinte simple

$L(A \times r) = A \times L(r)$

b) A = déclaration d'identité

$L(A \times r)$ STRONG $(c_3\ L(r),\ c_2\ L(r))$

c) A = déclaration d'opération

$L(A \times r) = \text{eq}(b\ c_1\ L(r),\ d_1\ L(r))$

d) A = union

$L(A \times r) = \text{si non} L(L(r)) = \Lambda \text{ alors } L(r) \text{ sinon } 0$

e) A = borne de tableau

$L(A \times r) = \text{STRONG}(L(r),$ int $)$

#### Définition de la fonction res

$\text{res}(r, \Lambda) = 0$

$\text{res}(r, s_1 + s_2) = \text{si res}(r, s_1) \neq \Lambda \text{ alors res}(r, s_1)$ sinon $\text{res}(r, s_2)$

$\text{res}(r, A \times s) = \text{res}(r, s)$ sauf

A = empreinte double

$\text{res}(r, A \times s) = \text{si FIRM}(c_1(r), c_1(s)$ et $\text{FIRM}(c_2(r), c_2(s))$ alors $c_3(s)$ sinon $\Lambda$

A = empreinte simple

$\text{res}(r, A \times s) = \text{si FIRM}(c_1(r), c_1(s))$ alors $c_2(s)$ sinon $\Lambda$

3.3.5.8 Mode d'une sélection

Soit $x_0$ le sélecteur de la sélection.

Soit $y_0$ le secondaire.

Pour que la sélection soit possible il faut que le mode de $y_0$ puisse être transformé par weak (cf 3.3.5) en un mode $\mu'_0$ avec :

$\mu'_0$ = struct ( .. , $\mu_i$ $x_i$, .. ) ou $\mu'_0$ = ref struct ( .. , $\mu_i$ $x_i$, .. ) et $x_i$ sélecteur de mode $\mu_i$.

S'il existe i tel que $x_0 = x_i$ alors le mode de la sélection est $\mu_i$ (ref $\mu_i$).

3.3.5.9 Mode d'une tranche

Soit $\mu_0$ le mode du primaire d'une tranche. Une condition de contexte sur ce mode $\mu$ est qu'il puisse être transformé par weak (cf 3.3.5) en un mode $\mu'_0$ = row of$^n$ $\mu$ (ref row of$^n$ $\mu$).

Soit $n_1$ le nombre d'indices de la tranche,

$n_2$ le nombre de zones d'indices de la tranche,

alors si $n_1 + n_2 > n$ c'est une erreur

si $n_1 + n_2 \leq n$ alors le mode de la tranche est row of$^{n-n_1}$ $\mu$ (ref row of$^{n-n_1}$ $\mu$) ; en particulier si $n_1 = n$ et $n_2 = 0$ le mot est $\mu$ (ref $\mu$). La fonction rg effectue les vérifications sur $\mu_0$ et calcule le mode de la tranche.

3.3.5.10 Vérification sur les déclareurs d'union

Deux modes $\mu_1$, $\mu_2$ sont dits liés s'il existe un mode $\mu'$ tel que FIRM ($\mu'$, $\mu_1$) = FIRM ($\mu'$, $\mu_2$) = $\Lambda$.

Un déclareur d'union ne peut contenir deux modes liés.

Pour vérifier cette condition, on opère de la façon suivante :

Etant donnés $n_{-1}$ modes $\mu_2$, ..., $\mu_{n-1}$ non liés et un mode $\mu_1$.

$\mu_1$, $\mu_2$, ..., $\mu_n$ sont non liés si

$\mu_1$ n'est lié à aucun des $\mu_i$ i = 2, n.



3.3.5.8' Définition de la fonction select

select (r, $\Lambda$) = 0

select (r, $s_1 + s_2$) = select (r, $s_1$) + select (r, $s_2$)

select (r, A x s) = si A = ref alors A x select (r, s)

si A = struct alors select (r, s)

si A = champ formel
champ effectif
champ virtuel

alors si r = $c_2$ (s) alors $c_1$ (s)

sinon 0.

3.3.5.9' Définition de la fonction rg

rg (r, $\Lambda$) = 0

rg (r, $s_1 + s_2$) = 0

rg (r, A x s) = ranger (r, A x s) sauf

A = ref

rg (r, A x s) = ref ranger (r, s)

Définition de la fonction ranger

ranger ($\Lambda$, s) = s

ranger ($r_1 + r_2$, s) = 0

ranger (A x r, s) = 0 sauf

A = row of 1

ranger (A x r, s) = si $\rho$ (s) = row of alors row of x ranger (r, b(s))
sinon 0

A = row of

ranger (A x r, s) = si $\rho$ (s) = row of alors ranger (r, b(s))
sinon 0

3.3.5.10' Définition de la fonction nonL

nonL ($\Lambda$) = $\Lambda$

nonL (r+s) = n$\ell$ (r, s) et nonL (s)

nonL (A x r) = $\Lambda$

Définition de la fonction n$\ell$

n$\ell$ (r, $\Lambda$) = $\Lambda$

n$\ell$ (r, $s_1 + s_2$) = n$\ell$ (r, $s_1$) et n$\ell$ (r, $s_2$)

n$\ell$ (r, A x s) = non lié (r, A x s).

3.3.5.11 Modes liés

Deux modes $\mu_1$ et $\mu_2$ sont liés s'il existe $\mu$ tel que

FIRM ($\mu$, $\mu_1$) = FIRM ($\mu$, $\mu_2$) = $\Lambda$

ou encore :

$\mu$ = (proc, ref)$^{n_1}$ $\mu'_1$ = (proc, ref)$^{n_2}$ $\mu'_2$ et

$\mu_1$ = (proc, union)$^{m_1}$ $\mu'_1$ et $\mu_2$ = (proc, union)$^{m_2}$ $\mu'_2$.

On peut toujours supposer que $\mu'_1$ ne commence pas par proc si $\mu'_1$ = proc $\mu''_1$ on remplace alors $\mu'_1$ par $\mu''_1$, $n_1$ par $n_1 + 1$, et $m_1$ par $m_1 + 1$ ; de même pour $\mu'_2$.

Supposons $n_1 > n_2$ on obtient alors :

$\mu_1$ = (proc, union)$^{m_1}$ $\mu'_1$ et $\mu'_2$ = ref (proc, ref)$^{n_1 - n_2 - 1}$ $\mu'_1$.

On en déduit que :

FIRM ($\mu'_2$, $\mu_1$) = $\Lambda$

$\mu'_2$ commence par ref

$\mu_2$ = (proc, union)$^{m_2}$ ref (proc, ref)$^{n_1 - n_2 - 1}$ $\mu'_1$

Réciproquement si $n_2 > n_1$ on obtient :

FIRM ($\mu'_1$, $\mu_2$) = $\Lambda$

$\mu'_1$ commence par ref

$\mu_1$ = (proc, union)$^{m_1}$ ref (proc, ref)$^{n_2 - n_1 - 1}$ $\mu'_2$

Si $n_1 = n_2$ alors $\mu'_1 = \mu'_2$

$\mu_1$ = (proc, union)$^{m_1}$ $\mu'_1$ = (proc, union)$^{k_1}$ $\mu''_1$

$\mu_2$ = (proc, union)$^{m_2}$ $\mu'_1$ = (proc, union)$^{k_1}$ $\mu''_1$

où $\mu''_1$ ne commence ni par proc ni par union

et FIRM ($\mu''_1$, $\mu_2$) = $\Lambda$ FIRM ($\mu''_1$, $\mu_1$) = $\Lambda$

Soit deb l'application qui à tout mode $\mu$ fait correspondre le mode $\mu'$ tel que :

$\mu$ = (proc, union)$^n$ $\mu'$

$\mu'$ ne commence ni par proc, ni par union, alors on obtient :

$\mu_1$ et $\mu_2$ liés $\Leftrightarrow$ FIRM (deb ($\mu_2$), $\mu_1$) ou FIRM (deb ($\mu_1$), $\mu_2$)

Définition de la fonction lié

lié (r, s) = FIRM (deb (r), s) ou FIRM (deb (s), r).



Définition de la fonction deb

deb (A x r) = A x r sauf

A = proc, union

deb (A x r) = deb (r).

### 3.3.6 Transformations de mode - Egalité de mode

En général, dans une clause, les modes des différentes composantes doivent être homogènes, ou encore compatibles entre eux; dans certain cas ils ne sont pas à priori égaux et pour exprimer cette compatibilité il est nécessaire de préciser des moyens de reconnaissance : les transformations par contrainte. Il existe quatre positions de contraintes : forte - ferme - faible - molle.

Une transformation est un enchainement de modifications élémentaires : (cf [ A. W. ] 8. 2. 0. 1)

- de reperer
- de procédurer
- procédurer
- unir
- élargir
- mettre en rang
- hisser
- neutraliser

La transformation d'un mode c'est-à-dire la suite de modifications utilisées dépend :

- de la position de contrainte
- du mode à priori.

Dans certains cas la transformation dépend aussi du mode à posteriori.

Nous allons distinguer deux sortes de transformations :

- les transformations simples,
- les transformations avec vérification de compatibilité.

#### 3.3.6.1 Transformations simples

Il y a trois transformations simples : soft, weak, firm. Elles ont pour but de modifier les modes en les rendant plus simple ; par exemple la transformation soft remplace les modes de procédures sans paramètre par le mode de leur résultat .

En particulier elles remplacent les modes généralisés de racine ser par un mode déduit de ceux de leurs composantes. Soit $\mu$ un mode généralisé et $\{\mu_i\}_{i=1,n}$ la famille des modes de ses composantes. Alors le mode déduit de $\mu$ est le mode $\mu'$ tel que :

- la transformation simple considérée (soft, weak, firm) transforme un des modes $\mu_i$ en $\mu'$

- tous les modes $\mu_j$ pour j = 1, n sont fortement compatibles au mode $\mu'$.

a) Transformation soft

La transformation soft est un enchainement de la seule modification "de procédurer". Elle a pour but de remplacer le mode d'une procédure sans paramètre par le mode de son résultat ; elle vérifie que le mode obtenu est celui d'une référence à un mode $\mu$.

Cette transformation intervient par exemple, dans les assignations le mode transformé d'une destination (cf [ V. W. ] 8.3.1.1.b) doit être d'un mode référence à $\mu$.

b) Transformation weak

La transformation weak est un enchainement des modifications élémentaires : " de procedurer" et "de reperer". Elle permet de supprimer les racines ref et proc des modes en n'effectuant pas la dernière modification si celle-ci doit être de reperer.

Elle vérifie que le mode résultat est celui d'une structure ou d'un tableau.

c) Transformation firm

La transformation firm est un enchainement des modifications élémentaires : "de procédurer" et " de reperer". Elle permet de supprimer les racines ref et proc de modes en vérifiant que le mode restitué est celui d'une procédure avec paramètres (proc 1) (cf 3.3.1.4-f).



3.3.6.1' Définition de la fonction soft

a)
soft ($\Lambda$) = 0
soft (r+s) = si STRONG (s, soft (r)) alors soft (r)
sinon si STRONG (r, soft (s)) alors soft (s)
sinon 0
soft (A x r) = 0 sauf
A = proc, ser
soft (A x r) = soft (r)
A = ref
soft (A x r) = A x r.

b) Définition de la fonction weak

weak ($\Lambda$) =0
weak (r+s) = si STRONG (s, weak (r)) alors weak (r)
sinon si STRONG (r, weak (s)) alors weak (s)
weak (A x r) = 0 sauf
A = proc, ser
weak (A x r) = weak (r)
A = ref
weak (A x r) = si $\rho$ (r) = ref ou proc alors weak (r)
sinon si $\rho$ (r) = struct ou row of alors A x r
A = struct, row of
weak (A x r) = A x r.

c) Définition de la fonction firm

firm ($\Lambda$) = 0
firm (r+s) = si STRONG (s, firm (r)) alors firm (r)
sinon si STRONG (r, firm (s)) alors firm (s)
sinon 0
firm (A x r) = 0 sauf
A = proc, ref, ser
firm (A x r) = firm (r)
A = proc 1
firm (A x r) = A x r.

### 3.3.6.2 Transformations compatibles

Cette catégorie de transformations a pour but de vérifier qu'un mode a priori peut être transformé en un mode à posteriori : étant donné un mode initial et un mode final il faut rechercher s'il existe une succession de modifications transformant le premier en le second. Il existe deux transformations : relatives aux deux positions de contraintes :

- ferme - fonction FIRM
- forte - fonction STRONG.

Si la transformation ferme (forte) existe le mode à priori est dit fermement (fortement) compatible au mode a postériori.

a) Compatibilité ferme - Fonction FIRM

La transformation de compatibilité ferme est un enchainement des modifications élémentaires : "de procédurer", "de reperer", "procédurer", "unir". Elle est utilisée pour vérifier la validité du mode des opérandes d'une formule (cf 3.3.5.3).

Le mode a priori peut être un mode généralisé :

- si c'est un mode généralisé sériel de modes $\mu_1, \ldots, \mu_n$, alors il doit exister un mode $\mu_i$ fermement compatible au mode a posteriori, et tous les $\mu_j$, $j = 1, n$ doivent être fortement compatibles au mode à posteriori (cf ⌊ V. W. ⌋ 6.1.1-g).
- Si c'est un mode généralisé collatéral de modes $\mu_1, \ldots, \mu_n$ alors le mode a posteriori doit être un mode row of $\times \mu$, il doit exister $\mu_i$ fermement compatible à $\mu$ et tous les $\mu_j$ $j = 1, n$ doivent fortement compatible à $\mu$ (cf ⌊ V. W. ⌋ 6.1.2-e).

La vérification est effectuée de la façon suivante :

a) on effectue les vérifications sur les modes généralisés (fonction FIRM).
b) En "de procédurant" et "de reperant" le mode a priori (fonction fi) et en "desunissant" et "de procédurant" le mode a posteriori (fonction f) on vérifie que la transformation existe.



Définition de la fonction FIRM

FIRM $(\Lambda, s) = 0$

FIRM $(r_1 + r_2, s)$ = FIRM $(r_1, s)$ ou FIRM $(r_2, s)$

FIRM $(A \times r, s) = f_i\ (A \times r, s)$ sauf

A = ser

FIRM $(A \times r, s)$ = FIRM $(r, s)$ et STRONG $(r, s)$

A = coll

FIRM $(A \times r, s)$ = si $\rho\ (s)$ = row of alors

FIRM $(r,\ b\ (s))$ et STRONG $(r,\ b\ (s))$.

Définition de la fonction $f_i$

$f_i\ (\Lambda, s) = 0$

$f_i\ (r_1 + r_2, s) = 0$

$f_i\ (A \times r, s)$ = eq $(A \times r, s)$

ou si A = proc ou ref

alors $f_i\ (r, s)$ ou f $(A \times r, s)$

sinon 0

Définition de la fonction f

f $(r, \Lambda) = 0$

f $(r,\ s_1 + s_2)$ = $(r,\ s_1)$ ou $(r,\ s_2)$

f $(r,\ A \times s)$ = eq $(r,\ A \times s)$

ou si A = proc ou union

alors f $(r, s)$

b) Compatibilité forte - Fonction STRONG

La transformation de compatibilité forte est un enchainement de toutes les modifications élémentaires.

Le mode à priori peut être un mode généralisé.

a) Si c'est un mode généralisé seriel, ses modes composantes doivent être fortement compatibles au mode à postériori (cf [ V. W. ] 6.1.1-f).

b) Si c'est un mode généralisé collatéral, le mode à posteriori doit être soit un mode row of $\times \mu$, soit struct $(\mu_1\ x_1, \ldots, \mu_n\ x_n)$.

- Si le mode à posteriori est row of $\times \mu$ toutes les composantes du mode à priori doivent fortement compatible à $\mu$ (cf [ V. W. ] 6.2.1-c).

- Si le mode a posteriori est struct $(\mu_1, x_1, \ldots, \mu_n, x_n)$ le mode généralisé doit être composé de n modes $\mu'_i$ et chaque $\mu'_i$ doit être fortement compatible à $\mu_i$ (cf [ V. W. ] 6.2.1-f).

La vérification est effectuée de la façon suivante :

a) on effectue les vérifications sur les modes généralisés (Fonction STRONG

b) on vérifie si le mode n'est pas skip ou nil auquel cas il est compatible au mode à posteriori ("hissage").

On vérifie si le mode à posteriori n'est pas void, auquel cas tout mode à priori est compatible ("neutralisation") (fonction strong).

c) En "de procédurant" et "de reperant" (fonction str), ou en "élargissant" (fonction $w_i$) le mode a priori et en "de procédurant", "desunissant" et "mettant en rang" le mode a posteriori (fonction st), on vérifie que la transformation existe.



Définition de la fonction STRONG

STRONG $(\Lambda, s) = \Lambda$

STRONG $(r_1 + r_2, s)$ = STRONG $(r_1, s)$ et STRONG $(r_2, s)$

STRONG $(A \times r, s)$ = strong $(A \times r, s)$ sauf

A = ser

STRONG $(A \times r, s)$ = STRONG $(r, s)$

A = coll

STRONG $(A \times r, s)$ = si $\rho(s)$ = row of alors STRONG $(r, b(s))$

si $\rho(s)$ = struct alors STR $(r, b(r))$.

Définition de la fonction STR

STR $(\Lambda, s) = 0$

STR $(r_1 + r_2, s)$ = STR $(r_1, c_1(s))$ et STR $(r_2, d_1(s))$

STR $(A \times r, s)$ = STRONG $(A \times r, c'_1(s))$

Définition de la fonction strong

strong $(\Lambda, s) = \Lambda$

strong $(r_1 + r_2, s) = 0$

strong $(A \times r, s)$ = si A = hip, skip ou s = void alors $\Lambda$

sinon str $(A \times r, s)$.

Définition de la fonction str

str $(\Lambda, s) = 0$

str $(r_1 + r_2, s) = 0$

str $(A \times r, s)$ = st $(A \times r, s)$ sauf

A = ref ou proc

str $(A \times r, s)$ = st $(A \times r, s)$ ou str $(r, s)$.

Définition de la fonction st

st (r, $\Lambda$) = 0

st (r, $s_1 + s_2$) = 0

st (r, A $\times$ s) = $w_i$ (r, A $\times$ s) sauf

A = proc

st (r, A $\times$ s) = st (r, s) ou $w_i$ (r, A $\times$ s)

A = union

st (r, A $\times$ s) = f (r, A $\times$ s) ou $w_i$ (r, A $\times$ s)

A = row of

st (r, A $\times$ s) = st (r, s) ou $w_i$ (r, A $\times$ s) ou r = $\Lambda$

A = ref

st (r, A $\times$ s) = si $\rho$ (r) = ref alors si (b (r), s) ou $w_i$ (r, A $\times$ s)

Définition de la fonction $w_i$

$w_i$ (r, s) = eq (r, s)

ou r = int et s = real, compl

ou r = real et s = compl

ou $\rho$ (r) = long et $\rho$ (s) = long et $w_i$ (b (r), b (s))

ou r = bits et s = row of $\times$ bool

ou r = bytes et s = row of $\times$ char.



3.3.6.3 Egalité de deux modes - Fonction eq

Deux modes non unis sont égaux s'ils sont représentés par la même ramification.

Duex modes unis $\mu_1$ et $\mu_2$ sont égaux si l'ensemble des composantes de $\mu_1$ est égal à l'ensemble des composantes de $\mu_2$.

Définition de la fonction eq

eq ($\Lambda$, s) = 0

eq ($r_1 + r_2$, s) = eq ($r_1$, $c_1$ (s)) et eq ($r_2$, $d_1$ (s))

eq (A $\times$ r, s) = si A = $\rho$ (s) alors eq (r, b(s))

ou si A = union et $\rho$ (s) = union

alors eq (r, b (s)) et eq (b (1), r)

sinon 0

Définition de la fonction eg

eg ($\Lambda$, s) = 0

eg ($r_1 + r_2$, s) = eg ($r_1$, s) et eg ($r_2$, s)

eg (A $\times$ r, s) = si s $\neq \Lambda$ alors

eq (A $\times$ r, $c_1$ (s)) ou eg (A $\times$ r, $d_1$ (s))

sinon 0.

# CONCLUSION

L'étude précédente pourrait être prolongée de diverses manières.

Nous avons donné une définition complétement formelle de langages de programmation reposant sur la notion de fonction récursive de ramifications. En réalité, cette notion aurait besoin d'être précisée : nous avons défini au chapitre 1 celle de fonction récursive primitive, mais nous avons utilisé certaines fonctions qui ne sont pas définies comme telles; on peut montrer [QUERE] que certaines d'entre-elles sont bien récursives primitives, mais d'autres ne le sont sans doute pas.

Au contraire de ce qui est fait habituellement, nous définissons un langage par un algorithme de reconnaissance. L'avantage est qu'on est certain de ne définir ainsi que des langages récursifs. Il serait cependant intéressant de comparer ce type de "grammaires" à des "grammaires génératives", notamment la grammaire proposée pour Algol 68 à celle de Van Wijngaarden ;

Les grammaires du type de celle de Van Wijngaarden permettent d'engendrer tout langage récursivement énumérable [SINTZOFF] :

- comment démontrer qu'une grammaire de notre type engendre le même langage qu'une grammaire de Van Wijngaarden ?
- peut-il y avoir un passage automatique de l'une à l'autre ?
- quelles sont les grammaires de Van Wijngaarden qui sont équivalentes à une grammaire de notre type ?

Une telle définition des langages permet une reconnaissance automatique. Pour permettre son implémentation, il est indispensable de disposer facilement des ramifications et des fonctions récursives sur elles ; on peut pour cela avoir recours à des langages tels que LISP. Une étude est en cours pour définir de tels langages.

La définition donnée pour Algol 60 ou Algol 68 conduirait à une inefficacité certaines de la reconnaissance. Cette inefficacité est essentiellement due à la forme des fonctions de transport utilisées. Il faudrait définir une fonction générale de transport, qui serait traduite en utilisant comme à l'habitude des tables et des fonctions de recherche dans ces tables.

## REFERENCES


BACKUS J. W. — Report on the algorithmic language Algol 60
Numerische Mathematik 2 (1960) p. 106-136

GROSS M - LENTIN A. — Notions sur les grammaires formelles
Gauthier-Villars Paris (1967)

MAROLDT J. — Thèse de spécialité à paraitre.
Faculté des Sciences de Nancy.

NAUR P. — Revised report on the Algorithmic langage ALGOL 60
COMMUN (ACM Juin 1963).

PAIR C. — Sur des notions algébriques liées à l'analyse syntaxique.
Exposé fait au Centre d'Automatique de l'Ecole des Mines - Fontainebleau (mars 1969).

PAIR C. - QUERE A. — Definition et étude des bilangages réguliers.
Information and Control 13, 565 - 593 (1968).

QUERE A. — Etude des ramifications et des bilangages.
Thèse de spécialité - Faculté des Sciences de Nancy (Juillet 1969).

SINTZOFF M. — Existence of a VAN WIJNGAARDEN syntax for every recursevly enumerable set.
Ann SCC. SCIENTIF. de Bruxelles 81, II : 115 - 118 (1967)

VAN WIJNGAARDEN A. - MAILLOUX B. J. - PECK J. E. L. - KOSTER C. H. A.
Report on the algorithmic language Algol 68.
Amsterdam - MR 101 - (Février 1969).


---

NOM DE L'ETUDIANT : LECLAIRE J.Marie

Nature de la thèse : Doctorat de Spécialité en Mathématiques Appliquées

Vu, Approuvé

et Permis d'imprimer

NANCY, le 18 Avril 1970

Le Doyen,

UNIVERSITÉ DE NANCY * FACULTÉ DES SCIENCES *

J. AUBRY