UNIVERSITÉ DE NANCY

FACULTÉ DES SCIENCES

ESSAI DE FORMALISATION DE LA DESCRIPTION DES COMPILATEURS

APPLICATION A ALGOL 60

2° Sujet : Propositions données par la Faculté

THÈSE

présentée à la

Faculté des Sciences de l'Université de Nancy

pour l'obtention du grade de

DOCTEUR INGÉNIEUR

par

Raymond KHALIL

soutenue le 8 Mai 1970
devant la Commission d'Examen

7

JURY : M. J. LEGRAS, *Président*
M. C. PAIR, *Examinateur*
M. C. GILORMINI, *Examinateur*

UNIVERSITÉ DE NANCY

FACULTÉ DES SCIENCES

ESSAI DE FORMALISATION DE LA DESCRIPTION DES COMPILATEURS

APPLICATION A ALGOL 60

2° Sujet : Propositions données par la Faculté

THÈSE

présentée à la

Faculté des Sciences de l'Université de Nancy

pour l'obtention du grade de

DOCTEUR INGÉNIEUR

par

Raymond KHALIL

soutenue le 8 Mai 1970
devant la Commission d'Examen

JURY : M. J. LEGRAS, *Président*
M. C. PAIR, *Examinateur*
M C. GILORMINI. *Examinateur*

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

L'idée de ce travail est due à Monsieur le Professeur PAIR, vers qui va toute ma reconnaissance pour en avoir assuré la direction avec attention et bienveillance.

Je remercie Monsieur le Professeur MARI, Directeur de l'Institut des Sciences de l'Ingénieur, pour l'accueil qu'il m'y a réservé, ainsi que Monsieur GILORMINI, Maître de Conférences, qui a accepté de participer au jury.

Je remercie également mes collègues et collaborateurs de l'I.U.C.A. et de l'I.S.I.N. pour l'aide qu'ils m'ont apportée et les nombreux services qu'ils m'ont rendus.

Université de Nancy — Faculté des Sciences

ANNEE SCOLAIRE 69/70

DOYEN : M. AUBRY

ASSESSEUR : M. GAY

Doyens honoraires : MM. CORNUBERT - ROUBAULT

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

Maîtres de Conférences honoraires : MM. LIENHART - PIERRET - Melle 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 Org. Ind.
BOLFA	Minéralogie	BERNARD	Géologie Appliq.
NICLAUSE	Chimie	*CHAMPIER	Physique
FAIVRE	Physique Appliquée	*GAY	Chimie Biolog.
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	FELDEN5	Physique
SUHNER	Physique Expériment.	*GOSSE	Mécanique phys.
CHAPON	Chimie Biologique	*DAVOINE	Physique (ENSMIN)
HEROLD	Chimie Minérale Indus.	HORN	Physique (1 Cycle)
SCHWARTZ	Exploitation Minière	*ROCCI	Géologie
MALAPRADE	Chimie	*Mme LUMER	Mathématiques
*MANGENOT	Botanique	DELPUECH	Chimie physique
BLAZY	Minéralogie Appliquée	Melle HUET	Analyse Supér.

N... Chimie Biologique
N... Mécanique Appliquée
N... Méthodes Mathématiques de la Physique
N... Mécanique Rationnelle

(*) Professeur titulaire à titre personnel.

PROFESSEURS SANS CHAIRE

Mme	BASTICK	Chimie P.C. EPINAL	
M.M	GUDEFIN	Physique	
	VUILLAUME	Psychophysiologie	
	FRENTZ	Biologie Animale	
	MARI	Chimie (ISIN)	
	DEVIOT	Physique du solide	
	FLECHON	Physique P.C.	
	VIGNES	Métallurgie	
	JURAIN	Géologie C.B.G.	
	BALESDENT	Thermodynamique, Chimie Appliquée (ENSIC)	
	JANOT	Physique P.C. EPINAL	
	CACHAN	Entomologie Appliquée (ENSA)	
	JACQUIN	Pédologie & chimie agr.	
	MAINARD	Physique M.P.	
	MARTIN	Chimie P.C.	
	PAULMIER	Mécanique Expérim.	
	PROTAS	Minéralogie	
	RIVAIL	Chimie Appl. (CUCES)	
	DEPAIX	Probabilité et Statist.	

MAITRES DE CONFERENCES

MM.	VILLERMAUX	Génie Chimique	FERRIER	Mathématiques
	METCHE	Biochimie Appliquée (BRASSERIE)	GILORMINI	Mécanique des Fluides (ISIN)
	BAUMANN	Physique (1 Cycle)	HILY	Mathématiques P.C.
	DURAND	Physique	CASTRO	Chimie Organique
	GRANGE	Physique (ISIN)	DEXHEIMER	Botanique
	BAVEREZ	Chimie (ENSIC)	NOVERRAZ	Mathématiques
	CHAMBON	Exploitation Minière (MINES)	TOURET	Géologie
			N...	Maths Appliquées
	HUSSON	Physique (ENSEM)	N...	Mathématiques
	WEISSLINGER	Physique	N...	Mathématiques
	GERL	Physique (ENSEM)	N...	Mathématiques
	ROQUES	Chimie dMinérale	N...	Physiologie animale
			N...	Chimie (ENSIC)
			N...	Laiterie (ECCLE LAIT.)
			N...	Chimie Organique
			N...	Mathématiques Appl.

CHARGES D'ENSEIGNEMENT

MM. AMARIGLIO - COEURE - DAVRAINVILLE - GIRARDEAU - CVAERT - RUYER - WEBER - PUJOL - SCHREIBER.

TABLE DES MATIERES

= = = = = = = = = =

I - INTRODUCTION -

Le but de ce travail est de donner une description formelle simple d'un compilateur.

La compilation peut être conçue comme une suite de transformations : d'abord celle d'une chaîne de caractères en une ramification, ensuite un certain nombre de transformations de ramifications, enfin la transformation d'une ramification en une chaîne.

Dans la pratique, ces transformations sont souvent mêlées. La recherche des qualités de généralité, clarté et simplicité pour la description et ses applications (l'enseignement de la compilation par exemple) nous les a fait étudier séparément. Nous considérerons sept parties :

- l'analyse syntaxique. Le langage source est défini par une grammaire à contexte libre. Les techniques d'analyse syntaxique sont bien connues et n'entrent pas dans le cadre de notre étude. Nous partirons de la ramification fournie par cette analyse.
- la vérification des conditions de syntaxe du langage source, non exprimées par la grammaire à contexte libre. (cf. [7]). Nous ne nous occuperons pas de cette partie.
- une phase de transformations préliminaires visant un double but : la simplification des instructions syntaxiquement complexes en utilisant les équivalences sémantiques à l'intérieur du langage source et le traitement des constantes.
- une étude lexicographique qui fournit une table des identificateurs (variables ou étiquettes) avec, pour les variables, une adresse associée selon le mode de gestion de mémoire adopté.
- une phase d'exploitation de cette table, transportant les renseignements associés aux identificateurs, dans le programme où ils remplaceront les occurences de ces identificateurs.
- une étude des résultats intermédiaires, prévoyant les éventuelles conversions de type, et attachant à chaque résultat intermédiaire son type et l'adresse dévolue par la gestion mémoire.
- une dernière phase enfin de génération des instructions du programme objet, utilisant toutes les informations accumulées par les précédents traitements. Cette dernière phase sera suivie d'un assemblage qui déterminera les adresses des instructions, en fonction d'un mode de chargement.

L'étude détaillée des cinq dernières parties, dans le cas de la compilation d'Algol 60, fait l'objet des chapitres 3 et 4 de ce travail.

Nous considérerons ces transformations comme des fonctions récursives de ramifications. Elles seront décrites dans un langage complètement formalisé qui est exposé au chapitre 2. Ce chapitre comprend également quelques rappels concernant les ramifications et les fonctions récursives de ramifications.

Si l'on implémentait le langage de description, on pourrait obtenir un compilateur à partir d'une description comme celle que nous proposons.

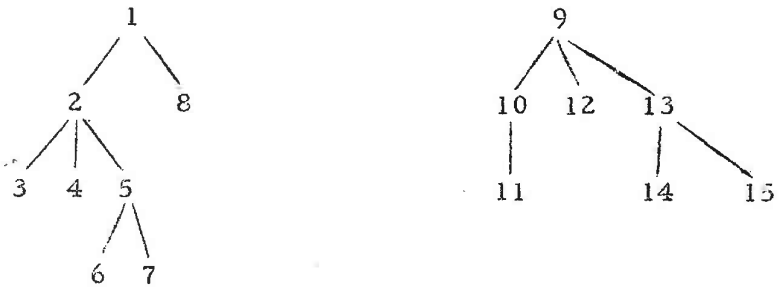
II - LE FORMALISME DE DESCRIPTION

II.1 - RAPPELS SUR LES RAMIFICATIONS ET FONCTIONS RECURSIVES

Ces quelques rappels sont inspirés de [10] et [13] où l'on trouvera une plus ample justification des procédés utilisés dans la suite.

II.1.1) Forêt : on appelle forêt un graphe (E,Γ) sans circuit, tel que, pour tout $x \in E$, l'ensemble $\Gamma^{-1}(x)$ contienne au plus un élément.
Si $\Gamma^{-1}(x)$ est vide, on dit que x est racine.
Si $\Gamma(x)$ est vide, on dit que x est une feuille.
Chaque composante connexe d'une forêt est une arborescence.

exemple



- forêt à deux composantes connexes -

II.1.2) Orientation d'une forêt : une orientation d'une forêt (E,Γ) est un ordre partiel O dans l'ensemble E tel que

a) les restrictions de O à l'ensemble des racines et à chacun des ensembles $\Gamma(x)$ $(x \in E)$ sont des ordres totaux
b) si $y \in \Gamma(x)$ et $z \notin \Gamma(x)$, y et z ne sont pas comparables par la relation O

Un triple (E,Γ, O) où (E,Γ) est une forêt et O une de ses orientations s'appelle forêt orientée. La forêt orientée particulière pour laquelle $E = \emptyset$ est dite vide.
Soient deux forêts orientées (E,Γ,O) et (E',Γ',O') ; un isomorphisme de la première sur la deuxième est une bijection ψ de E dans E' telle que

$(\forall x, y \in E)((x \Gamma y \iff \psi(x)\ \Gamma'\ \psi(y))$ et $(x\ O\ y \iff \psi(x)\ O'\ \psi(y)))$

II.1.3) Ramifications sur un ensemble V

Etiqueter les points d'une forêt par les éléments d'un ensemble V revient à définir une application de l'ensemble des points de la forêt dans V. Il est souhaitable que deux forêts orientées isomorphes, dont les points qui se correspondent dans l'isomorphisme ont même étiquette, déterminent la même ramification. Cela conduit à la définition :

Soit un ensemble V, introduisons dans l'ensemble des couples (I, F) formés par une forêt orientée I dont les points sont des entiers, et une application F de l'ensemble des points de I dans V, la relation d'équivalence

$(I, F) \sim (I', F') \iff$ il existe un isomorphisme φ de I sur I' tel que $F = F' \cdot \varphi$

Une ramification sur V est une classe de cette équivalence. Elle pourra être représentée par l'un de ses couples (I, F). On appellera pseudo-arborescence une ramification à une racine.
Nous noterons $\hat{V}$ l'ensemble des ramifications sur V.

Cette définition est relativement complexe ; la théorie algébrique qui suit permettra d'engendrer les ramifications de $\hat{V}$ à partir de V grâce à deux lois de composition, et ensuite, de ne plus guère employer explicitement la définition.

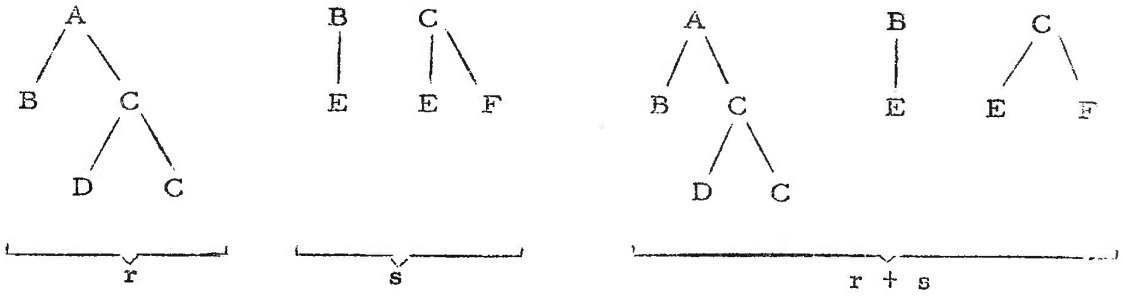
Notons deux cas particuliers de ramifications sur V :

- la ramification vide, notée \$, classe de (I, F) où I est la forêt vide
- les ramifications (I, F) où I ne possède qu'un point x.
  Une telle ramification peut être identifiée à $F(x) \in V$, ainsi V est contenu dans $\hat{V}$.

II.1.4) Produit

Nous introduisons, ici, une loi de composition interne dans V, le produit, noté +. Intuitivement, r + s est obtenu en plaçant la ramification s à droite de la ramification r .

exemple



Soient deux ramifications r = (E, Γ, O) et s = (E', Γ', O') pour lesquelles on peut supposer E et E' disjoints ;
r + s est la ramification (E'', Γ'', O'') définie par :

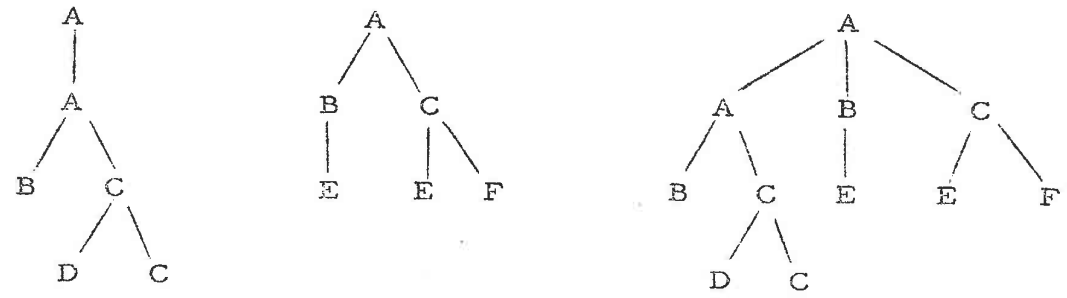
- E'' est la réunion de E et E' ;
- Γ'' est la relation (Γ ou Γ') ;
- x O'' y $\iff$ x O y ou x C' y ou (x est racine de (E, Γ) et y est racine de (E', Γ')) ;
- f'' (x) = f (x) si $x \in E$, f' (x) si $x \in E'$;

le produit ainsi défini est une loi de composition interne associative dont la ramification vide \$ est élément neutre.

II. 1. 5) Enracinement

Ce sera une loi de composition externe à opérateur dans V, notée $*$ avec opérateur à gauche. Intuitivement, A * r est obtenu en adjoignant à r une racine A

exemple avec les ramifications r et s de l'exemple précédent



A * r A * s A * (r + s)

Soit une ramification (E, Γ, O, f) sur V et un élément A de V, $A * r$ est la ramification (E', Γ', O', f') définie par :

- $E' = E \cup \{a\}$ où $a \notin E$
- $x \Gamma' y \iff x \Gamma y$ ou ($x = a$ et y est racine de (E, Γ))
- $x O' y \iff x O y$ ou $x = y = a$
- $f'(x) = f(x)$ si $x \in E$, $f'(a) = A$

Avec ces notations, les ramifications r et s de l'exemple s'écrivent

$$r = A * (B + (C * (D + C)))$$
$$s = (B * E) + (C * (E + F))$$

On admettra, par la suite, que l'enracinement a la priorité sur le produit et l'on pourra écrire

$$r = A * (B + C * (D + C))$$
$$s = B * E + C * (E + F)$$

II.1.6) Binoïde sur un ensemble V

On appelle binoïde sur un ensemble V un ensemble muni d'une loi de composition interne associative, admettant un élément neutre, et d'une loi de composition externe à opérateurs dans V.

$\hat{V}$ est donc un binoïde sur V.

II.1.7) Proposition

Pour tout $r \in \hat{V}$, il existe $A \in V$, $s \in \hat{V}$, $t \in \hat{V}$ uniques, tels que $r = (A * s) + t$, ce qui résulte facilement des définitions. On en déduit le

II.1.8) Principe de récurrence dans $\hat{V}$

Soit P un prédicat, tel que

- P ($) soit vrai
- $(\forall r, r' \in \hat{V})$ (P (r) et P(r') $\Longrightarrow$ P(r + r'))
- $(\forall r, \in \hat{V})$ $(\forall A \in V)$ (P (r) $\Longrightarrow$ P(A * r))

alors, P (r) est vrai pour tout $r \in \hat{V}$.

On peut employer ce principe de récurrence pour démontrer que toute ramification sur V est obtenue par un nombre fini de compositions d'éléments par + et $*$.

Une autre conséquence de la proposition est l'important théorème qui permet de justifier l'existence d'applications dont $\hat{V}$ est l'ensemble de départ, et qui sont définies récursivement à l'aide des deux lois de compositions de $\hat{V}$

II.1.9) Théorème

Pour tout binoïde B sur V, il existe un homomorphisme et un seul de $\hat{V}$ dans B.

Avec les notations précédentes, un homomorphisme de $\hat{V}$ dans B est une application ψ de $\hat{V}$ dans B, telle que, pour r, $s \in \hat{V}$ et $A \in V$: $\psi(r + s) = \psi(r) + \psi(s)$; $\psi(\$) = \$$; $\psi(A * r) = A * \psi(r)$; ce qui conduit à nommer $\hat{V}$ binoïde universel déduit de V.

II.1.10) Ramification engendrée par une grammaire

Une grammaire sera ici un quadruplet (T, N, ::=, X) où :

- T et N sont des ensembles finis. T est le vocabulaire terminal et V = T∪N est le vocabulaire de la grammaire ;
- ::= est une relation binaire entre N et le monoïde universel V^* telle que, pour tout $A \in N$, l'ensemble des mots φ vérifiant $A ::= \varphi$ soit un langage régulier ; 11
- X est un langage régulier sur V ;

L'ensemble $\mathcal{L}$ des ramifications engendrées, au sens large, par la grammaire est la plus petite partie de $\hat{V}$:

- contenant T
- stable pour le produit
- telle que si $r \in \mathcal{L}$ et si le mot des racines $\mathcal{P}(r)$ vérifie $A ::= \mathcal{P}(r)$ $(A \in V)$ alors, $A \uparrow r \in \mathcal{L}$ (voir II.1.18 c)

Une ramification $r \in \mathcal{L}$ est dite engendrée par la grammaire si $\mathcal{P}(r) \in X$.

notation

Nous appellerons dans la suite $\mathcal{F}^k$ l'ensemble des fonctions de $\hat{V}^k$ dans $\hat{V}$ en convenant que $\mathcal{F}^0 = \hat{V}$

II.1.11) Opérateurs

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

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

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

$(\forall r_1, \ldots, r_k, r, s \in \hat{V}, a \in V, s$ étant une pseudo-arborescence)

$$(f(r_1, \ldots, r_k, \$) = g(r_1, \ldots, r_k),$$

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

$$f(r_1, \ldots, r_k, a*r) = g_a(r_1, \ldots, r_k, r, f(r_1, \ldots, r_k, r))).$$

II.1.12) Fonction de base : ce sont les fonctions suivantes

a) $\$ \in \mathcal{F}^0$

b) $(\forall r_1, r_2 \in \hat{V})\ f(r_1, r_2) = r_1 + r_2$

c) $(\forall A \in V)\ (\forall r \in \hat{V})\ (g_A(r) = A * r)$

d) $(\forall k > 0)\ (\forall 1 \leqslant i \leqslant k)\ ((\forall r_1, \ldots, r_k \in \hat{V})\ \Pi_i^k(r_1, \ldots, r_k) = r_i)$

II.1.13) Fonctions récursives primitives

L'ensemble des fonctions récursives primitives sur $\hat{V}$ est le plus petit ensemble contenant les fonctions de base, et stable par les opérations de composition et récurrence.

II.1.14) Proposition

Une fonction f est récursive primitive si, et seulement si, elle s'obtient à partir des fonctions de base par un nombre fini de compositions et de récurrences.

II.1.15) Conséquence

Si f, g_1, g_2 sont des fonctions récursives primitives à k variables, alors la fonction h :

$h(r_1, \ldots, r_k) = \underline{si}\ f(r_1, \ldots, r_k) = \$\ \underline{alors}\ g_1(r_1, \ldots, r_k)$
$\underline{sinon}\ g_2(r_1, \ldots, r_k)$

est récursive primitive.

II.1.16) Remarque

L'ensemble des fonctions engendrées n'est pas modifié si l'on remplace b et c de II.1.12 par

$(\forall A \in V)\ ((\forall r_1, r_2 \in \hat{V})\ f_A(r_1, r_2) = r_1 + A * r_2)$

II.1.17) Remarque

Certaines des fonctions utilisées au chapitre IV ne sont pas toujours définies à l'aide des seuls opérateurs de composition et récurrence. On trouvera dans [12] et [13] des procédés permettant de se ramener aux cas définis précédemment. On y trouvera, également, une justification du fait que les récurrences peuvent être faites sur l'une quelconque des variables.

II.1.18) Exemples de fonctions récursives primitives

a) les fonctions constantes $(\forall r \in \hat{V})\ Cu(r) = u$

b) l'identité qui n'est autre que la projection p_i^i (voir II.1.12 d)

c) le mot des racines d'une ramification, noté ρ et défini par :

$\rho(\$) = \$$;
$\rho(r + s) = \rho(r) + \rho(s)$;
$\rho(A * r) = A$;

d) le mot des feuilles d'une ramification, noté φ et défini par :

$\varphi(\$) = \$$;
$\varphi(r + s) = \varphi(r) + \varphi(s)$;
$\varphi(A * r) = \text{si } \varphi(r) \neq \$ \text{ alors } \varphi(r) \text{ sinon } A$;

e) la suppression des racines des pseudo-arborescences composant une ramification

$\xi(\$) = \$$;
$\xi(r + s) = \xi(r) + \xi(s)$;
$\xi(A * r) = r$;

f) le nombre de racines d'une ramification, notée ν. Cette fonction prend ses valeurs dans l'ensemble N des entiers positifs. La loi d'addition dans N sera notée $\oplus$. La fonction ν est définie par :

$$\nu(\$) = 0 \ ;$$
$$\nu(r + s) = \nu(r) \oplus \nu(s) \ ;$$
$$\nu(A \ast r) = 1 \ ;$$

g) la première composante connexe d'une ramification notée η et définie par :

$$\eta(\$) = \$ \ ;$$
$$\eta(r + s) = \text{si } \nu(r) = 1 \text{ alors } r \text{ sinon } \eta(r) \ ;$$
$$\eta(A \ast r) = A \ast r \ ;$$

h) la dernière composante connexe d'une ramification notée σ et définie par :

$$\sigma(\$) = \$ \ ;$$
$$\sigma(r + s) = \text{si } s \neq \$ \text{ alors } s \text{ sinon } r \ ;$$
$$\sigma(A \ast r) = A \ast r \ ;$$

i) la suppression de la première composante connexe d'une ramification, notée ε et définie par :

$$\varepsilon(\$) = \$ \ ;$$
$$\varepsilon(r + s) = \text{si } \nu(r) = 1 \text{ alors } s \text{ sinon } \varepsilon(r) + s \ ;$$
$$\varepsilon(A \ast r) = A \ast r \ ;$$

j) la suppression de la dernière composante connexe d'une ramification, notée ω et définie par :

$$\omega(\$) = \$ \ ;$$
$$\omega(r + s) = \text{si } s \neq \$ \text{ alors } r \text{ sinon } \$ \ ;$$
$$\omega(A \ast r) = A \ast r \ .$$

II.2 - LANGAGE DE DESCRIPTION

II.2.1) Introduction

Le langage traite les ramifications sur l'alphabet formé des entiers, des réels et des caractères utilisés par Algol 60. Il permet d'abord la description de fonctions définies sur $\hat{V}^p$ (p variant selon les fonctions) et à valeurs dans $\hat{V}$. Il permet ensuite de définir une transformation sur une ramification donnée, par une expression contenant certaines des fonctions définies et la ramification argument. La syntaxe de ce langage est décrite ci-après en notation de Backus. Un compilateur peut être décrit par un programme dans ce langage, la ramification argument étant celle obtenue par l'analyse syntaxique du texte source.

II.2.2) Syntaxe

<programme> ::= DEBUT <partie déclaration> <expression de ramification> FIN

<partie déclaration> ::= <déclaration> | <partie déclaration> <déclaration>

<déclaration> ::= FONCTION <déclaration récursive> | FONCTION <déclaration non récursive>

<déclaration non récursive> ::= <identificateur> <partie paramètres formels> <expression de ramification> ;

<déclaration récursive> ::= <identificateur> <partie paramètres formels> <déclaration de récursivité> <expression de ramification> ; <expression de ramification> ; <expression de ramification> ;

<déclaration de récursivité> ::= REC (<identificateur>)

<partie paramètres formels> ::= (<liste d'identificateurs>)

<liste d'identificateurs> ::= <identificateur> | <liste d'identificateur> , <identificateur>

<expression de ramification> ::= <expression simple> | <expression booléenne> : <expression simple>, <expression de ramification>

<expression booléenne> ::= <terme booléen> | <expression booléenne> ou <terme booléen>

<terme booléen> ::= <secondaire booléen> | <terme booléen> ET <secondaire booléen>

<secondaire booléen> ::= <primaire booléen> | NON <primaire booléen>

<primaire booléen> ::= <expression simple> <opérateur de relation> <expression simple> | (<expression booléenne>)

<opérateur de relation> ::= = | ≠ | >

<expression simple> ::= <expression simple> + <terme> | <terme>

<terme> ::= <racine> ✳ <terme> | <primaire>

<primaire> ::= (<expression de ramifications>) | <racine> | <indicateur de fonction> | $

<racine> ::= <expression arithmétique> | ' <chaîne de caractères> '

<indicateur de fonction> ::= <identificateur> (<liste d'expressions simples>)

<liste d'expressions simples> ::= <liste d'expressions simples> , <expression simple> | <expression simple>

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

<terme arithmétique> ::= <primaire arithmétique> | <terme arithmétique> ⊛ <primaire arithmétique>

<primaire arithmétique> ::= <nombre> | <identificateur> | <indicateur de fonction> | (<expression arithmétique>)

<opérateur additif> ::= ⊕ | ⊖

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

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

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

<chaîne de caractères> ::= <caractère> | <chaîne de caractères> <caractère>

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

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

<caractère> ::= ce sont les chiffres, lettres et délimiteurs au sens de la syntaxe d'Algol 60.

II.2.3) Sémantique

II.2.3.1) expressions arithmétiques

Ce sont les ramifications réduites aux valeurs entières ou réelles que l'on calcule à partir des valeurs des primaires arithmétiques les composant, au moyen des opérations d'addition, soustraction et multiplication de réels, notées $\oplus$, $\ominus$ et $\circledast$.

exemple : l'expression arithmétique : $F(X) \ominus ((3 \oplus X) \circledast 2)$ représente la ramification 1.5, si X vaut 1 et F(X) vaut 9.5

II.2.3.2) expressions simples

Elles représentent des ramifications sur l'ensemble des caractères définis. Les opérations utilisées sont le produit de deux ramifications, noté + et l'enracinement d'une ramification, notée $*$. D'après les règles de syntaxe énoncées, l'opération d'enracinement a la priorité sur l'opération de produit dans une expression simple non parenthésée.
Les opérandes d'une expression simple peuvent être des expressions arithmétiques dont on aura calculé la valeur, des ramifications, ou la ramification vide notée $.

II.2.3.3) expressions booléennes

Ce sont des valeurs booléennes calculées à partir des primaires booléens au moyen des opérations traditionnelles NON, ET, OU (citées, ici, dans l'ordre des priorités décroissantes).
Les primaires booléens sont les valeurs booléennes obtenues par la comparaison de deux ramifications au moyen des opérateurs de relation habituels =, $\neq$ et $>$. L'égalité est définie par la coïncidence, élément par élément, des deux ramifications arguments ou par l'égalité des valeurs des deux expressions arithmétiques. La relation de supériorité stricte n'est définie que pour deux expressions arithmétiques.

II.2.3.4) priorité des opérateurs

Les opérations d'une expression seront effectuées successivement de gauche à droite en respectant les priorités suivantes :

1°	⊛
2°	⊕ ⊖
3°	✳
4°	+
5°	= ≠ >
6°	NON
7°	ET
8°	OU

Les expressions contenues entre parenthèses seront calculées séparément et leurs valeurs utilisées dans la suite des calculs.

II.2.3.5) expression de ramification

Une telle expression a pour valeur une ramification. Lorsque l'expression de ramification se compose d'une expression booléenne, d'une expression simple et d'une seconde expression de ramification, l'expression booléenne est évaluée en premier. Si la valeur obtenue est VRAI, alors l'ensemble a pour valeur la ramification décrite par l'expression simple, sinon la valeur de la deuxième expression de ramification est choisie pour remplacer l'ensemble.

II.2.3.6) déclarations

Ce sont des définitions de fonctions dont les paramètres et les valeurs sont des ramifications.

II.2.3.7) déclaration non récursive

Une telle déclaration est utilisée pour définir une fonction par une seule expression de ramification, construite à partir des paramètres formels de la fonction, et des fonctions déclarées dans le programme.

II.2.3.8) déclaration récursive

Cette déclaration définit une fonction, par récurrence, sur celui de ses paramètres qui est indiqué dans la "déclaration de récursivité".
La première expression de ramification définit la valeur de la fonction pour ce paramètre égal à $.
La seconde expression de ramification définit la valeur de la fonction pour ce paramètre mis sous la forme d'un produit d'une ramification notée R, et d'une pseudo-arborescence notée S.
La troisième expression de ramification définit la valeur de la fonction pour ce paramètre considéré comme le résultat de l'enracinement de la ramification U par la racine T.
Outre ces quatre identificateurs réservés : R, S, T, U, les trois expressions de ramification utilisent tous les paramètres formels de la déclaration, sauf celui sur lequel est effectuée la récurrence, et les fonctions déclarées dans le programme.

II.2.3.9) programme

C'est la description d'une transformation de la ramification argument, notée W, par une expression de ramification utilisant cet unique paramètre, les lois de produit, d'enracinement et certaines des fonctions définies dans la partie déclaration.
L'exécution d'un programme consiste en la construction de la ramification décrite par l'expression de ramification.

II.2.4) Fonctions standard du langage

II.2.4.1)

RO fournit, à partir d'une ramification r, la ramification constituée du produit, dans l'ordre, des racines de r :
RO (r) = $\int^{\circ}$ (r) (cf. II.1.18 c)

II.2.4.2)

SR fournit, à partir d'une pseudo-arborescence s, la ramification obtenue en privant s de sa racine :
SR (s) = ξ (s) (cf. II.1.18 e)

II.2.4.3)

C1 fournit, à partir d'une ramification r, la pseudo-arborescence constituant la première composante connexe de r :
C1 (r) = η (r) (cf. II.1.18 g)

II.2.4.4)

C2 fournit, à partir d'une ramification r, la pseudo-arborescence constituant la deuxième composante connexe de r :
C2 (r) = η (ε (r)) (cf. II.1.18 g et II.1.18 i)

II.2.4.5)

On définit de même C3 (r) = η (ε (ε (r))),
C4 (r) = η (ε (ε (ε (r)))), etc...

II.2.4.6)

CM1 fournit, à partir d'une ramification r, la pseudo-arborescence constituant la dernière composante connexe de r :
CM1 (r) = σ^- (r) (cf. II.1.18 g)

II.2.4.7)

CM2 fournit, à partir d'une ramification r, la pseudo-arborescence constituant l'avant-dernière composante connexe de r :
CM2 (r) = σ^- (ω (r)) (cf. II.1.18 g et II.1.18 j)

II.2.4.8)

On définit de même CM3 (r) = σ^- (ω (ω (r))),
CM4 (r) = σ^- (ω (ω (ω (r)))), etc...

II.2.4.9)

CA1 fournit, à partir d'une ramification r, la ramification obtenue en privant r de sa première composante connexe :
CA1 (r) = ε (r)

II.2.4.10)

CA2 fournit, à partir d'une ramification r, la ramification obtenue en privant r de ses deux premières composantes connexes :
CA2 (r) = ε (ε (r))

II.2.4.11)

On définit de même CA3 (r) = ε (ε (ε (r))),
CA4 (r) = ε (ε (ε (ε (r)))), etc...

II.2.4.12)

CAM1 fournit, à partir d'une ramification r, la ramification obtenue en privant r de sa dernière composante connexe :
CAM1 (r) = ω (r)

II.2.4.13)

CAM2 fournit, à partir d'une ramification r, la ramification obtenue en privant r de ses deux dernières composantes connexes :
CAM2 (r) = ω(ω(r))

II.2.4.14)

On définit de même CAM3 (r) = ω (ω (ω(r)))
CAM4 (r) = ω(ω (ω(ω(r)))), etc...

II.2.4.15)

SC1, SC2, SC3, etc... fournissent, à partir d'une ramification r, la ramification obtenue en privant la 1ère, 2e, 3e, etc... composante connexe de r, de sa racine :
SC1 (r) = SR (C1 (r)) = ξ (η (r))
SC2 (r) = SR (C2 (r)) = ξ (η (ε (r)))
SC3 (r) = SR (C3 (r)) = ξ (η (ε (ε (r)))), etc...

II.2.4.16)

SCM1, SCM2, SCM3, etc... fournissent, à partir d'une ramification r, la ramification obtenue en privant la dernière, l'avant-dernière, l'avant-avant-dernière, etc... composante connexe de r de sa racine :

SCM1 (r) = SR (CM1 (r)) = ξ (σ(r))
SCM2 (r) = SR (CM2 (r)) = ξ (σ(ω(r)))
SCM3 (r) = SR (CM3 (r)) = ξ (σ(ω(ω(r)))), etc...

II.2.4.17)

CS1, CS2, CS3, etc... fournissent, à partir d'une pseudo-arborescence r, la pseudo-arborescence obtenue en privant r de sa racine et en ne conservant du résultat, que la première, deuxième, troisième, etc... composante connexe.
CS1 (r) = C1 (SR (r)) = η (ξ (r))
CS2 (r) = C2 (SR (r)) = η (ϵ(ξ(r)))
CS3 (r) = C3 (SR (r)) = η (ϵ(ϵ(ξ (r)))), etc...

II.2.4.18)

CSM1, CSM2, CSM3, etc... fournissent, à partir d'une pseudo-arborescence r, la pseudo-arborescence obtenue en privant r de sa racine et en ne conservant du résultat que la dernière, l'avant-dernière, l'avant-avant-dernière, etc... composante connexe :
CSM1 (r) = CM1 (SR (r)) = σ(ξ (r))
CSM2 (r) = CM2 (SR (r)) = σ(ω(ξ (r)))
CSM3 (r) = CM3 (SR (r)) = σ(ω(ω(ξ (r)))), etc...

II.2.4.19)

CSA1, CSA2, CSA3, etc... fournissent, à partir d'une pseudo-arborescence r, la ramification obtenue en privant r de sa racine, puis en privant le résultat de sa première, de ses deux premières, de ses trois premières, etc... composantes connexes :
CSA1 (r) = ϵ (ξ (r))
CSA2 (r) = ϵ (ϵ (ξ (r)))
CSA3 (r) = ϵ (ϵ (ϵ (ξ (r)))), etc...

II.2.4.20)

CSAM1, CSAM2, CSAM3, etc... fournissent, à partir d'une pseudo-arborescence r, la ramification obtenue en privant r de sa racine, puis en privant le résultat de sa dernière, de ses deux dernières, de ses trois dernières, etc... composantes connexes :
CSAM1 (r) = ω(ξ (r))
CSAM2 (r) = ω(ω(ξ (r)))
CSAM3 (r) = ω(ω(ω(ξ (r)))), etc...

II.2.4.21)

NR fournit, à partir d'une ramification r, la pseudo-arborescence réduite à un seul élément, le nombre de racines de r :
NR (r) = ν (r)

II.2.4.22)

ETA fournit, à partir d'une pseudo-arborescence réduite à un entier n, un identificateur différent de tous les identificateurs Algol 60 permis par l'implémentation et de tous les identificateurs correspondant à $k \neq n$.

III.1 - DEFINITION DE LA RAMIFICATION DONNEE

III.1.1) Généralités

Le langage source est un sous-ensemble d'Algol 60 ne contenant pas de :

- variables rémanentes
- paramètres formels non spécifiés
- paramètres formels de type tableau, appelés par valeur
- déclarations portant sur plusieurs identificateurs
- affectations multiples.

L'analyse syntaxique transforme le texte source en une ramification ; elle supprime une certain nombre de séparateurs inutiles par la suite, comme le deux-points, le symbole allera, les parenthèses, etc... ; elle transforme les spécifications des paramètres formels et marque différemment les appels par nom des paramètres formels selon qu'ils interviennent à gauche ou à droite du signe d'affectation.

La ramification obtenue est engendrée par la grammaire à contexte libre décrite au paragraphe III.1.3 en notation de Backus étendue (cf. paragraphe III.1.2). Le paragraphe III.1.4 fait la liaison entre nos notations pour les symboles non terminaux, et les dénominations usuelles [1].

Nous supposons de plus que le texte source était un programme Algol 60 correct, qui vérifiait en particulier les conditions relatives aux déclarations. Cela revient à dire que la vérification des diverses conditions syntaxiques contextuelles est supposée faite avec l'analyse syntaxique, avant l'exécution du programme de compilation.

Nous n'avons enfin pas explicité l'écriture des identificateurs et des constantes.

III.1.2) Notation de Backus étendue

On utilisera la notation de Backus avec les conventions supplémentaires :

a) Une règle dont le second membre comporte un ou plusieurs groupements entre parenthèses de séquences de symboles non terminaux séparées par des | , est en fait une notation abrégée pour toutes les règles obtenues en remplaçant dans la règle initiale chacun de ces groupements par l'une de ses séquences composantes.

exemple : <A> ::= <B> (<B> <C>|<C>) (<D>|<E>)| <B>
est une notation abrégée de
<A> ::= <B>
<A> ::= <B> <B> <C> <D>
<A> ::= <B> <B> <C> <E>
<A> ::= <B> <C> <D>
<A> ::= <B> <C> <E>

b) Le signe * placé en exposant d'un symbole non terminal ou d'un groupement entre parenthèses indique que ce symbole non terminal ou l'une quelconque (pas nécessairement toujours la même) des séquences composantes du groupement peut être répétée un certain nombre, non nul, de fois.

exemple : <BL> ::= <D>* <IM> indique que
<BL> ::= <D> <IM> ou
<BL> ::= <D> <D> <IM> ou
<BL> ::= <D> <D> ... <D> <IM> sont possibles.
de même <A> ::=(<B>|<C>)* indique que <A> est une séquence de <B> ou de <C> mélangés, par exemple,
<A> ::= <B> <C> <B> <B> <C>

c) Le symbole terminal Λ représente la ramification vide.

III.1.3) Grammaire engendrant la ramification donnée

III.1.3.1) programme

<PG> ::= <BL> | <IM> | <I>

III.1.3.2) instructions étiquetées

<I> ::= <DE> (<BL>| <IM> | <ISP> | <IP>|<IC>|<IA>
<IB> | <IC>)

<DE> ::= <ETIQ>*

III.1.3.3) bloc

<BL> ::=(<D>| <DP>| <DA>)*(<I>| <BL>| <IM>| <ISP>
|<IP>| <IC>| <IA>| <IB>|<IO>)

III.1.3.4) instruction composée

<IM> ::= (<I>|<BL>| <IM> |<ISP> | <IP> | <IC> | <IA>
|<IB> | <IO>)*

III - GENERALITES SUR LE COMPILATEUR DECRIT

III.1.3.5) instruction procédure ou indicateur de fonction

<ISP> ::= <ID> (<ISP>|<EC>|<E>|<VAR>|<VFD>
|<CR>|<CE>|<CB>|<CH>)*

III.1.3.6) instruction POUR

<IP> ::= (<VAR>|<VFG>)<LP>(<I>|<BL>|<IM>|<ISP>
<IP> <IC> <IA> <IB> <IO>)*
<LP> ::= (<EL1>|<EL2> <EL3>)*
<EL1>::= (<ISP>|<EC>|<E>|<VAR> |<VFD>|<CR>|<CE>)
<EL2>::= (<ISP>|<EC>|<E>|<VAR>|<VFD>|<CR>|<CE>)
(<ISP>|<EC>|<E>|<VAR>|<VFD>|<CB>)
<EL3>::= (<ISP>|<EC>|<E>|<VAR>|<VFD>|<CR>|<CE>)
(<ISP>|<EC>|<E>|<VAR>|<VFD>|<CR>|<CE>)
(<ISP>|<EC>|<E>|<VAR>|<VFD>|<CR>|<CE>)

III.1.3.7) instruction conditionnelle

<IC> ::= (<ISP>|<EC>|<E>|<VAR>|<VFD>|<CB>)
(<I>|<BL>|<IM>|<ISP>|<IP>|<IC>|<IA>|<IB>|<IO>)
(<I>|<BL>|<IM>|<ISP>|<IP>|<IC>|<IA>|<IB>|<IO>
|Λ)

III.1.3.8) instruction d'affectation

<IA> ::= (<VAR>|<VFG>)(<ISP>|<EC>|<E>|<VAR>|<VFD>
|<CR>|<CE>|<CB>|<CH>)

III.1.3.9) instruction ALLERA

<IB> ::= <AA>|<EC>|<ETIQ>
<ETIQ> ::= <ID>

III.1.3.10) instruction vide

IO ::= Λ

III.1.3.11) expressions

<EC> ::= (<ISP> | <EC> | <E> | <VAR> | <VFD> | <CB>)
(<ISP> | <AA> | <EC> | < E > | < VAR> | < VFD> ! < ETIQ>
| <CR> | <CE> | <CH>)
(<ISP> | <AA> | <EC> | < E> | < VAR> | <VFD> | < ETIQ>
| <CR> | <CE> | <CH>)
<E> ::= (¬ | + | -) (< ISP> | <EC> | <E> | <VAR> | <VFD> | <CR>
| <CE> | <CB>) |
(<ISP> | <EC> | <E> | <VAR> | <VFD> | <CR> | <CE> | <CB>)
(+ | - | * | / | ↑ | ÷ | ≡ | ⊃ | ∨ | ∧ | < | ≤ | = | ≠ | ≥ | >)
(<ISP> | <EC> | <E> | <VAR> | <VFD> | <CR> | < CE> | <CB>)

III.1.3.12) déclarations de procédures

<DP> ::= (REEL | ENTIER | BOOLEEN | INSTR) < ID> <S>*
(<BL> | <IM> | <ISP> | <IP> | <IC> | <IA> | <IB> | <IO>)
<S> ::= <CS> <ID>

III.1.3.13) déclaration et indicateur d'aiguillage

<DA> ::= <ID> (<AA> | <EC> | <ETIQ>)*
<AA> ::= <ID> (< ISP> | <EC> | <E> | < VAR> | < VFD> | < CE> | <CR>)

III.1.3.14) déclaration de tableau et variables

<D> ::= (REEL | ENTIER | BOOLEEN)(< ID> | TABL <ID> <PDB>*)
<PDB> ::= (< ISP> | <EC> | <E> | <VAR> | <VFD> | <CR> | <CE>)
(< ISP> | <EC> | <E> | <VAR> | <VFD> | <CR> | <CE>)

III.1.3.15) variables

<VAR> ::= (< ID> | <ID> (<ISP> | <EC> | <E> | <VAR> | <VFD>
| <CR> | <CE>)*)

III.1.3.16) paramètres formels appelés par nom

<VFG> ::= <ID>
<VFD> ::= <ID>

III.1.4) Annexe : glossaire des abréviations de la grammaire

AA	indicateur d'aiguillage
BL	bloc
CB	valeur logique
CE	entier
CH	chaîne
CR	nombre
CS	code de spécification regroupant le type et la nature de l'appel
D	déclaration de type ou déclaration de tableaux
DA	déclaration d'aiguillage
DE	déclaration d'étiquette
DP	déclaration de procédure
E	expression arithmétique simple ou expression booléenne simple
EC	expression arithmétique ou expression booléenne ou expression de désignation (forme conditionnelle)
EL1	élément d'une liste de pour (du type expression arithmétique)
EL2	élément d'une liste de pour (avec TANTQUE)
EL3	élément d'une liste de pour (avec PAS et JUSQUA)
ETIQ	étiquette
I	instruction étiquetée
IA	instruction d'affectation
IB	instruction allera
IC	instruction conditionnelle
ID	identificateur
IM	instruction composée
IO	instruction vide
IP	instruction pour
ISP	instruction procédure ou indicateur de fonction
LP	liste de pour
PDB	paire de bornes
PG	programme
S	spécification
VAR	variable
VFD	paramètre formel appelé par nom à droite du signe d'affectation
VFG	paramètre formel appelé par nom à gauche du signe d'affectation

III.2 - DEFINITION DU PROGRAMME OBJET

III.2.1) Les instructions

Le langage objet est un langage d'assembleur admettant l'adressage indexé, direct et indirect.

Les instructions d'un programme en ce langage sont normalement exécutées en séquence. Une instruction peut être étiquetée au moyen d'une étiquette symbolique. Chaque instruction admet un certain nombre d'opérandes.

III.2.2) Les opérandes

Un opérande peut être une constante immédiate (entière, réelle ou booléenne), une étiquette symbolique ou une adresse.

L'opérande adresse sera constitué d'un couple d'entiers : l'adresse et le numéro du registre d'index utilisé. Ce couple pourra être précédé du signe d'adressage indirect, indiquant que l'adresse effectuée se trouve à l'adresse indexée décrite par le couple.

Afin d'éviter dans les programmes les séquences d'instructions relatives aux calculs d'adresses les plus courants, nous avons admis deux formes supplémentaires pour les éléments du couple d'un opérande adresse :

- une adresse indexée, directe ou indirecte qui contiendra l'élément ainsi noté. Ceci représente une extension de l'adressage indirect et évite d'expliciter les chargements d'index ;
- une somme d'adresses et d'entiers, ce qui étend quelque peu la notion d'adressage indexé : l'élément décrit sera le résultat de la somme des entiers et des contenus des adresses indiquées.

III.2.3) Les opérations

Les codes instructions caractérisés par la nature de l'opération à effectuer (notée X) et les types des opérandes (notés Y) ne sont pas explicités. Ils seront notés CODE (X, Y). Les opérations utilisées sont indiquées dans le tableau ci-après :

X	Y	nombre d'opérandes	description de l'opération
AFE	ENTIER ou REEL ou BOOLEEN ou CHAINE	2	Les deux opérandes sont de même type. On affecte au second la valeur du premier
TRE	ENTIER	2	On affecte au second opérande la conversion en entier de la valeur réelle du premier
TER	REEL	2	On affecte au second opérande la conversion en réel de la valeur entière du premier
BR	0	1	Il y a branchement inconditionnel à l'adresse symbolique représentée par l'opérande
BR	BOCL	2	Si le premier opérande (booléen) est vrai, il y a branchement à l'adresse symbolique, deuxième opérande
BR	>	3	Si la valeur du premier opérande est supérieure à celle du second, il y a branchement à l'adresse symbolique, troisième opérande
+	REEL ou ENTIER	2	Les deux opérandes sont de même type. On affecte au second la valeur du premier
—	REEL ou ENTIER	2	Les deux opérandes sont de même type. On affecte au second la valeur changée de signe du premier
+ − *	REEL ou ENTIER	3	Les trois opérandes sont de même type. On affecte au troisième la valeur du premier plus (ou moins, ou multiplié par) la valeur du second
/	REEL	3	Les trois opérandes sont réels. Le troisième est le résultat de la division du premier par le second

$\uparrow$	REEL	3	Les trois opérandes sont réels. On affecte au troisième le résultat de l'élévation du premier à la puissance représentée par le second, avec la définition usuelle des puissances
$\frac{\circ}{\circ}$	ENTIER	3	Les trois opérandes sont entiers. Le troisième contiendra l'entier le plus proche du résultat de la division du premier par le second
$\equiv$ $\supset$ $\vee$ $\wedge$	BOOL	3	Les trois opérandes sont booléens. On affecte au troisième le résultat de la composition du premier par le second, selon les opérations traditionnelles de l'algèbre de Boole : équivalence, implication, réunion, intersection
$\neg$	BOOL	2	Les deux opérandes sont booléens. On affecte au second la négation du premier
$<$ $\leqslant$ $=$ $\neq$ $\geqslant$ $>$	ENTIER ou REEL	3	Les deux premiers opérandes sont du même type : réel ou entier, le troisième est booléen. On lui affectera la valeur VRAI ou FAUX selon que la relation exprimée entre le premier et le second opérande aura été ou non satisfaite. Les opérateurs désignent les relations habituelles d'infériorité stricte, infériorité large, égalité, différence, supériorité large, supériorité stricte
SP	0	N	C'est le code d'appel d'un sous-programme. Le premier opérande est un entier identifiant le sous-programme, les autres sont les paramètres à transmettre. Il faudra indiquer, à chaque appel, l'adresse de retour dans le programme. Les sous-programmes utilisés par le compilateur sont décrits en annexe du chapitre IV.

III.2.4) Le programme objet

Ce sera une ramification engendrée par la grammaire suivante (en notation de Backus étendue - cf. III.1.2)

<programme> ::= <ordre>*
<ordre> ::= (IT | ∧) <code instruction> (< GA> | <IT> | <CI>)*
<GA> ::= (≠ | ∧) < entier> < entier> |
(≠ | ∧) (< VA > | <entier>) (< VA> | <entier>)
<VA> ::= <GA1> | (< GA1> | < entier>) (⊞ (< GA1> | < entier>))*
<GA1> ::= (≠ | ∧) < entier> < entier>
<IT> ::= <entier>
<CI> ::= <entier> | <entier> . < entier> | VRAI | FAUX

IT représente une étiquette symbolique, CI une constante immédiate et GA un opérande adresse. Nous n'avons pas explicité l'écriture d'un entier.

III.3 - GESTION DE LA MEMOIRE

III.3.1) Introduction

Nous avons supposé que l'ordinateur utilisé pour l'exécution du programme compilé possédait un nombre suffisant de registres d'index, numérotés de 0 à N. La solution adoptée pour la gestion de la mémoire, ainsi que la politique de compilation des appels de procédures qui en découle, resteraient applicables à tout langage à structure de blocs et admettant des procédures récursives.

III.3.2) Cas d'un programme sans procédures

III.3.2.1) constantes et instructions

Les constantes du programme ont été regroupées dans une zone virtuelle adressée à partir de zéro et indexée par le registre zéro. Il suffira que le chargeur garnisse le registre zéro, de l'adresse de début de la zone réelle, avant de la garnir des codes relatifs aux constantes. Cette zone est donc totalement indépendante du reste.

Une phase d'assemblage déterminera les adresses réelles des instructions du programme objet, le compilateur n'ayant prévu que des adresses symboliques. Le programme objet occupera donc une zone de mémoire indépendante de celle des constantes et de celle des variables.

III.3.2.2) bornes de la mémoire gérée

La zone affectée aux variables fait seule l'objet d'une réelle gestion. Les réservations sont effectuées dans une zone virtuelle où les adresses vont de 0 à A2. En fait, le début de la zone des variables sera indexé par le registre un, la fin par le registre deux, ce qui permettra d'avoir une zone réelle de variables allant de p à q, quels que soient les entiers p et q.

III.3.2.3) variables

Une variable ne requérant un emplacement en mémoire que pendant l'exécution du bloc où elle est déclarée, la zone des variables sera organisée en pile.

Cela veut dire qu'à chaque entrée dans un bloc, on emploiera, pour les déclarations de ce bloc, les premiers éléments de mémoire encore inutilisés, qu'on les libérera à la sortie du bloc, et que pour deux blocs disjoints contenus immédiatement tous deux, dans un même troisième, les premières mémoires affectées aux déclarations de l'un coïncideront avec celles affectées aux déclarations de l'autre.

exemple : Début réel X ; Début réel Y, Z ; ... fin ;
Début réel A ; ... fin ; ... fin ;

L'emplacement mémoire alloué à A sera le même que celui alloué à Y. Cet emplacement suit immédiatement en mémoire celui alloué à X, et précède celui alloué à Z.

III.3.2.4) réservations statiques

Les informations de longueur connue à la compilation (variables simples, renseignements relatifs aux tableaux) seront donc organisées en pile ascendante à partir de l'adresse 0 indexée par le registre 1, dans ce que l'on appellera la zone statique.

Comme on ne connaît pas à la compilation la longueur d'un tableau, on se contentera de réserver en zone statique un vecteur renseignements, et de générer des instructions qui, à l'entrée dans le bloc, garniront ce vecteur des valeurs convenables. Ce vecteur contiendra :

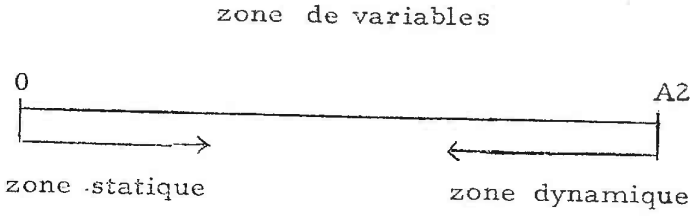
- l'adresse initiale en mémoire du tableau, considéré comme la suite ordonnée de ses éléments. Cette adresse sera calculée à partir de l'adresse initiale et de la longueur du tableau qui le précède dans l'ordre d'exécution des déclarations ;
- le nombre d'indices du tableau ;
- le type du tableau ;
- pour chaque indice, la valeur de la borne inférieure de l'indice et le nombre de valeurs possibles pour cet indice ;

Cela permet de calculer l'adresse de n'importe quel élément du tableau.

III.3.2.5) réservations dynamiques

Les éléments des tableaux, seront, eux, rangés à l'autre bout de la zone des variables, dans une pile descendante que nous appellerons zone dynamique.

Le premier tableau déclaré sera implanté à partir de l'adresse α , de façon que son dernier élément occupe l'adresse A2. Le second tableau déclaré, dans l'ordre d'exécution des déclarations, sera implanté à partir de l'adresse β de façon que son dernier élément occupe l'adresse $\alpha - 1$, etc...



III.3.2.6) résultats intermédiaires

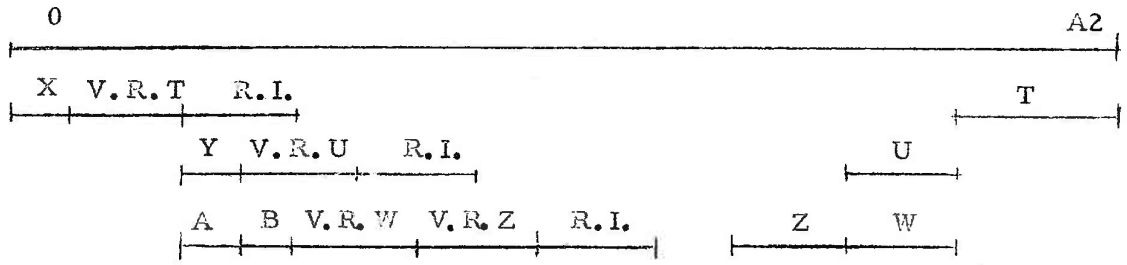
On prendra pour résultats intermédiaires de chaque bloc, les emplacements nécessaires à partir de la fin de la zone allouée à ses réservations statiques.

exemple : Début réel;X ; X := 2 * A + B * * 2 ;
Début réel Y ; ... fin ; ... fin ;

Pour calculer l'expression 2 * A + B * * 2, on aura besoin de deux emplacements pour les résultats intermédiaires de 2 * A et B * * 2 puis, pour leur somme, le premier de ces emplacements mémoire coïncidera avec celui alloué à Y immédiatement après.

exemple : Début Réel X ; Tableau T [1 : N] ; ... ;
Début Réel Y ; Tableau U [1 : X, 1 : X]; ... fin ;
Début Réel A, B; Tableau W, Z[-10 : X]; ... fin ;
fin ;

La zone des variables sera organisée selon le schéma suivant, où V.R. indique un vecteur renseignement et R.I. des résultats intermédiaires :



III.3.3) Cas d'un programme avec procédures

III.3.3.1) généralités

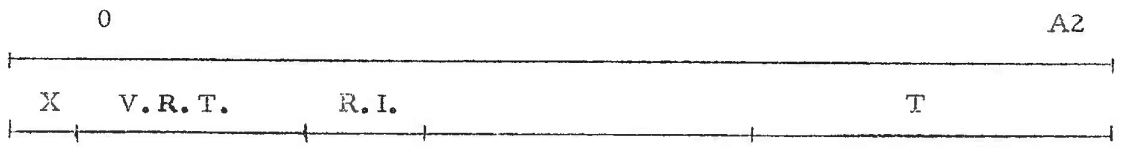
Les variables internes à chaque procédure n'existent que pendant l'appel de la procédure. Nous avons effectué, pour chaque procédure, l'adressage comme pour un programme indépendant. Il suffit alors de définir les bornes de la zone de variables au moment de l'appel, et, pour cela, de garnir les index de début et fin de cette zone.

Nous prendrons pour borne inférieure, l'adresse du premier élément de la zone statique, libre après les résultats intermédiaires utilisés au moment de l'appel.

Nous prendrons pour borne supérieure, l'adresse précédant l'adresse initiale du dernier tableau déclaré.

exemple : Début Réel X ; Tableau T [1 : 10]; Procédure F(A) ;
... ;
X := (X + 3) 2 / F (X) ; ... fin ;

zone de variables du programme



zone de variables de la procédure F lors de son appel dans l'exemple ci-dessus.

remarque : l'éxécution du programme est considérée comme son appel.

III. 3. 3. 2) niveau d'une procédure

définition :

Nous disons d'une procédure qu'elle est de niveau n si elle est déclarée dans le corps d'une procédure de niveau n - 1 ; le programme sera considéré comme une procédure de niveau 1.

Nous associerons à chaque niveau un couple de registres d'index pour définir les bornes de la zone de variables, à l'appel d'une procédure de ce niveau. Les variables internes aux procédures de ce niveau seront indexées au moyen de ces registres. On pourra, de plus, utiliser dans le corps d'une procédure, des variables globales, indexées par les registres relatifs au niveau de la procédure contenant leur déclaration. Nous utiliserons donc, simultanément, des adresses indexées par plusieurs registres d'index distincts.

remarque :

Cette gestion de la mémoire aurait pu être réalisée en n'utilisant qu'un registre par niveau, celui relatif à la borne inférieure de la zone de variables. La borne supérieure serait alors conservée dans une mémoire de la zone statique. Les différences essentielles apparaîtraient lors du maniement des termes de la zone dynamique, pour le calcul de l'adresse d'une variable indicée par exemple.

III. 3. 3. 3) gestion des index

Il faut générer, à la compilation d'un appel, les instructions de garnissage des index associés à son niveau. Les valeurs à charger sont connues à la compilation : ce sont des adresses indexées par les registres associés à la procédure en cours lors de l'appel.

Il ne faut cependant pas risquer de perdre les valeurs de certains couples d'index qui devront être restaurés, au retour, après l'appel. Nous étudierons successivement les différents cas relatifs à l'appel d'une procédure de niveau k, intervenant dans le corps d'une procédure de niveau l.

III.3.3.4) registres des niveaux supérieurs ou égaux à k

a) Si k est strictement supérieur à 1, les registres associés aux niveaux k, k + 1, k + 2, etc... ne sont pas encore concernés et ne font l'objet d'aucun traitement.

b) Si k est inférieur ou égal à 1 (cas des appels récursifs, par exemple), il faudra, avant le saut vers le corps de procédure, ranger les registres associés aux niveaux k, k + 1, k + 2, ..., L, actuellement en cours d'utilisation et pouvant être réutilisés après l'appel.

Ces valeurs seront rangées dans les premières mémoires libres de la zone statique : on devra générer les instructions de rangement.

On pourra ensuite garnir des nouvelles bornes de la zone des variables, les registres associés au niveau k.

On génèrera également, après l'appel, des instructions de restauration des registres.

III.3.3.5) registres des niveaux strictement inférieurs à k

a) La procédure appelée n'est pas un paramètre formel : on a accès lors de l'appel à la déclaration de la procédure et à tous les identificateurs, même non internes, utilisés dans son corps. Les registres associés aux niveaux strictement inférieurs à k ne sont donc pas changés par l'appel.

b) La procédure F appelée est un paramètre formel de la procédure H (de niveau h) qui contient la procédure L où se trouve l'appel :

```
Procédure F (...) ...
.....................
.....................

H (F, ...)
.....................
```

```
Procédure H (X, ...)
.....................
Procédure L (...)
.....................
  X (...)
.....................
.....................
```

Un cas analogue se présente pour les paramètres formels appelés par nom et pour lesquels le paramètre effectif est une expression : le paramètre effectif est alors considéré comme une procédure.

Nous avons admis que les variables valables à l'appel de F ainsi défini, étaient celles déterminées juste avant l'appel H (F, ...)

Il manque pour justifier ce choix une théorie de la sémantique des appels de procédure en Algol 60. Le rapport Algol 60 permet d'autres interprétations. Notre option semble coïncider au mieux avec la théorie des recopies.

Comme depuis l'appel H (F, ...) jusqu'à l'appel formel de F les registres des niveaux supérieurs ou égaux à h ont pu être modifiés, il faut, dans le cas $k > h$, rétablir, avant l'appel formel de F, les registres des niveaux h, h + 1, ..., k - 1 dans leurs valeurs précédant l'appel de H, et conserver leurs valeurs actuelles pour les rétablir après l'appel de F.

Cela implique que l'on associe à chaque paramètre formel procédure, les registres des niveaux h, h + 1, ..., k - 1. Le nombre de ces valeurs variant avec le paramètre effectif, nous les rangerons en zone dynamique, et ne conserverons en zone statique que l'indication de leur adresse et de leur nombre.

III.4 - COMPILATION DES PROCEDURES

III.4.1) Vérification des types

Afin de permettre les vérifications relatives aux paramètres d'une procédure lors d'un appel et de simplifier les traitements des appels formels, nous avons associé à chaque déclaration de procédure un vecteur de renseignements contenant :

- l'étiquette de début du corps de procédure
- le niveau de la procédure
- le type de la procédure
- le nombre de paramètres
- par paramètre, l'indication d'appel, par valeur ou par nom, et son type.

Nous supposerons que chacun de ces renseignements occupe un mot de mémoire. Ce vecteur, comme celui d'un tableau, figure parmi les réservations statiques du bloc, et nous génèrerons des instructions de garnissage.

III.4.2) Zone statique lors d'un appel

Les deux premiers mots de mémoire libres en zone statique, lors d'un appel, seront réservés pour la transmission de valeurs entre les sous-programmes de la séquence remplaçant l'appel. Les index relatifs aux niveaux k, k + 1, ... L (cf. III.3.3.4 b) seront rangés dans les mots suivants.

III.4.3) Zone statique d'une procédure

Les k premiers mots de mémoire de la zone statique d'une procédure contiennent des renseignements relatifs à la procédure et à ses paramètres. Ces renseignements facilitant la communication entre la procédure appelée et celle où se trouve l'appel, l'appel des sous-programmes d'évaluation des paramètres formels appelés par nom, et l'exécution des appels formels.

Ces premiers mots de la zone statique d'une procédure sont garnis pendant l'exécution de la séquence d'appel de la procédure. Ils contiendront respectivement :

a) renseignements relatifs à la procédure :

- l'éventuel résultat de la procédure... (2 mots)
- l'adresse de retour après l'appel.... (1 mot)
- l'adresse de début du corps de procédure..... (1 mot)
- le niveau de la procédure..... (1 mot)
- la différence k - L (cf. III.3.3.4 et III.3.3.5).... (1 mot)

et par paramètre formel, les

b) renseignements relatifs à un paramètre formel :

- un indicatif d'expression pour le paramètre effectif..... (1 mot)
- le résultat de l'évaluation, si c'est un paramètre appelé par valeur ; l'adresse et le nombre d'index rangés en zone dynamique, si c'est une procédure..... (2 mots)

- l'adresse du paramètre effectif..... (1 mot)
- l'index du paramètre effectif..... (1 mot)

  Ces deux renseignements sont relatifs au cas d'un paramètre formel appelé par nom, le paramètre effectif n'étant pas une expression, ou d'un paramètre formel procédure dont on repère ainsi le vecteur renseignements, ou d'un paramètre formel étiquette

- l'étiquette de début du paramètre effectif, dans le cas d'un paramètre formel appelé par nom, le paramètre effectif étant une expression..... (1 mot)
- l'adresse de retour, dans le corps de procédure, après l'évaluation d'un paramètre effectif expression..... (1 mot)

Les détails de la compilation des déclarations et appels de procédures sont exposés au chapitre IV.

IV - DESCRIPTION FORMELLE DU COMPILATEUR

Nous exposerons, dans ce chapitre, les cinq transformations de ramifications intervenant dans un compilateur, et écrirons parallèlement le programme de description du compilateur défini au chapitre III.

Chaque transformation fera l'objet d'une introduction explicative. Les détails de chaque fonction la composant seront précisés en regard de la déclaration de la fonction.

Des schémas seront également fournis pour faciliter la compréhension des fonctions.

IV.1 - TRANSFORMATION PRELIMINAIRE DE LA RAMIFICATION DONNEE

IV.1.1) Constantes

Un premier résultat de la compilation d'un programme est une table contenant une seule fois chacune des constantes utilisées par le programme, et, en regard, l'adresse où sera rangée son code. Les adresses seront comptées à partir de zéro et indexées par le registre zéro. Le programme I décrit la construction de cette table. Elle sera fournie au chargeur qui garnira les adresses réservées, des codes appropriés. (L'explication des fonctions du programme I est donnée en regard de leurs déclarations dans le programme de description du compilateur).

Nous remplacerons ensuite les constantes par leurs adresses, dans la ramification du programme.

IV.1.2) Instructions

Le deuxième but de cette transformation est la simplification des instructions syntaxiquement complexes du langage source.

Pour Algol 60, nous remplacerons les instructions POUR par des séquences ne comportant plus que des instructions conditionnelles, les instructions conditionnelles par des branchements conditionnels et les déclarations d'aiguillages par des déclarations de procédures.

Nous déborderons légèrement le cadre d'Algol 60 en utilisant dans la transformation des instructions POUR, une variable à valeur étiquette, c'est-à-dire une adresse indirecte en langage machine. Il en sera de même pour les procédures remplaçant les déclarations d'aiguillages : elles auront pour résultat une étiquette.

Nous nous servirons de la structure de bloc d'Algol 60 pour simplifier la génération de nouvelles variables et étiquettes dans cette première transformation.

IV.1.3) Traitement de l'instruction conditionnelle

a) l'instruction est du type SI e ALORS I1 ; nous génèrerons la pseudo-arborescence équivalente à la séquence :

DEBUT SI NON e ALORS ALLERA ETA (1) ; I1 ; ETA (1) : FIN ;

b) l'instruction est du type SI e ALORS I1 SINON I2 ; nous génèrerons la pseudo-arborescence équivalente à la séquence :

DEBUT SI NON e ALORS ALLERA ETA(1) ; I2 ; ALLERA ETA(2) ;
ETA(1) : I1 ; ETA(2) : FIN ;

Ces séquences sont considérées comme des blocs, la fonction ETA, qui fournit les étiquettes, est définie en II.2.4.20.

IV.1.4) Traitement de la déclaration d'aiguillage

L'instruction AIGUILLAGE A : E1, E2, E3 ; sera transformée en :

ENTIER PROCEDURE A(I) ; VALEUR I ; ENTIER I ; SI I = 1
ALORS A := E1 SINON SI I = 2 ALORS A := E2
SINON SI I = 3 ALORS A := E3 ;

Nous transformerons ensuite les instructions conditionnelles de ce corps de procédure selon IV.1.3.

IV.1.5) Traitement de l'instruction POUR

Nous remplacerons chaque instruction POUR X:= liste de pour FAIRE I ; par :

DEBUT ENTIER ETA (1) ;

Traitement de la liste de pour

ALLERA ETA(3) ; ETA (2) : I ; ALLERA INDIRECT ETA (1) ;
ETA (3) : FIN ;

Le traitement de la liste de pour se compose de la séquence des traitements des éléments de la liste de pour, c'est-à-dire :

- si l'élément est une expression arithmétique E :
  ETA(1) := ETA(A) ; X:= E ; ALLERA ETA (2) ; ETA(A) :;
- si l'élément est de la forme E TANTQUE EB :
  ETA(1) := ETA(A) ; ETA(A) : X := E ; SI EB ALORS
  ALLERA ETA (2) ;
- si l'élément est de la forme E1 PAS E2 JUSQUA E3 :
  ETA(1) := ETA(A) ; X:= E1 ; ALLERA ETA (A + 1) ;
  ETA(A) : X := X + E2 ;
  ETA(A + 1) : SI (X - E3) * SIGN (E2) $\geqslant$ 0 ALORS
  ALLERA ETA (2) ;

IV.1.6) Remarque

Vu la similitude, au niveau machine, d'une expression arithmétique traduite et d'une instruction d'affectation, nous aurions également pu traiter, dans cette partie, le cas des expressions conditionnelles. L'étude de ce problème a montré que les complications introduites dépassaient le bénéfice obtenu et empiètaient sur le traitement des résultats intermédiaires.

Ce cas n'a donc pas été exposé ici.

PROGRAMME I

```
DEBUT
FONCTION CT(A) REC(A)
$;
CT(R) + CT(S);
T='CR' OU T='CE' OU T='CB' OU T='CH' : T*U, CT(U);

FONCTION CTI(A,B)
NR(A)>1 : (CT2(CMI(A), CAMI(A))≠$ : CTI(CAMI(A), B),
          CTI(CAMI(A), B⊕CT3(CMI(A))) + 'C'*('GA'*(B+0)+CMI(A))),
'C'*('GA'*(B+0)+A);

FONCTION CT2(A,B)
NR(B)=1 : (A=B : A, $), A=CMI(B):A, CT2(A, CAMI(B));

FONCTION CT3(A)
RO(A)='CR' : 2, RO(A)='CE' OU RO(A)='CB' : 1, G(A);

CTI(CT(W), 0)

FIN
```

La fonction CT construit, à partir de son argument, la ramification composée de la suite des constantes contenues dans son argument.

La fonction CT1 supprime, dans cette suite (représentée par son premier argument), les occurences multiples d'une constante et procède à l'adressage des constantes à partir de son deuxième argument. Il sera pris initialement égal à zéro.

Dans le cas où la ramification des constantes comporte plus d'un élément, on compare le dernier à tous ceux qui le précèdent, au moyen de la fonction CT2.

Le résultat de CT2 n'est $ que dans le cas où la pseudo-arborescence premier argument n'est égale à aucune des composantes connexes de la ramification deuxième argument.

La fonction CT3 calcule le nombre de mots de mémoire occupés par son argument : la pseudo-arborescence représentant une constante. Le cas des chaînes n'est pas explicité ; nous supposons que la fonction G rendra un nombre convenable.

La fonction CT4 a pour argument une constante A et la table B construite par le programme I. Elle fournit l'adresse associée à la constante, ou $, si la constante n'existe pas dans la table. La comparaison de A avec les constantes de la table s'effectue, séquentiellement, de droite à gauche.

La fonction TC a pour arguments la table construite par le programme I = A, et la ramification du programme à compiler = B. Elle y remplace les occurences des constantes par leurs adresses et types.

PROGRAMME DE DESCRIPTION DU COMPILATEUR

```
DEBUT

FONCTION  CT(A)  REC(A)
$;
CT(R) + CT(S);
T='CR' ou T='CE' ou T='CB' ou T='CH' : T* 4 , CT(U);

FONCTION  CTI(A,B)
NR(A)>1 : (CT2(CMI(A), CAMI(A)) ≠ $ : CTI(CAMI(A), B),
           CTI(CAMI(A), B⊕ CT3(CMI(A))) + 'C'* ('GA'* (B+0) + CMI(A))),
'C'* ('GA'* (B+0) + A);

FONCTION  CT2(A,B)
NR(B)=1 : (A=B : A, $), A= CMI(B); A, CT2(A, CAMI(B));

FONCTION  CT3(A)
RO(A)='CR' : 2, RO(A) = 'CE' ou RO(A) = 'CB' : 1, 6(A);

FONCTION  TC(A,B)  REC(B)
$;
TC(A,R) + TC(A,S);
(T='CR' ou T='CE' ou T='CB' ou T='CH') ET (CT4(U,A) ≠ $) :
'VAR'* (CT4(U,A) + (T='CR' : 'REEL', T='CE' : 'ENTIER', T='CB' : 'BOOL', 'CHAINE')),
T* TC(A,U);

FONCTION  CT4(A,B)
NR(B)=1 : (SR(CSAI(B)) = A : CI(SR(B)), $),
          SCMI(SCMI(B)) = A : CI(SCMI(B)), CT4(A, CAMI(B));
```

La fonction P est appliquée à la pseudo-arborescence du programme où l'on a remplacé les constantes par les adresses et types correspondants.

Elle transforme les instructions conditionnelles (IC) d'après 1.3.a et 1.3.b.

Elle remplace les instructions POUR par les pseudo-arborescences correspondant à la séquence décrite en 1.5, en appelant la fonction P1 pour construire la ramification résultat du traitement de la liste de pour.

Elle transforme les déclarations d'aiguillage, en faisant construire, par la fonction FDA, le corps de procédure décrit en 1.4. Ce corps de procédure est ensuite transformé par P et assemblé avec la tête de procédure pour former la déclaration de procédure qui remplacera la déclaration d'aiguillage.

IOO est une instruction vide qui sera supprimée à la génération.

Le rôle de la fonction P1 est de faire progresser l'entier (son premier argument) qui sera utilisé comme argument de la fonction ETA, pour former des étiquettes différentes pour chaque élément de la liste de pour.

Son deuxième argument est la liste de pour, son troisième argument, la variable contrôlée du pour. Ayant défini l'entier A, elle appelle la fonction P2.

Les trois arguments de P2 sont respectivement les mêmes que les trois arguments de P1. Le rôle de cette fonction est de construire la ramification correspondant à un élément de la liste de pour. (Cf. 1.5).

La fonction FDA construit le corps de procédure défini en 1.4. Son premier argument est l'identificateur de l'aiguillage, son deuxième argument, la liste d'aiguillage, et son troisième argument, un entier qui numérote les identificateurs de la liste d'aiguillage.

La fonction PREL décrit l'ensemble de la transformation préliminaire de la ramification donnée.

```
FONCTION  P(A)  REC(A)
$;
P(R) + P(S);
T='ic':(C3(P(U))=$:'BL'*('ic'*('e'*('7'+C1(P(U)))+'ETIQ'*ETA(1))+C2(P(U))+
                         'j'*('DE'*'ETIQ'*ETA(1)+'100')),
        'BL'*('ic'*('e'*('7'+C1(P(U)))+'ETIQ'*ETA(1))+C2(P(U))+'iB'*'ETIQ'*ETA(2)+
              'j'*('DE'*'ETIQ'*ETA(1)+C3(P(U)))+'i'*('DE'*'ETIQ'*ETA(2)+'100'))),
T='iP':'BL'*('D'*('ENTIER'+ETA(1))+P1(4,SC2(P(U)),C1(P(U)))+'iB'*'ETIQ'*ETA(3)+
             'j'*('DE'*'ETIQ'*ETA(2)+C3(P(U)))+'iB'*('IND'+'VAR'*ETA(1))+
             'i'*('DE'*'ETIQ'*ETA(3)+'100')),
T='DA':'DP'*('ENTIER'+C1(P(U))+'S'*('VE'+ETA(0))+P(FDA(C1(P(U)),CAI(P(U)),1))),
T='AA':'ISP'*P(U),
T*P(U);

FONCTION  P1(A,B,C)
NR(B)=1:P2(A,B,C),
P1(A⊕(RO(B)='EL1' OU RO(B)='EL2':1,2),CAMI(B),C)+P2(A,CMI(B),C);

FONCTION  P2(A,B,C)
RO(B)='EL1':'iA'*('VAR'*ETA(1)+'ETIQ'*ETA(A))+'iA'*(C+SR(B))+'iB'*'ETIQ'*ETA(2)+
            'j'*('DE'*'ETIQ'*ETA(A)+'100'),
RO(B)='EL2':'iA'*('VAR'*ETA(1)+'ETIQ'*ETA(A))+'i'*('DE'*'ETIQ'*ETA(A)+'iA'*(C+CSI(B)))+
            'ic'*(CS2(B)+'ETIQ'*ETA(2)),
'iA'*('VAR'*ETA(1)+'ETIQ'*ETA(A))+'iA'*(C+CSI(B))+'ETIQ'*ETA(A⊕1)+
'i'*('DE'*'ETIQ'*ETA(A)+'iA'*(C+'E'*(C+'+'+CS2(B))))+'i'*('DE'*'ETIQ'*ETA(A⊕1)+
'ic'*('E'*('E'*('E'*(C+'-'+CS3(B))+'*'+'ISP'*('SIGN'+CS2(B)))+'≥'+'ci'*0)+
'ETIQ'*ETA(2)));

FONCTION  FDA(A,B,C)
NR(B)=1:'ic'*('E'*('VAR'*ETA(0)+'='+C)+'iA'*('VAR'*A+B)),
'ic'*('E'*('VAR'*ETA(0)+'='+C)+'iA'*('VAR'*A+C1(B))+FDA(A,CAI(B),C⊕1));

FONCTION  PREL(A)
P(TC(CTI(CT(A),0),A));
```

IV.2 - ETUDE LEXICOGRAPHIQUE

IV.2.1) Déclarations de variables et de tableaux

Nous associons, à chaque déclaration de variable ou de tableau, un groupe adresse (adresse et registre d'index) indiquant le début de la zone mémoire allouée à la variable ou au vecteur de renseignements du tableau.

Dans ce dernier cas, nous génèrerons également la ramification des instructions de garnissage du vecteur de renseignements et le contrôle des débordements de mémoire.

IV.2.2) Déclarations de procédures - appels de procédures

Nous procédons à l'étude lexicographique du corps de procédure en comptant les adresses à partir de k, et en utilisant les registres d'index relatifs au niveau de la procédure. (Cf. III.4.3).

Nous génèrerons également la ramification des instructions de garnissage du vecteur de renseignements de la procédure.

Nous joignons, à chaque appel de procédure, le niveau de la procédure le contenant, pour prévoir les éventuels rangements d'index, et la première adresse libre en zone dynamique, pour définir le début de la zone dynamique de l'appel.

IV.2.3) Déclarations d'étiquettes

Nous avons appelé déclaration d'étiquettes, l'apparition d'une ou plusieurs étiquettes à gauche d'une instruction. Nous associons, à chaque déclaration d'étiquettes, une étiquette symbolique : un entier enraciné par "IT", en supprimant la structure de bloc pour les étiquettes, et l'étiquetage multiple d'une instruction.

IV.2.4) Adjonction de nouvelles étiquettes

En prévision de la génération du programme objet, nous avons associé :

- deux étiquettes à chaque expression conditionnelle
- une étiquette par variable indicée ou contrôle de débordemant, pour être l'étiquette de retour après le calcul d'adresse ou le contrôle
- deux étiquettes par déclaration de procédure : une pour indiquer le début du corps de procédure et l'autre pour pouvoir l'ignorer
- une ou deux étiquettes par paramètre formel appelé par nom pour indiquer le retour après l'évaluation
- un nombre d'étiquettes, fonction du nombre de paramètres, pour chaque appel de procédure, en prévision de la génération de la séquence d'appel

Ces étiquettes répondent aux critères de 2.3.

La ronction d'adressage : A possède cinq paramètres :

- B est la ramification à étudier
- C désigne la première adresse libre à affecter aux réservations de la ramification traitée
- D est le groupe adresse de début du vecteur de renseignements du dernier tableau déclaré. Il est initialement pris égal à 'GA' * (- 1 + E)
- E est le registre d'index utilisé pour la procédure. Il correspond à 2 * (niveau de la procédure - 1)
- F est le premier entier non encore utilisé comme étiquette symbolique.

La fonction A exprime que, pour une ramification R + S, on effectue d'abord l'adressage pour S, puis on fait progresser les entiers C et F ; on modifie D, si S est une déclaration de tableau, et l'on procède à l'adressage de R.

Elle associe deux étiquettes à chaque expression conditionnelle, une à chaque déclaration d'étiquettes.

Pour chaque bloc, comprenant au moins une déclaration, (ce qui n'est pas vérifié pour tous les blocs apparus lors des préliminaires), elle associe un contrôle de débordement mémoire (SP1) et la première adresse libre en zone dynamique. Le sommet de la zone statique sera associée lors du traitement des résultats intermédiaires.

Elle associe les adresses aux déclarations de variables, de tableaux et de procédures. De plus, pour les déclarations de procédures, elle regroupe, sous DPP, les renseignements propres à l'identificateur de la procédure.

Elle joint enfin les étiquettes aux variables indicées et aux paramètres formels appelés par nom, le niveau de la procédure contenant l'appel et la première adresse libre en zone dynamique aux appels de procédure.

La fonction ADR décrit l'étude lexicographique de sa ramification argument.

La fonction TO calcule le nombre de mots de mémoire réservés pour son argument.

```
FONCTION A(B,C,D,E,F) REC (B)
§;
A(R, C⊕TO(S), (CS2(S)='TABL':'GA'*(C+E),D), E, F⊕EO(S))+A(S,C,D,E,F);
T='EC': T*(A(U,C,D,E,F)+'IT'*F+'IT'*(F⊕1)),
T='DE': T*('IT'*F+A(U,C,D,E,F)),
T='BL' ET RO(CI(U))='D': T*('ISP'*(D+'IT'*F)+'GA'*(CI(SCI(SCI(A(U,C,D,E,F⊕1)))))
                                    ⊕TO(CI(U))+E)+A(U,C,D,E,F⊕1)),
T='D':(C2(U)='TABL': T*GV(A(U,C,D,E,F),C,D,E,F),
        T*('GA'*(C+E)+A(U,C,D,E,F))),
T='DP': GR(U,C,D,E,F⊕EO(T*U))+'DP'*('GA'*(C+E)+CI(U)+
        'DPP'*(E⊕2+CI(U)+C2(U))+SI(CAMI(CA2(U)),1,E⊕2)+
        A(CMI(U),TO('DP'*U),'GA'*(⊖1+E⊕2),F⊕2)+'IT'*F+'IT'*(F⊕1)),
T='VAR' ET NR(U)>1: T*(A(U,C,D,E,F⊕1)+'IT'*F),
T='VFG': T*(A(U,C,D,E,F)+'IT'*F),
T='VFD': T*(A(U,C,D,E,F)+'IT'*F+'IT'*(F⊕1)),
T='ISP': T*(E+D+EF(A(U,C,D,E,F),F)),
T*A(U,C,D,E,F);

FONCTION ADR(X)
A(X,0,'GA'*(⊖1+1),1,1);

FONCTION TO(X)
RO(X)='DP': 6⊕7⊛NR(CA3(SR(X))),
RO(X)='D':(CS2(X)='TABL':3⊕2⊛NR(CA3(SR(X))),CS2(X)='REEL':2,1),
0;
```

La fonction EO calcule le nombre d'étiquettes symboliques nécessaires pour l'étude lexicographique de sa ramification argument.

La fonction GV génère la ramification des instructions de garnissage du vecteur de renseignements d'un tableau. Son premier argument est la déclaration du tableau ; ses quatre arguments suivants sont respectivement identiques aux quatre derniers arguments de A. Elle fait précéder les instructions de garnissage, du groupe adresse du vecteur de renseignements, du type et de l'identificateur du tableau déclaré.

La première instruction garnit le premier élément du vecteur de renseignements du nombre de paires de bornes du tableau.

La deuxième instruction est relative au code de type ;

l'avant-dernière au calcul et rangement de l'adresse initiale ;

la dernière est un contrôle de débordement de mémoire. Les instructions intermédiaires sont générées par GV2. A2 est une constante de l'ordinateur utilisée pour l'exécution : la dimension allouée à la zone des variables.

La fonction GV2 a quatre arguments : le premier est la ramification de la liste des paires de bornes ; les second, troisième et quatrième sont respectivement identiques aux second, quatrième et cinquième argument de la fonction A. Son rôle est de faire progresser les compteurs d'adresses et d'étiquettes et de générer l'expression de ramification représentée par la fonction GV3, pour chaque paire de bornes.

Cette fonction a les mêmes arguments que GV2, le premier ne portant plus que sur une paire de bornes. Elle calcule et range la première valeur de l'indice, le nombre de valeurs possibles de cet indice, et met en place un contrôle de validité de l'intervalle des valeurs de l'indice.

La fonction GV1 construit la pseudo-arborescence de calcul du nombre d'éléments du tableau déclaré. Ses arguments sont :
- le nombre d'indices du tableau,
- l'adresse contenant le nombre de valeurs possibles du premier indice,
- le registre d'index utilisé.

```
FONCTION  EO(A)  REC (A)
0;
EO(R) ⊕ EO(S);
T='DE' OU T='BL' OU (T='VAR' ET NR(U)>1) OU T='VFG': EO(U)⊕1,
T='EC' OU T='DP' OU T='VFD' : EO(U)⊕2,
T='D' ET C2(U)='TABL' : EO(U)⊕1⊕ NR(CA3(U)),
T='ISP': EO(U)⊕7⊕4⊗ NR(U),
EO(U);

FONCTION  GV (U, C, D, E, F)
'GA'* (C+E) + C1(U) + C3(U) +
'IA'* ('VAR'* ('GA'* (C⊕1+E)+'ENTIER') + 'Ci'* NR(CA3(U))) +
'IA'* ('VAR'* ('GA'* (C⊕2+E)+'ENTIER') + 'Ci'* (C1(U)='REEL':2, 1) +
GV2 (CA3(U), C⊕3, E, F⊕1) +
'IA'* ('VAR'* ('GA'* (C+E)+'ENTIER')+'E'* ('VAR'* ((CS1(D)=0!:'GA'* (A2+CS2(D)), D)
      +'ENTIER')+'-'+(C1(U)='REEL':'E'* ('Ci'*2+'*'+ GV1(NR(CA3(U)), C⊕4, E)),
                        GV1(NR(CA3(U)), C⊕4, E)))) +
'SPi'* ('GA'* (C+E)+'IT'* F);

FONCTION  GV2 (U, C, E, F)
NR(U)=1: GV3 (U, C, E, F),
GV2(CAM1(U), C⊕2, E, F⊕1) + GV3(CM1(U), C, E, F);

FONCTION  GV3 (A, B, C, D)
RO(A)* ('IA'* ('VAR'* ('GA'* (B+C)+'ENTIER')+'E'* ('CNV'+CS1(A)))+
        'IA'* ('VAR'* ('GA'* (B⊕1+C)+'ENTIER')+'E'* ('E'* ('CNV'+CS2(A)) +
              '-'+'E'* ('VAR'* ('GA'* (B+C)+'ENTIER')+'-'+'Ci'*1))) +
        'SP2'* ('GA'* (B⊕1+C)+'IT'* D));

FONCTION  GV1 (B, C, E)
B=1: 'VAR'* ('GA'* (C+E)+'ENTIER'),
'E'* ('VAR'* ('GA'* (C+E)+'ENTIER')+'*'+ GV1(B⊖1, C⊕2, E));
```

La fonction GR construit la ramification des instructions de garnissage du vecteur de renseignements de la déclaration de procédure, son premier argument. Ses deuxième, troisième et quatrième arguments sont respectivement identiques aux deuxième, quatrième et cinquième arguments de la fonction A.

Les quatre premières affectations portent sur les quatre premières grandeurs du vecteur renseignement ; (cf. III.4.1).

La fonction GR1 a trois arguments :

- la liste des spécifications,
- la première adresse libre dans le vecteur de renseignements de la procédure,
- l'index à associer à cette adresse.

Elle construit les instructions de garnissage du vecteur, d'un indicatif d'appel par valeur ou nom et du type des paramètres formels de la procédure.

Les fonctions GR2 et GR3, non explicités, fournissent des codes relatifs à ceux des spécifications, qui n'ont pas été, eux-mêmes, explicités.

La fonction S1 admet, pour paramètres :

- la liste des spécifications,
- un entier initialisé à 1,
- l'index associé à la procédure.

Elle joint, à chaque spécification, le numéro de l'index et le rang du paramètre dans la liste des paramètres formels, en se servant du second paramètre qu'elle fait progresser.

La fonction EF joint, aux paramètres effectifs d'un appel de procédure ainsi qu'à l'appel lui-même, les étiquettes nécessaires à la séquence d'appel. Elle regroupe sous 'PEF' un paramètre formel et ses étiquettes. Son premier argument est la liste des paramètres formels (qui sera, en fait, traitée par la fonction EF1) ; son second argument est le premier entier non encore utilisé comme étiquette symbolique.

La fonction EF1 a des paramètres identiques à ceux de la fonction EF.

```
FONCTION  GR (A, B, C, D)
'iA'* ('VAR'* ('GA'* (B+C)+'ENTIER') + 'iT'* F )+
'iA'* ('VAR' * ('GA'* (B⊕1+C) + 'ENTIER') + 'ci'* C⊕2 ) +
'iA'* ('VAR' * ('GA'* (B⊕2+C) + 'ENTIER') + 'ci' *(ci(A)='REEL':4 , ci(A)='ENTIER':3, ci(A)='BOOL':2, 1))+
'iA'* ('VAR' * ('GA'* (B⊕3+C) + 'ENTIER') + 'ci'* NR(CA3(A)) +
GRI (CAMI (CA2(A)), B⊕4 , C);

FONCTION  GRI (A, B, C)
NR(A)=1 : 'iA'* ('VAR'* ('GA'* (B+C)+ 'ENTIER') + 'ci'* GR2 (CSI(A))) +
          'iA'* ('VAR'* ('GA'* (B⊕1+C)+ 'ENTIER') + 'ci'* GR3 (CSI(A))) ,
'iA'* ('VAR'* ('GA'* (B+C)+'ENTIER')+'ci'* GR2 (CSI (CMI (A))))+
'iA'* ('VAR'* ('GA'* (B+C)+'ENTIER') + 'ci'* GR3 (CSI (CMI (A)))) +
GRI (CAMI(A), B⊕2, C);

FONCTION  SI (A, B, C)
NR(A) = 1 : 'S'* (C + SR (A) + B),
'S'*(C + SR (CMI (A))+B) + SI (CAMI (A), B⊕1, C);

FONCTION  EF (A, B)
RO(A)* ( CI(A) + EFI (CAI (A), B⊕8) + 'iT'* B + 'iT'* (B⊕1) + 'iT'* B⊕2 + 'iT'* B⊕3
       + 'iT'* B⊕4 + 'iT'* B⊕5 + 'iT'* B⊕6 );

FONCTION  EFI (A,B)
NR(A)=1 : 'PEF' * (A + 'iT'* B + 'iT'* B⊕1 + 'iT'* B⊕2 + 'iT'* B⊕3),
'PEF'* (CI(A) + 'iT'* B + 'iT'* B⊕1 + 'iT'* B⊕2 + 'iT'* B⊕3) + EFI (CAI(A), B⊕4);
```

IV.3 - TRANSPORT DES RESERVATIONS

L'étude lexicographique a associé des adresses aux déclarations de variables, de tableaux et de procédures, et des étiquettes symboliques aux déclarations d'étiquettes.

Cette phase remplace chaque utilisation d'un identificateur par les renseignements associés à sa déclaration. Elle supprime ensuite le lexique devenu inutile.

La fonction TD organise le transport des renseignements associés aux identificateurs dans le lexique. Pour chaque bloc de la ramification du programme, elle appelle la fonction TF. Elle supprime ensuite, en appelant la fonction TS, les éléments du lexique contenus dans le bloc traité.

Les deux arguments de la fonction TF sont le bloc à traiter. Elle recherche, dans le deuxième argument, les utilisations d'un identificateur, et, s'il est déclaré dans ce bloc, le remplace par les renseignements associés dans le lexique, résultat de la fonction TG.

Le premier argument de la fonction TG est le bloc où est utilisé l'identificateur deuxième argument. Elle le compare aux déclarations du bloc et rend $ en cas d'échec, l'adresse symbolique si l'étiquette est déclarée dans le bloc, l'adresse et le type si la variable ou le tableau ou la procédure est déclaré dans le bloc , l'adresse en début de zone statique si l'identificateur est celui d'un paramètre formel ; (cf. III.4.3).

Les comparaisons portent, dans l'ordre, sur :

- les déclarations de variables et tableaux ;
- les déclarations d'étiquettes. TG1 recherche, dans la liste d'étiquettes déclarées, son premier argument ou une éventuelle coïncidence avec l'étiquette deuxième argument ;
- l'identificateur d'une procédure pour le cas où il se trouverait devant une affectation ;
- les spécifications :

  = d'un paramètre formel appelé par valeur
  = d'un paramètre formel étiquette
  = d'un paramètre formel tableau
  = d'un paramètre formel appelé par nom
  = d'une procédure lors d'un appel formel ;

- les déclarations de procédures pour un appel direct.

La fonction GR5, qui n'est pas explicitée, fournit, à partir du code de spécification, la chaîne 'ENTIER' ou 'REEL' ou 'BOOL' selon les cas.

La fonction TS supprime les éléments inutiles du bloc traité, et assimile, pour la suite, les constantes immédiates à des variables.

```
FONCTION  TD (A)   REC (A)
$ ;
TD(R) + TD(S):
T='BL' OU T='DP': T * TS (TF (TD(U), TD (U))), T * TD (U);

FONCTION  TF (A, B)   REC (B)
$;
TF (A, R) + TF (A, S);
T='VAR' OU T='VFG' OU T='VFD' OU T='ISP': (TG (A, U) ≠ $ : TF (A, TG (A, U)), T * TF (A, U)),
RO(U) = 'ETIQ' : (TG (A, U) ≠ $ : T * TG (A, U), T * U),
T ≠ 'DE' ET (RO(C2(U)) = 'ETIQ' OU RO(C3(U)) = 'ETIQ'):
  (TG (A, C2(U)) ≠ $ : (TG (A, C3(U)) ≠ $ : T * (TF (A, C1(U)) + TG (A, C2 (U)) + TG (A, C3(U)) + TF (A, CA3(U))),
                        T * (TF (A, C1(U)) + TG (A, C2 (U)) + TF (A, CA2 (U)))),
   TG (A, C3(U)) ≠ $ : T * (TF (A, C1(U) + C2(U)) + TG (A, C3 (U)) + TF (A, CA3 (U))), T * TF (A, U)),
T * TF (A, U);

FONCTION  TG (A, B)   REC (A)
$;
TG (R, B) + TG (S, B);
T='D' ET C3 (U) = CS1 (B) : 'VAR' * (C1 (U) + CSA1 (B) + C2 (U)),
T='DE' ET TG1 (CA1 (U), SR (B)) = 'VRAI' : C1 (U),
T='DPP' ET C8 (U) = SR (B) : 'VAR' * ('GA' * (0 + C1 (U)) + C2 (U)),
T='S' ET C8 (U) = CS1 (B):
  (GR4 (C2(U)) = 'VALEUR' : 'VAR' * ('GA' * (4 ⊛ C4 (U) + C1 (U)) + GR5 (C2 (U))),
   RO(B) = 'ETIQ' : 'VAR' * ('GA' * ('#' + 2 ⊕ 7 ⊛ C4 (U) + C1 (U)) + 'ENTIER'),
   RO(B) = 'VAR' : 'VAR' * ('GA' * ('VA' * 'GA1' * ('#' + 2 ⊕ 7 ⊛ C4 (U) + C1 (U)) +
                                'VA' * 'GA1' * ('#' + 3 ⊕ 7 ⊛ C4 (U) + C1 (U))) + CSA1 (B) + 'ENTIER'),
   RO (B) = 'VFG' : 'VFG' * (CS2 (B) + 'GA' * (4 ⊛ C4 (U) ⊖ 1 + C1 (U)) + 'GA' * ('VA' * 'GA1' * ('#' + 2 ⊕ 7 ⊛ C4 (U) +
                C1 (U)) + 'VA' * 'GA1' * ('#' + 3 ⊕ 7 ⊛ C4 (U) + C1 (U))) + GR5 (C2 (U))),
   RO(B) = 'VFD' : 'VFD' * ('GA' * (4 ⊛ C4 (U) + C1 (U)) + CS2 (B) + GR5 (C2 (U))), $),
T='S' ET C3 (U) = CS3 (B) : 'ISP' * ('GA' * ('VA' * 'GA1' * ('#' + 2 ⊕ 7 ⊛ C4 (U) + C1 (U)) + 'VA' * 'GA1' * ('#' +
      3 ⊕ 4 ⊛ C4 (U) + C1 (U))) + 'GA' * ('VA' * ('GA1' * ('#' + 2 ⊕ 7 ⊛ C4 (U) + C1 (U)) + '⊞' + 1) + 'VA' * 'GA1' *
      ('#' + 3 ⊛ 7 ⊛ C4 (U) + C1 (U))) + C4 (U) + CS1 (B) + CS2 (B) + CSA3 (B) + GR5 (C2 (U))),
T='DP' ET C3 (SC3 (U)) = CS3 (B) : 'ISP' * (C1 (U) + CS1 (C3 (U)) + CS2 (C3 (U)) + CS1 (B) + CS2 (B) +
                                            CSA3 (B) + SC2 (C3 (U))),
 $;

FONCTION  TG1 (A, B)
NR (A) = 0 : 'FAUX', CS1 (A) = B : 'VRAI', TG1 (CA1 (A), B);

FONCTION  TS (A)   REC (A)
$;
TS (R) + TS (S);
T='CI' : 'VAR' * (T * U + 'ENTIER')
T='D' : (C4 (U) ≠ $ : CA3 (TS (U)), $),
T='DE' : C1 (U),
T='PDB' : TS (U),
T * TS (U);
```

IV.4 - TRANSFERT DES TYPES ET ADRESSAGE DES RESULTATS INTERMEDIAIRES

IV.4.1) But

Le but de cette transformation est d'associer un type et une adresse à chaque résultat intermédiaire, en mettant en place les conversions nécessaires.

Nous avons supposé que le programme source était syntaxiquement correct. Il ne serait guère plus compliqué de traiter ici le problème de la détection des erreurs, à condition de définir des libellés d'erreur dans le langage de description.

IV.4.2) Adressage des résultats intermédiaires

L'adressage des résultats intermédiaires se fera, pour chaque bloc, à partir de la première mémoire libre en zone statique, dont l'adresse a été jointe au bloc lors de l'étude lexicographique.

Deux résultats intermédiaires nécessaires pour le calcul d'un troisième auront des emplacements distincts en mémoire : celui du premier suivant celui du second, dans l'ordre des adresses, le troisième occupant le même emplacement que le second.

Les résultats intermédiaires des trois expressions d'une expression conditionnelle seront adressés à partir de la même adresse.

Les valeurs des indices d'une variable indicée seront rangées, en suivant, parmi les résultats intermédiaires.

Nous réserverons deux mots de mémoire pour l'éventuel résultat d'un appel de procédure.

IV.4.3) Contrôles de débordement

Nous déterminerons, pour chaque bloc, le sommet de la zone statique, après l'adressage des résultats intermédiaires, et joindrons son adresse aux indicateurs de contrôle de débordement de mémoire.

Ce contrôle sera effectué à l'exécution, à l'entrée de chaque bloc et à chaque réservation dynamique.

La fonction TT associe, à chaque résultat intermédiaire, un type qu'elle détermine, en fonction des types des quantités utilisées pour son calcul.

Elle met en place les conversions nécessaires en les indiquant par les opérateurs TRE et TER qui serviront à générer, respectivement, la conversion d'un réel en un entier et d'un entier en un réel.

Pour ne pas alourdir cette partie, ainsi que la génération du programme objet, nous n'avons pas distingué les divers cas de l'exponentiation d'une valeur par une autre, et rendu un résultat réel dans tous les cas. Il est évident qu'un compilateur efficace ne saurait être conçu de la sorte.

La fonction TA5 traite le cas des indices d'une variable indicée : lorsqu'ils sont réels, on prévoit le calcul de l'entier le plus proche, et lorsqu'ils sont entiers, leur rangement a une adresse de résultat intermédiaire, à l'aide de l'opérateur AFE.

Ceci permettra d'avoir les valeurs des indices d'une variable indicée, regroupées à des adresses consécutives, et facilitera l'appel du sous-programme de calcul d'adresse.

```
FONCTION TT(A) REC(A)
$;
TT(R) + TT(S);
T='E': (C1(U)='+' OU C1(U)='-': T* (TT(U) + CSM1(C2(TT(U)))),
        C1(U)='?': T* (TT(U) + 'BOOL'),
        C1(U)='ENV': (CSM1(C2(TT(U)))='ENTIER': C2(TT(U)),
           'E'* ('TRE' + 'E'* (C2(TT(U)) + '+' + 'VAR'* ('Ci'* 0.5 + 'REEL') + 'REEL') + 'ENTIER'),
        C2(U)='=' OU C2(U)='⊃' OU C2(U)='∧' OU C2(U)='∨': T* (TT(U) + 'BOOL'),
        C2(U)='>' OU C2(U)='≥' OU C2(U)='=' OU C2(U)='≠' OU C2(U)='≤' OU C2(U)='<':
        (CSM1(C1(TT(U))) = CSM1(C3(TT(U))): T* (TT(U) + 'BOOL'),
         CSM1(C1(TT(U)))='ENTIER': T* ('E'* ('TER' + C1(TT(U)) + 'REEL') + CA1(TT(U)) + 'BOOL'),
         T* (CAM1(TT(U)) + 'E'* ('TER' + C3(TT(U)) + 'REEL') + 'BOOL')),
        C2(U)='+' OU C2(U)='-' OU C2(U)='*':
         (CSM1(C1(TT(U))) = CSM1(C3(TT(U))): T* (TT(U) + CSM1(C1(TT(U)))),
         CSM1(C1(TT(U))) = 'ENTIER': T* ('E'* ('TER' + C1(TT(U)) + 'REEL') + CA1(TT(U)) + 'REEL'),
         T* (CAM1(TT(U)) + 'E'* ('TER' + C3(TT(U)) + 'REEL') + 'REEL')),
        C2(U)='÷': T* (TT(U)) + 'ENTIER'),
        C2(U)='/' OU C2(U)='↑':
         (CSM1(C1(TT(U)))='REEL': (CSM1(C3(TT(U)))='REEL': T*(TT(U) + 'REEL'),
            T* (CAM1(TT(U)) + 'E'* ('TER' + C3(TT(U)) + 'REEL') + 'REEL')),
          CSM1(C3(TT(U)))='REEL': T* ('E'* ('TER' + C1(TT(U)) + 'REEL') + CA1(TT(U)) + 'REEL'),
          T* ('E'* ('TER' + C1(TT(U)) + 'REEL') + C2(U) + 'E'* ('TER' + C3(TT(U)) + 'REEL') + 'REEL')),
T='EC': (CSM1(C2(TT(U))) = CSM1(C3(TT(U))): T* (TT(U) + CSM1(C2(TT(U)))),
         CSM1(C2(TT(U)))='ENTIER': T* (C1(TT(U)) + 'E'* ('TER' + C2(TT(U)) + 'REEL') + CA2(TT(U)) + 'REEL'),
         T* (C1(TT(U)) + C2(TT(U)) + 'E'* ('TER' + C3(TT(U)) + 'REEL') + CA3(TT(U)) + 'REEL')),
T='IA': (CSM1(C1(TT(U))) = CSM1(C3(TT(U))): T* (TT(U) + CSM1(C1(TT(U)))),
         CSM1(C1(TT(U)))='REEL': T* (C1(TT(U)) + 'E'* ('TER' + C2(TT(U)) + 'REEL') + 'REEL'),
         T* (C1(TT(U)) + 'E'* ('TRE' + 'E'* (C2(TT(U)) + '+' + 'VAR'* ('Ci'* 0.5 + 'REEL') + 'REEL')
            + 'ENTIER') + 'ENTIER')),
T='VAR' ET CS3(U) ÷ $: T* (C1(U) + TA5(CA1(CAM2(TT(U)))) + CM2(U) + CM1(U)),
T='PEF': T* (TT(U) + CM1(SC1(TT(U)))),
T* TT(U);

FONCTION TA5(A)
NR(A)=1: (CSM1(A)='ENTIER': 'E'* ('AFE' + A + 'ENTIER'),
          'E'* ('TRE' + 'E'* (A + '+' + 'VAR'* ('Ci'* 0.5 + 'REEL') + 'REEL') + 'ENTIER')),
CSM1(C1(A)) = 'ENTIER': 'E'* ('AFE' + C1(A) + 'ENTIER'),
'E'* ('TRE' + 'E'* (C1(A) + '+' + 'VAR'* ('Ci'* 0.5 + 'REEL') + 'REEL') + 'ENTIER') +
TA5(CA1(A));
```

La fonction TA organise l'adressage des résultats intermédiaires. Elle appelle, pour chaque bloc n'en contenant pas d'autres, la fonction TA1 qui effectue l'adressage. Le premier paramètre de TA1 est le bloc à transformer. Les second et troisième paramètres de TA1 sont l'adresse et le numéro du registre d'index indiquant la première mémoire libre en zone statique. Ils ont été pris dans le groupe adresse, deuxième composante connexe du bloc, qui avait été créée à cette intention par la fonction A. Si T * U est le bloc à traiter, le bloc adressé est :

$$R = T * TA1\ (TA(U),\ CS1\ (C2(U)),\ CS2\ (C2(U))).$$

La fonction TA2 recherche, ensuite, dans le bloc adressé (son premier argument), l'adresse du sommet de la zone statique de ce bloc, en ignorant celle des blocs internes éventuels. Son second argument est le numéro du registre d'index utilisé pour la zone statique du bloc et est utilisé à des fins de vérification.

Ce groupe adresse R1 = TA2 (R, CS2 (C2(U))) est ensuite associé aux indicateurs de contrôle de débordement de mémoire, au moyen de la fonction TA3. Il en sera le second argument. Le premier argument est le bloc adressé, privé de sa deuxième composante connexe désormais inutile :

$$TA3\ (C1(U) + CA2\ (R),\ R1)$$

La transformation du bloc étant achevée, la fonction TA procède à la transformation du bloc le contenant, etc...

La fonction RESI indique l'organisation de la phase de traitement des résultats intermédiaires dans le compilateur.

```
FONCTION TA(A) REC (A)
$;
TA(R) + TA(S);
T='BL': T* TA3 (C1(U)+ CA2 (TAI(TA(U), CS1(C2(U)), CS2(C2(U)))),
                TA2 (TAI(TA(U), CS1(C2(U)), CS2(C2(U))), CS2(C2(U)))),
T* TA(U);

FONCTION TAI (A, B, C) REC (A)
$;
RO(S)='E' OU RO(S)='EC' OU (RO(S)='VAR' ET CS3(S)≠$) OU RO(S)='ISP':
TAI(R, B⊕(CSMI(S)≠'REEL' OU RO(S)='VAR': 1, 2), C)+TAI(S, B, C),
TAI(R, B, C)+TAI(S, B, C);
T='E': T* ('GA'* (B+C)+TAI(U, B, C)),
T='EC': T* ('GA'* (B+C)+TAI(C1(U), B, C)+TAI(C2(U), B, C)+TAI(CA2(U), B, C)),
T='ISP': T* ('GA'* (B+C)+TAI(U, B⊕2, C)),
T* TAI(U, B, C);

FONCTION TA2 (A, B) REC (A)
$;
MAX(TA2(R, B), TA2(S, B));
T='BL': $,
T='GA' ET C2(U)=B: T*U, TA2(U, B);

FONCTION MAX (A, B)
A=$: B,
B=$: A,
CS2(A)>CS2(B): A,
B;

FONCTION TA3 (A, B) REC (A)
$;
TA3(R, B)+TA3(S, B);
T='SPI' ET C3(U)=$: T* (U+B),
T* TA3(U, B);

FONCTION RESI(A)
TA(IT(A));
```

IV.5 - GENERATION DU PROGRAMME OBJET

Le passage de la pseudo-arborescence transformée du programme source à celle du programme objet s'effectue selon les schémas classiques de génération. Les transformations précédentes ont placé, dans la pseudo-arborescence du programme source, les opérandes d'une instruction au voisinage de la racine qui en déterminera la génération.

Nous avons utilisé, pendant la génération, une racine : 'DO', pour regrouper, temporairement, des instructions du programme objet que nous souhaitions manipuler conjointement.

L'étiquetage de l'instruction, suivant celle en cours de génération, a été effectué en deux temps :

- nous avons fait suivre l'instruction en cours de génération par une instruction étiquetée, artificielle, reconnaissable à son code : CODE (0, 0) ;
- nous avons, ensuite, supprimé, dans la pseudo-arborescence du programme généré, chacune de ces instructions artificielles, et reporté son étiquette sur l'instruction suivante.

Ce report d'étiquettes pourra faire apparaître un double étiquetage. Nous remplacerons alors les occurences de la première étiquette d'un double étiquetage par la seconde étiquette, et supprimererons la première étiquette des doubles étiquetages.

La fonction B effectue la génération des instructions du programme objet. L'adresse du dernier résultat calculé lui est fournie par la fonction DU. C'est, en général, le dernier opérande de la dernière instruction générée.

Les fonctions DIE et DII ont pour résultat la ramification composée de la séquence des instructions du programme objet, figurant dans leur ramification argument.

Variables indicées

Dans le cas d'une variable indicée, la fonction B met en place un appel au sous-programme de calcul d'adresse, et rend l'adresse indirecte de la variable indicée.

Paramètres formels

Les paramètres formels appelés par nom sont remplacés par des appels aux sous-programmes d'évaluation de ces paramètres.

Appels de procédures

Un appel de procédure est remplacé par la séquence :

1 - appel au sous-programme n° 6, vérifiant la conformité des paramètres
2 - instructions de calcul de l'écart k - L(cf. III.3.3.3)
3 - instructions d'évaluation des paramètres
4 - instructions de garnissage de l'indicatif d'expression des paramètres effectifs
5 - appel au sous-programme n° 7 de sauvegarde des registres, et de branchement vers le corps de procédure
6 - appel au sous-programme n° 8 de restauration des registres après l'appel.

Les séquences 2,,3 et 4 sont décrites par les fonctions TP3, TP2, TP1.

Autres instructions et expressions

La génération des autres instructions et expressions du programme est effectuée selon les schémas classiques.

```
FONCTION B(A) REC (A)
$;
B(R) + B(S);
T='VAR' ET C3(U)≠$:
 'DO'* (DIE (B(U))+'ORDRE'* (CODE (SP,0)+'ci'*3 + C1(U)+'ci'* NR(U)⊖3 + DU (CM3 (B(U))) +
       CM2(U))+'ORDRE'* (CM2(U)+ CODE (0,0)+'6A'* (#+ SR (DU (CM3 (B(U))))))),
T='E': (NR(U)=4:
        'DO'* (DIE(B(U))+'ORDRE'*(CODE (C2(U), CM1(U))+ DU (C3(B(U)))+ C1(U))),
        'DO'* (DIE (B(U))+'ORDRE'*(CODE (C3(U), CM1(U))+DU (C2(B(U))+ DU (C4(B(U)))+ C1(U)))),
T='EC': 'DO'*(DIE (C2(B(U)))+'ORDRE'*(CODE (BR, BOOL)+ DU (C2(B(U)))+ CM3(U))+
              DIE (C8(B(U)))+'ORDRE'*(CODE (AFE, CM1(U))+ DU (C3(B(U)))+ C1(U))+
              'ORDRE'*(CODE (BR,0)+ CM2(U))+ 'ORDRE'*(CM3(U)+ CODE (0,0))+
              DIE (C4(B(U)))+'ORDRE'*(CODE (AFE, CM1(U))+ DU (C4(B(U)))+ C1(U))+
              'ORDRE'*(CM2(U)+ CODE (0,0)+ DU (C4(B(U))))),
T='IO' OU T='IOO': $,
T='IM': 'DO'* DII (B(U)),
T='I': 'DO'*('ORDRE'*(C1(U)+CODE (0,0))+ DII (B(U))),
T='IC': 'DO'*(DII (B(U))+'ORDRE'*(CODE (BR, BOOL)+ DU (C1(B(U)))+ C2(U))),
T='IB': (RO(B(U))='DO' OU C1(U)='IND': 'DO'*(DII (B(U))+'ORDRE'*(CODE (BR,0)+
        '6A'*('#'+SR (DU (CM1(B(U)))))), 'DO'*'ORDRE'*(CODE (BR,0)+U)),
T='IA': 'DO'*(DIE (B(U))+'ORDRE'*(CODE (AFE, CM1(U))+DU (C2(B(U)))+DU (C1(B(U))))),
T='SP1': 'DO'*'ORDRE'*(CODE (SP,0)+'ci'*1+U),
T='SP2': 'DO'*'ORDRE'*(CODE (SP,0)+'ci'*2+U),
T='BP': 'DO'*('ORDRE'*(CODE (BR,0)+CM1(U))+'ORDRE'*(CM2(U)+CODE (0,0))+DII (B(U))+
        'ORDRE'*(CODE (BR,0)+'6A'*('#'+2+C3(C3(U)))+'ORDRE'*(CM1(U)+CODE(0,0)))),
T='VFG': 'DO'*('ORDRE'*(CODE (SP,0)+'ci'*4+CAM1(U))+'ORDRE'*(C1(U)+CODE (0,0)+C3(U))),
T='VFD': 'DO'*('ORDRE'*(CODE (SP,0)+'ci'*5+CAM2(U))+'ORDRE'*(C2(U)+CODE (SP,0)+'ci'*9+C3(U))+
        'ORDRE'*(C3(U)+CODE (0,0)+C1(U))),
T='PEF': (C1(U)='VAR' ET NR(SU(U))<3: T*U, T*(B(C1(U))+CAH(U))),
T='ISP': 'DO'*('ORDRE'*(CODE (SP,0)+'ci'*6+'ci'*NR(U)⊖14+C2(U)+CM2(U)+CAM8(CA6(U)))+
        'ORDRE'*(CM2(U)+CODE (0,0)+TP3(U)+
        TP2(C2(U), B(U), C3(U), C6(U), CM3(U), 1, C5(U))+
        'ORDRE'*(CM3(U)+CODE (0,0)+TP1(C1(U), B(U), 1)+
        'ORDRE'*(CODE (SP,0)+'ci'*7+C1(U)+C2(U)+C6(U)+CM6(U)+C4(U))+
        'ORDRE'*(CM6(U)+CODE (SP,0)+'ci'*8+C2(U)+CM7(U)+C4(U))+
        'ORDRE'*(CM7(U)+CODE (0,0)+C1(U))),
T='PG': 'PROGRAMME'* B(U),
$;
```

La fonction DU fournit le groupe adresse du dernier résultat calculé par les instructions du programme objet contenues dans sa pseudo-arborescence argument.

Les fonctions DII et DIE, utilisées respectivement lors de la traduction des instructions et expressions du programme source, extraient et ordonnent les séquences des instructions du programme objet contenues dans leur ramification argument.

La fonction TP1 a trois paramètres :

- le groupe adresse de la première mémoire libre en zone statique
- la liste des paramètres effectifs
- un entier initialisé à un, utilisé pour les calculs d'adresse.

Cette fonction génère, pour chaque paramètre effectif, l'affectation à deux variables du début de la zone statique de l'appel, d'une valeur booléenne indiquant si le paramètre effectif est une expression et de l'adresse symbolique des instructions d'évaluation du paramètre dans le cas d'un appel par nom.

La fonction TP3 construit, à partir de la ramification de l'appel avant la génération, la ramification des instructions de calcul de l'écart k - L (cf. III.3.3.3).

```
FONCTION DU (A)
RO (A) = 'DO' : CSMI (CSMI (A)),
RO (A) = 'ORDRE' : CSMI (A),
RO (A) = 'VAR' : CSI (A),
$;

FONCTION DIE (A) REC (A)
$;
DIE (S) + DIE (R);
T = 'DO' : U, $;

FONCTION DII (A) REC (A)
$;
DII (R) + DII (S);
T = 'DO' : U, $;

FONCTION TPI (A, B, C)
NR (B) = 0 : $,
NR (B) = 1 : (RO (CSI (B)) = 'DO' :
'ORDRE' * (CODE (AFE, BOOL) + 'CI' * 'VRAI' + 'GA' * ('VA' * ('GAI' * ('*' + SR (A)) + '⊞' + CSI (A) ⊛ 1 ⊛ 7 ⊛ C) + CS2 (A))) +
'ORDRE' * (CODE (AFE, ENTIER) + CSI (A) + 'GA' * ('VA' * ('GMI' * ('*' + SR (A)) + '⊞' + CSI (A) ⊛ 6 ⊛ 7 ⊛ C) + CS2 (A))),
'ORDRE' * (CODE (AFE, BOOL) + 'CI' * 'FAUX' + 'GA' * ('VA' * ('GAI' * ('*' + SR (A)) + '⊞' + CSI (A) ⊛ 1 ⊛ 7 ⊛ C) + CS2 (A))) +
'ORDRE' * (CODE (AFE, ENTIER) + CSI (B) + 'GA' * ('VA' * ('GAI' * ('*' + SR (A)) + '⊞' + CSI (A) ⊛ 6 ⊛ 7 ⊛ C) + CS2 (A))),
TPI (A, CAMI (B), C ⊛ 1) + TPI (A, CMI (B), C);

FONCTION TP3 (U)
'ORDRE' * (CODE (BR, >) + 'GA' * (CSI (C2 (U)) ⊛ 1 + CS2 (C2 (U))) + 'CI' * CS (U) + CM4 (U)) +
'ORDRE' * (CODE (AFE, ENTIER) + 'CI' * 2 ⊛ 2 ⊛ CS (U) + CI (U)) +
'ORDRE' * (CODE (-, ENTIER) + CI (U) + 'GA' * (CSI (C2 (U)) ⊛ 1 + CS2 (C2 (U))) + CI (U)) +
'ORDRE' * (CODE (-, ENTIER) + CI (U) + 'GA' * (CSI (C2 (U)) ⊛ 1 + CS2 (C2 (U))) + CI (U)) +
'ORDRE' * (CODE (BR, 0) + CM5 (U)) +
'ORDRE' * (CM4 (U) + CODE (AFE, ENTIER) + 'CI' * 0 + CI (U)) +
'ORDRE' * (CM5 (U) + CODE (IDO, 0));
```

Les sept arguments de la fonction TP2 sont respectivement :

- le groupe adresse du vecteur de renseignements de la procédure appelée
- la liste des paramètres effectifs
- le niveau de la procédure appelée
- l'adresse du dernier tableau
- l'étiquette EO de la suite de la séquence d'appel
- un entier indiquant le rang du paramètre effectif traité
- l'adresse du début de la zone dynamique.

La fonction TP2 construit la ramification des instructions d'évaluation des paramètres effectifs, c'est-à-dire pour chaque paramètre :

```
E1 : appel au sous-programme n° 10 ;
E2 : si le paramètre est appelé par valeur alors allera E4 ;
     X2 := adresse du paramètre effectif ;
     Y2 := numéro du registre associé ;
     Allera SUITE ; (s'il reste un paramètre effectif à
                     traiter, sinon :
     Allera EO ;)
E3 : X1 := paramètre effectif ; Allera indirect Y1 ;
     Allera SUITE ; (s'il reste un paramètre effectif à
                     traiter, sinon :
     Allera EO ;)
E4 : X1 := paramètre effectif ;
SUITE : .......
```

Les étiquettes E1, E2, E3, E4 sont celles associées au paramètre effectif, X1, X2, Y1, Y2 les mémoires réservées pour le paramètre, au début de la zone statique de l'appel.

La fonction GG calcule la nouvelle adresse de début de la zone dynamique, dans le cas où le paramètre traité, son deuxième argument, correspond à un paramètre formel procédure. Son premier argument (son résultat dans le cas général) est l'ancienne adresse de début de la zone dynamique de l'appel. Cette fonction n'a pas été explicitée.

La fonction TAS supprime les instructions de code : CODE (0, 0) et reporte leurs étiquettes sur l'instruction suivante.

```
FONCTION TP2 (A, B, C, D, E, F, G)
NR(B) = 0 : $,
NR(B) = 1 :
'ORDRE'*(CS2(B) + CODE(SP, 0) + 'Ci'*10 + CS3(B) + A + G + D + F) +
'ORDRE'*(CS3(B) + CODE(BR, BOOL) + CS4(B) + '6A'*('VA'*(CS1(A) + '⊞' + 3⊛2⊛F) + CS2(A))) +
(RO(CS1(B)) ≠ 'DO' :
  'ORDRE'*(CODE(AFF, ENTIER) + CS1(CS1(CS1(B))) + '6A'*('VA'*(CS1(A) + '⊞' + 4⊕7⊛F + '⊞' + A) + CS2(A))) +
  'ORDRE'*(CODE(AFF, ENTIER) + CS2(CS1(CS1(B))) + '6A'*('VA'*(CS1(A) + '⊞' + 5⊕7⊛F + '⊞' + A) + CS2(A))),
  $) +
'ORDRE'*(CODE(BR, 0) + E) +
'ORDRE'*(CS5(B) + CODE(0, 0)) + DIE(CS1(B)) +
'ORDRE'*(CODE(AFF, CS6(B)) + DU(CS1(B)) + '6A'*('VA'*(CS1(A) + '⊞' + 2⊛7⊛F + '⊞' + A) + CS2(A))) +
'ORDRE'*(CODE(BR, 0) + '6A'*('≠' + 'VA'*(CS1(A) + '⊞' + 4⊕7⊛F + '⊞' + A) + CS2(A))) +
'ORDRE'*(CS4(B) + CODE(0, 0)) + DIE(CS1(B)) +
'ORDRE'*(CODE(AFF, CS6(B)) + DU(CS1(B)) + '6A'*('VA'*(CS1(A) + '⊞' + 2⊛7⊛F + '⊞' + A) + CS2(A))),
NR(B) = 2 :
'ORDRE'*(CS2(B) + CODE(SP, 0) + 'Ci'*10 + CS3(C2(B)) + A + G + D + F) +
'ORDRE'*(CS3(C2(B)) + CODE(BR, BOOL) + CS4(C2(B)) + '6A'*('VA'*(CS1(A) + '⊞' + 3⊛2⊛F) + CS2(A))) +
(RO(CS1(C2(B))) ≠ 'DO' :
  'ORDRE'*(CODE(AFF, ENTIER) + CS1(CS1(CS1(C2(B)))) + '6A'*('VA'*(CS1(A) + '⊞' + 4⊕7⊛F + '⊞' + A) + CS2(A))) +
  'ORDRE'*(CODE(AFF, ENTIER) + CS2(CS1(CS1(C2(B)))) + '6A'*('VA'*(CS1(A) + '⊞' + 5⊕7⊛F + '⊞' + A) + CS2(A))),
  $) +
'ORDRE'*(CODE(BR, 0) + CS2(C1(B))) +
'ORDRE'*(CS5(C2(B)) + CODE(0, 0)) + DIE(CS1(C2(B))) +
'ORDRE'*(CODE(AFF, CS6(C2(B))) + DU(CS1(B)) + '6A'*('VA'*(CS1(A) + '⊞' + 2⊛7⊛F + '⊞' + A) + CS2(A))) +
'ORDRE'*(CODE(BR, 0) + '6A'*('≠' + 'VA'*(CS1(A) + '⊞' + 4⊕F⊕7 + '⊞' + A) + CS2(A))) +
'ORDRE'*(CS4(C2(B)) + CODE(0, 0)) + DIE(CS1(C2(B))) +
'ORDRE'*(CODE(AFF, CS6(C2(B))) + DU(CS1(C2(B))) + '6A'*('VA'*(CS1(A) + '⊞' + 2⊛7⊛F + '⊞' + A) + CS2(A))),
TP2(A, CAMI(B), C, D, E, F⊛I, GG(G, CMI(B))) +
  TP2(A, CM2(B) + CMI(B), C, D, E, F, G) ;

FONCTION TAS(A) REC(A)
$ ;
CS2(CMI(R)) = CODE(0, 0) : TAS(CAMI(R)) + 'ORDRE'*(CS1(CMI(R)) + SR(S)),
TAS(R) + TAS(S),
T = 'DO' : TAS(U), T*TAS(U);
```

La fonction ETD recherche, dans son premier argument, les occurences des étiquettes et y remplace celles provenant d'un double étiquetage par celle des deux qui sera conservée.

Son second argument sert de référence et est, comme le premier, le résultat de TAS.B.

La fonction VER appelée par la fonction ETD, à chaque occurence d'étiquette, rend l'étiquette qui sera conservée dans le cas d'un double étiquetage, la ramification vide dans tous les autres cas.

Son premier argument est l'étiquette fournie par la fonction ETD ; son deuxième argument est le résultat de TAS.B.

La fonction NET supprime la première des étiquettes de chaque double étiquetage d'une instruction de sa ramification argument.

La fonction GENER exprime les transformations à effectuer pour réaliser la génération du programme objet.

L'expression GENER (RESI(TD(ADR(PREL(W))))) indique la transformation à effectuer sur la pseudo-arborescence du programme source, pour obtenir celle du programme objet. Elle clôt le programme de description du compilateur.

```
FONCTION ETD (A,B) REC(A)
$;
ETD (R,B) + ETD (S,B);
T = 'IT': (VER (T*U,B) = $: T*U , VER (T*U, B)),
T* ETD (U,B);

FONCTION VER (A,B) REC (B)
$;
VER (A,R) + VER (A,S);
C1(U) = A ET RO(C2(U)) = 'IT': C2(U), VER (U, B);

FONCTION NET (A) REC (A)
$;
NET (R) + NET (S);
RO(C1(U)) = 'IT' ET RO(C2(U)) = 'IT': T* CAI (U),
T* NET (U);

FONCTION GENER (A)
   NET (ETD (TAS (B(A)), TAS (B(A))));

   GENER (RESI (TD (ADR (PREL (W)))))

FIN
```

IV.6 - ANNEXE : OBJET DES SOUS-PROGRAMMES D'INTERPRETATION

Sous-programme n° 1

Ce sous-programme effectue un contrôle de débordement de mémoire, en comparant l'adresse de la première mémoire libre après la zone statique et celle de la première mémoire libre avant la zone dynamique (ses deux premiers arguments). Dans le cas d'un débordement, il y a impression d'un message et arrêt de l'exécution. Le retour est effectué à l'adresse, troisième paramètre, dans le cas contraire.

Sous-programme n° 2

Ce sous-programme contrôle la validité de l'intervalle de variation d'un indice, dans une déclaration de tableau. En cas d'erreur, il y a impression d'un message et arrêt de l'exécution.

Sous-programme n° 3

Ce sous-programme calcule l'adresse d'une variable indicée. Les paramètres (outre l'adresse de retour) sont l'adresse du vecteur de renseignements du tableau, le nombre d'indices et l'adresse à partir de laquelle, en zone statique, sont rangées les valeurs des indices.
Cette dernière adresse contiendra l'adresse de la variable indicée, après l'appel du sous-programme.

Sous-programme n° 4

Ce sous-programme est appelé lors de l'utilisation à gauche du signe d'affectation, d'un paramètre formel appelé par nom.
Lorsque le paramètre effectif n'est pas une expression, son adresse est transmise, sinon il y a impression d'un message d'erreur et arrêt de l'exécution.

Sous-programme n° 5

Ce sous-programme évalue un paramètre formel appelé par nom, lors de son utilisation à droite d'une affectation. Il restaure, éventuellement, les registres pour permettre l'évaluation, et range les valeurs précédentes de ces registres.

Sous-programme n° 6

Ce sous-programme vérifie la conformité du nombre et des types des paramètres effectifs d'un appel de procédure. En cas d'erreur, il interrompt l'exécution et transmet un message.

Sous-programme n° 7

Ce sous-programme range, en zone statique, les valeurs des registres utilisés après l'appel, détruits pendant celui-ci, et initialise ceux relatifs à la zone de variables de l'appel. Dans le cas d'un appel formel, il restaure également les registres concernés. Il se termine par un branchement vers le corps de la procédure appelée.

Sous-programme n° 8

Ce sous-programme restaure, après un appel, les registres dans les valeurs rangées par le sous-programme n° 7.

Sous-programme n° 9

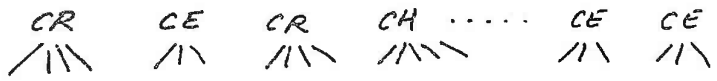
Analogue au sous-programme n° 8.; dans le cas d'un paramètre formel appelé par nom à droite du signe d'affectation, il restaure les registres dans les valeurs rangées par le sous-programme n° 5.

Sous-programme n° 10

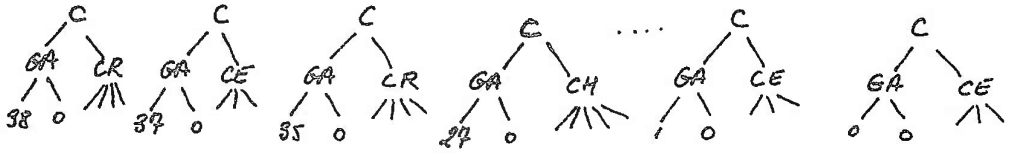
Ce sous-programme range, en zone dynamique, les valeurs des index nécessaires à l'appel formel de la procédure dont l'identificateur est un paramètre effectif de l'appel en cours de traitement.

IV.7 - SCHEMAS DES PRINCIPAUX RESULTATS DES FONCTIONS DU PROGRAMME DE DESCRIPTION

EXEMPLE DE RESULTAT DE CT

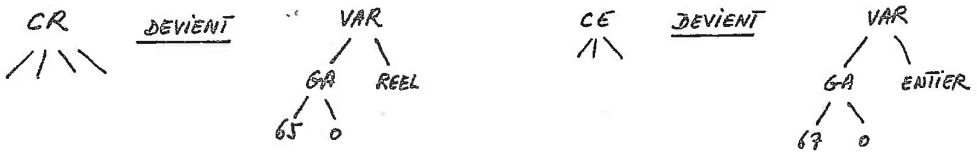


EXEMPLE DE RESULTAT DE CT1

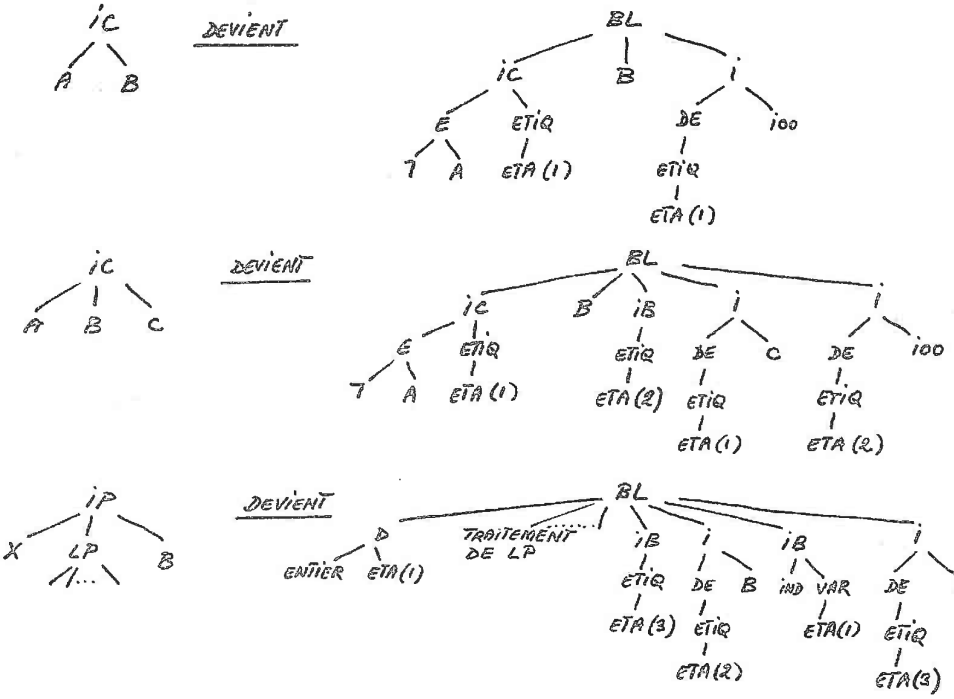


EXEMPLE DE RESULTAT DE CT4

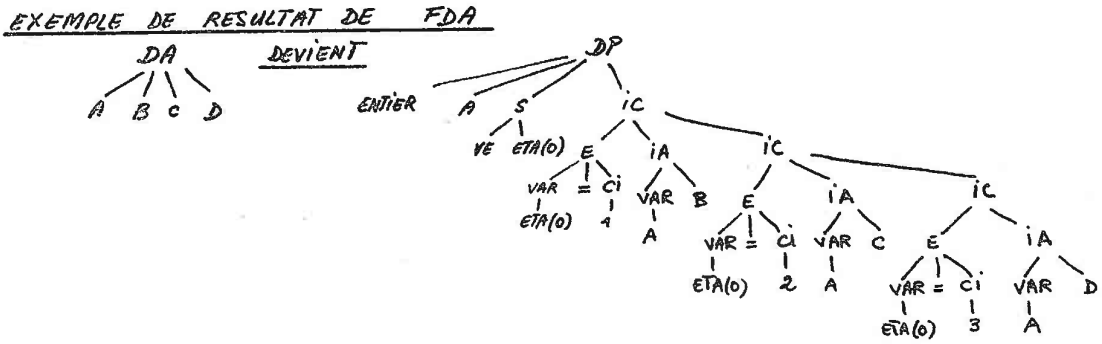
EXEMPLES DE RESULTATS DE TC



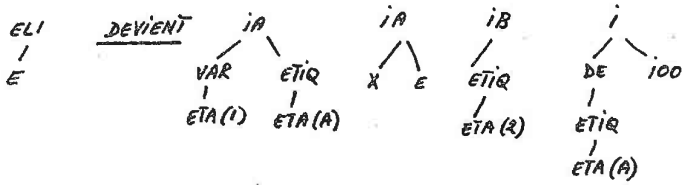
EXEMPLES DE RESULTATS DE P

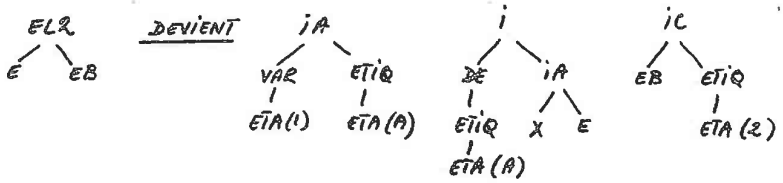


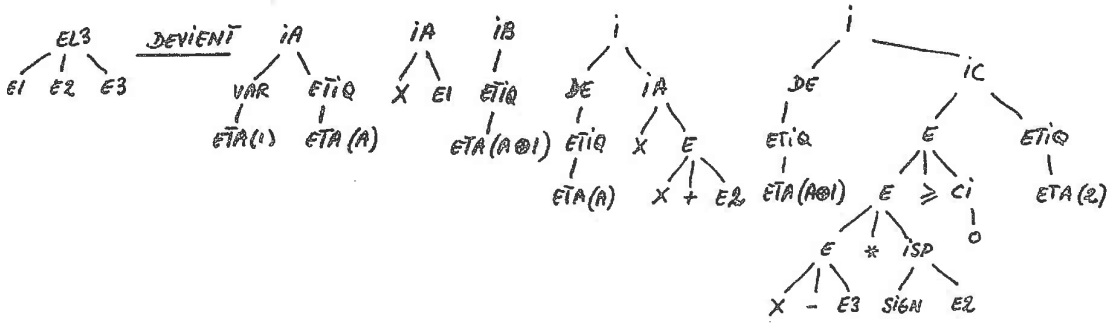
EXEMPLE DE RESULTAT DE FDA



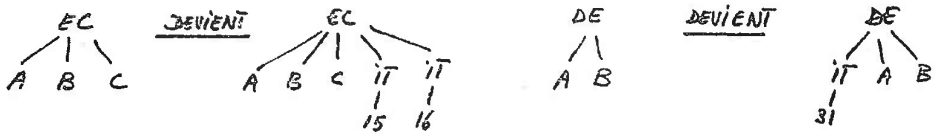
EXEMPLES DE RESULTATS DE P2

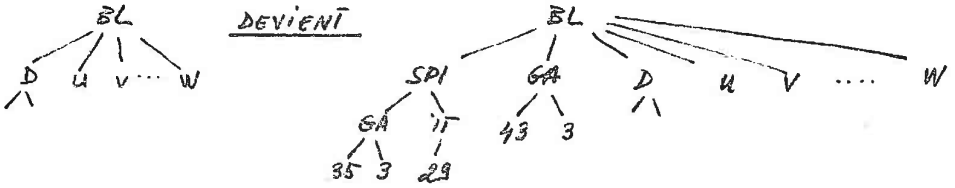


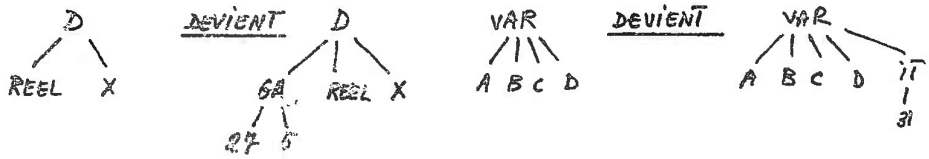


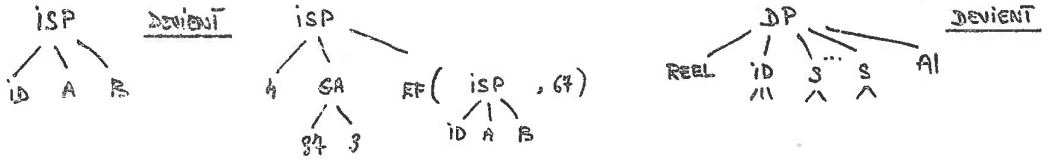


EXEMPLES DE RESULTATS DE A

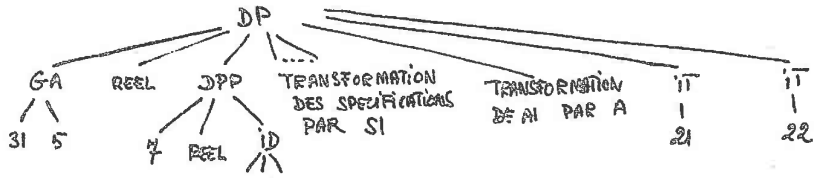




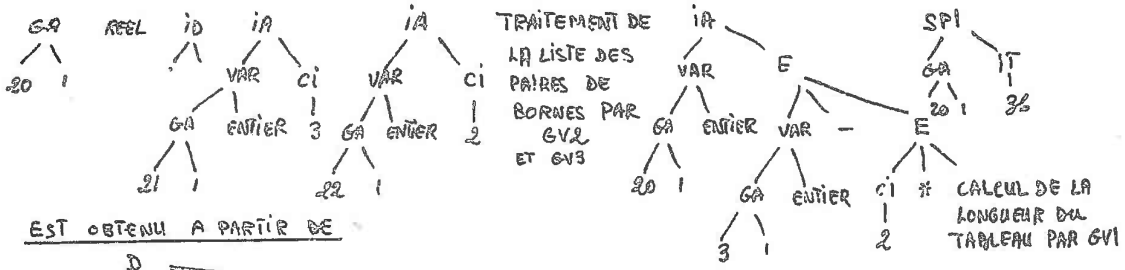




GARNISSAGE DU VECTEUR DE RENSEIGNEMENTS DE LA PROCEDURE PAR GR

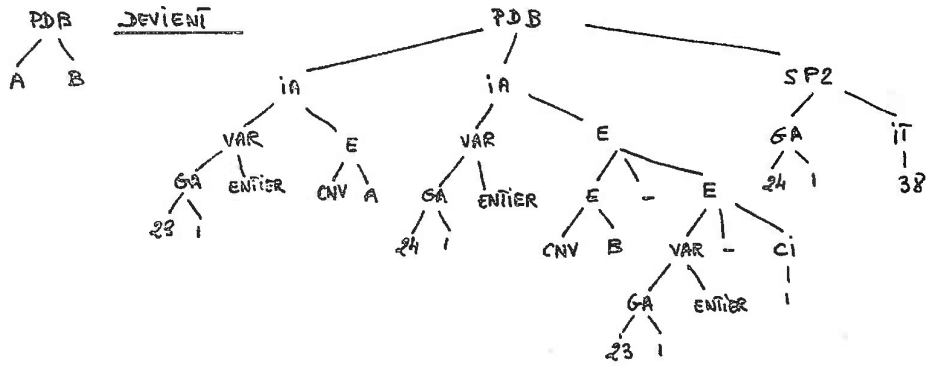


EXEMPLE DE RESULTAT DE GV

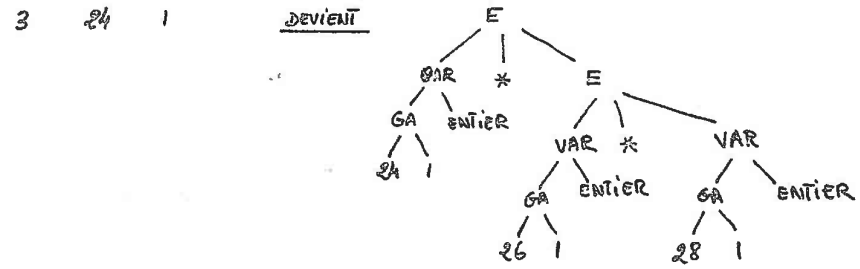


EST OBTENU A PARTIR DE

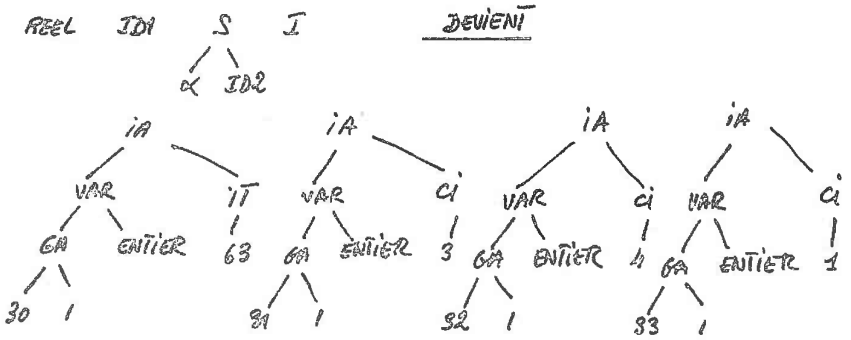
EXEMPLE DE RESULTAT DE GV3



EXEMPLE DE RESULTAT DE GV1

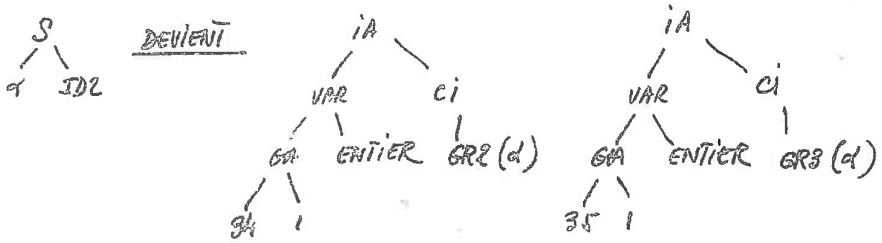


EXEMPLE DE RESULTAT DE GR

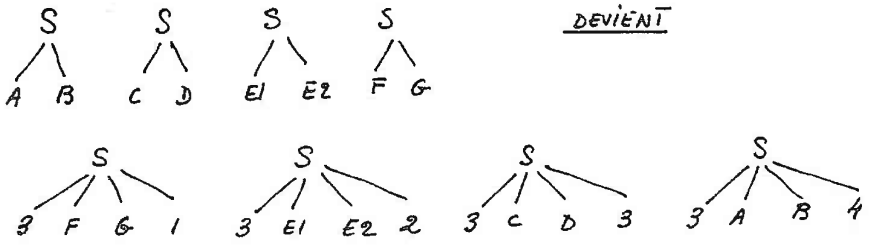


TRAITEMENT DE LA LISTE DES SPECIFICATIONS PAR GR1

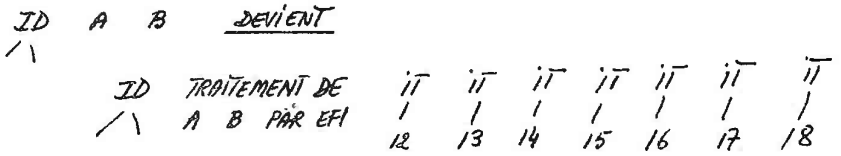
EXEMPLE DE RESULTAT DE GR1



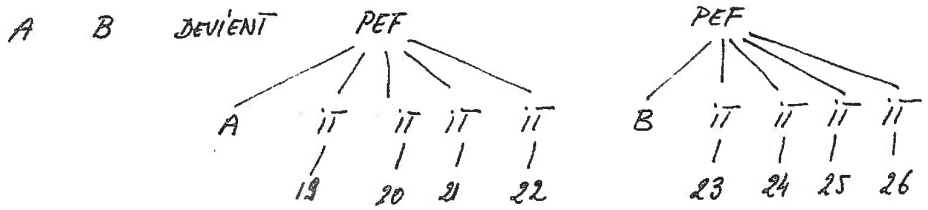
EXEMPLE DE RESULTAT DE SI



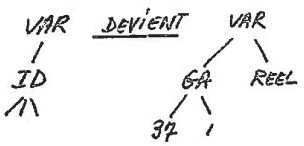
EXEMPLE DE RESULTAT DE EF

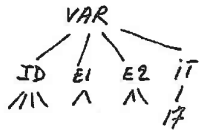


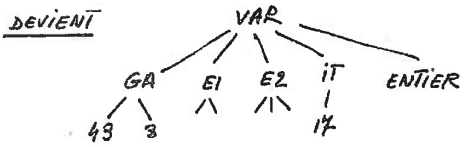
EXEMPLE DE RESULTAT DE EFI

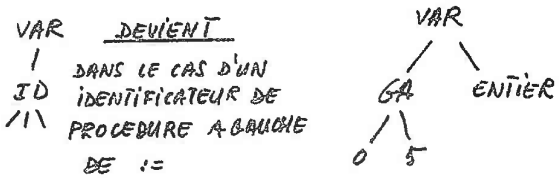


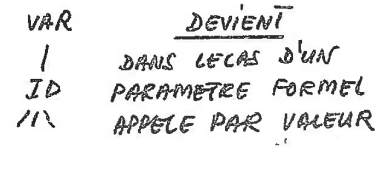
EXEMPLES DE RESULTATS DE TF ET TG

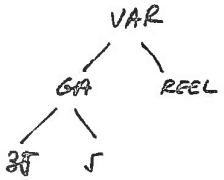


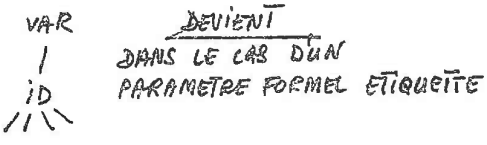


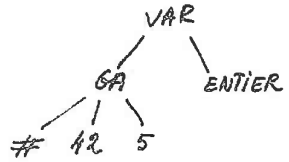


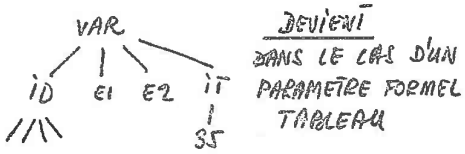


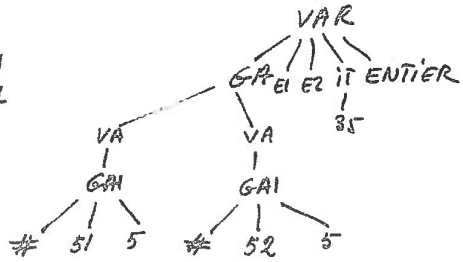


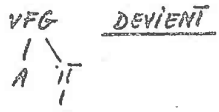


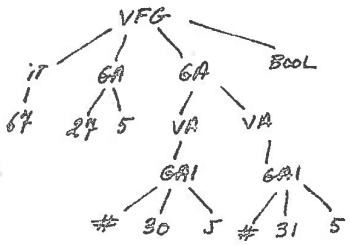


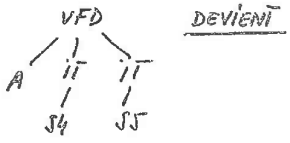


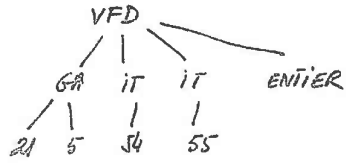


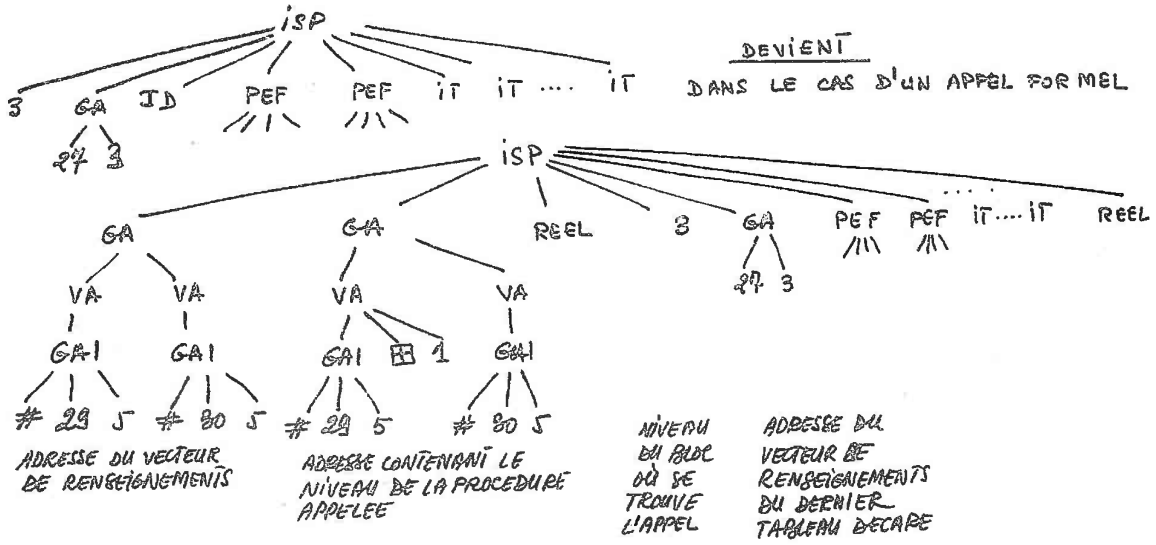


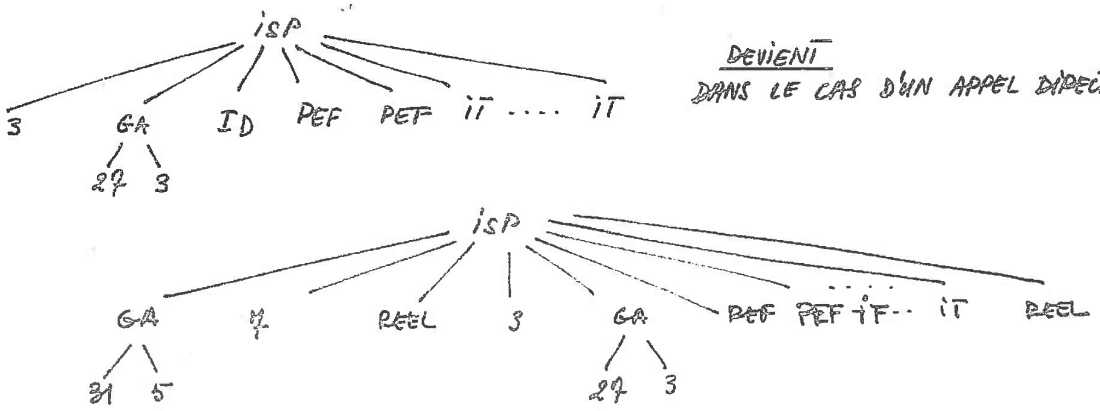




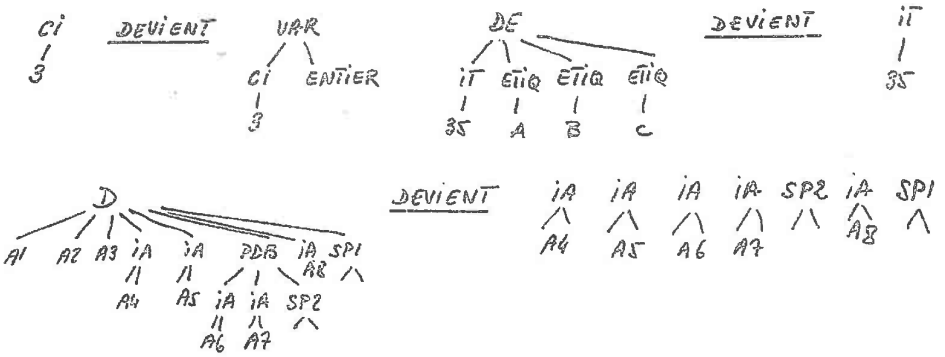




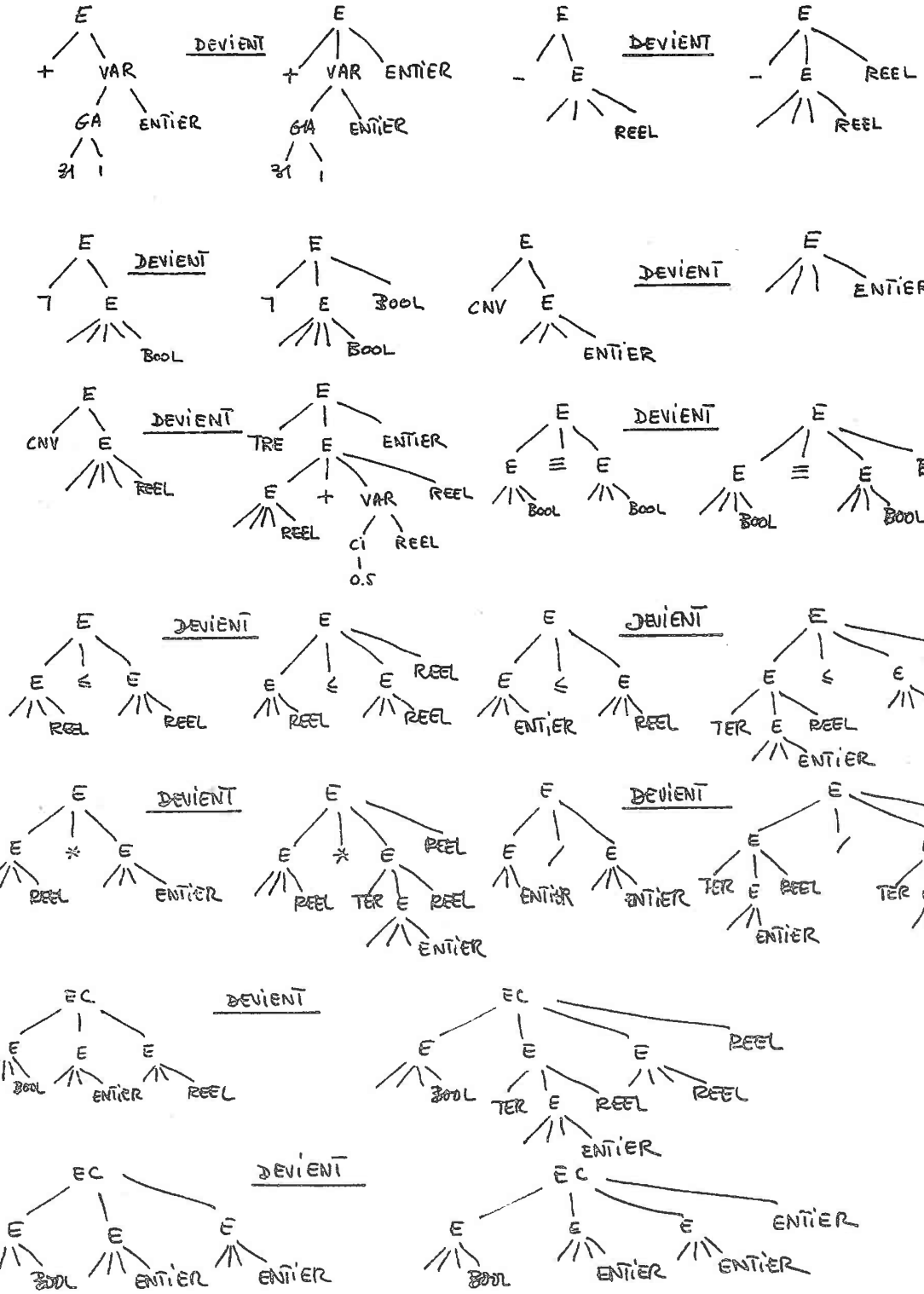


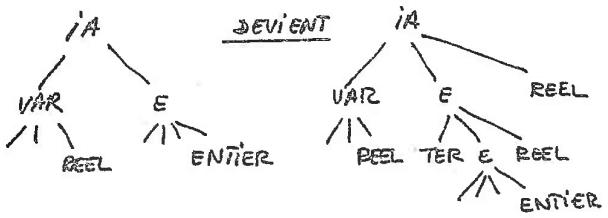


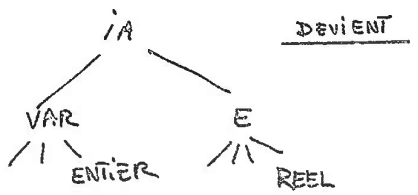
EXEMPLES DE REALISATIS DE TS

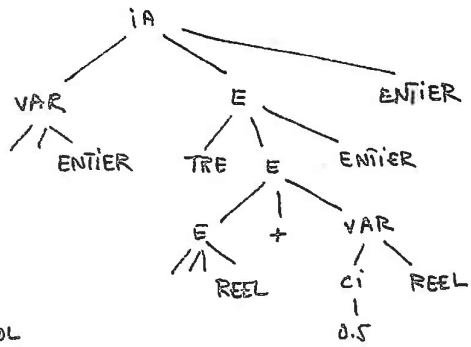


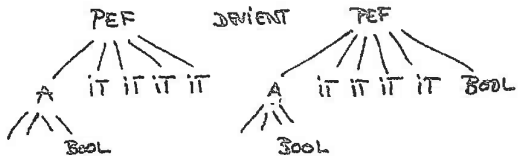
EXEMPLES DE RESULTATS DE TT

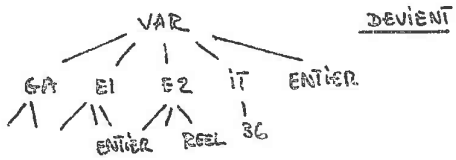




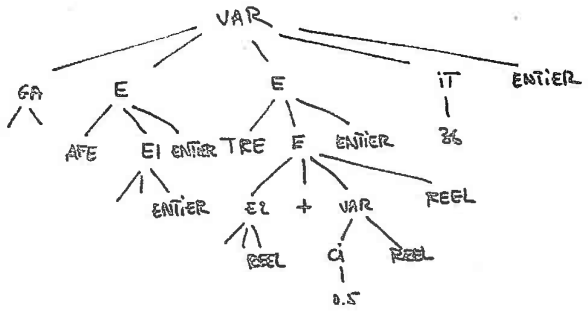




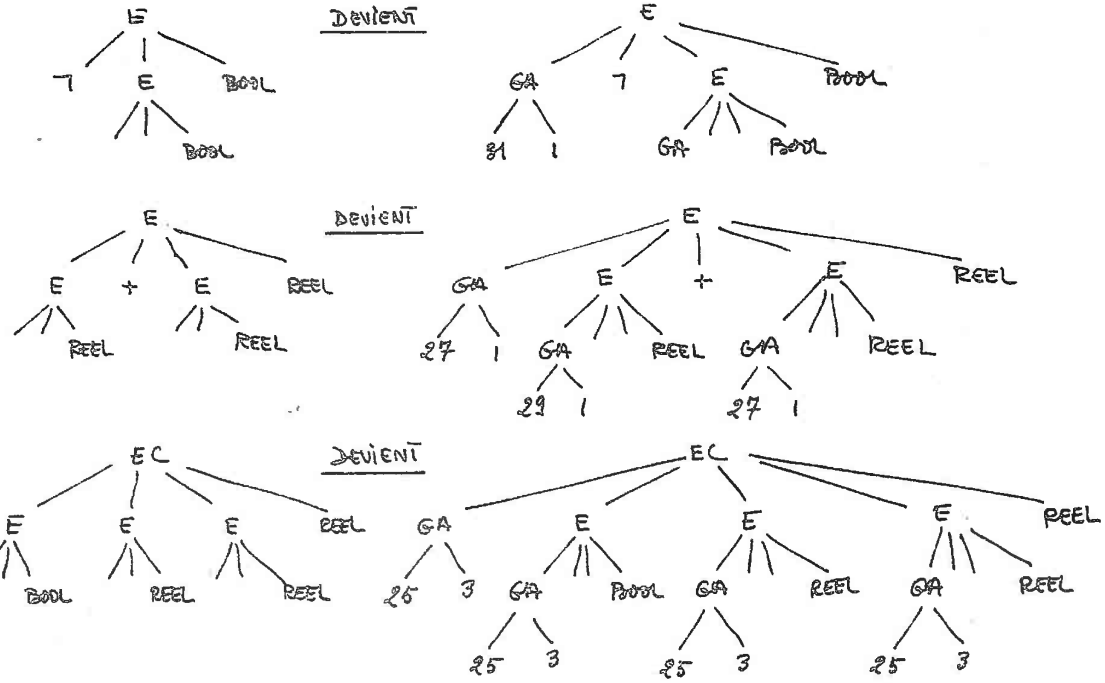




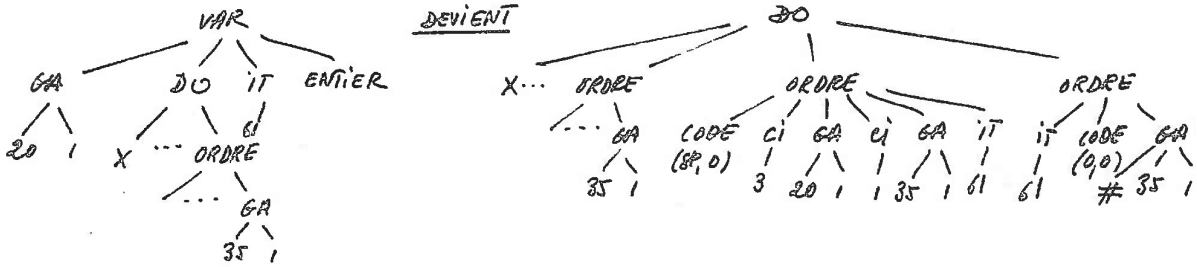
LES PSEUDO ARBORESCENCES
ENRACINEES PAR E1 ET E2
ETANT TRANSFORMEES PAR TAS

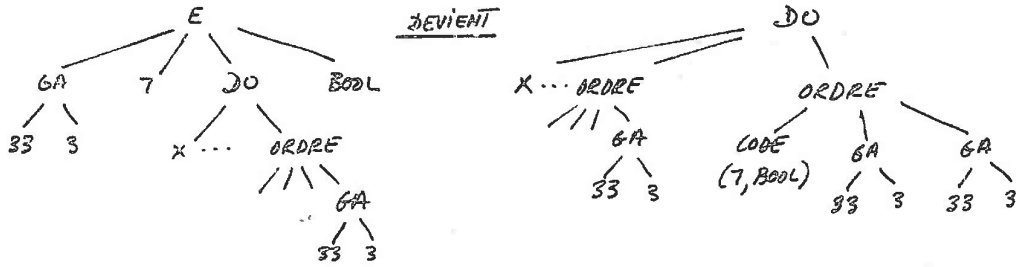


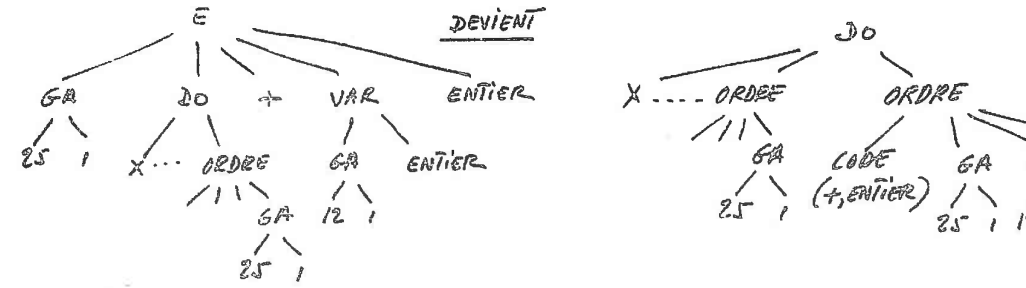
EXEMPLES DE RESULTATS DE TAI

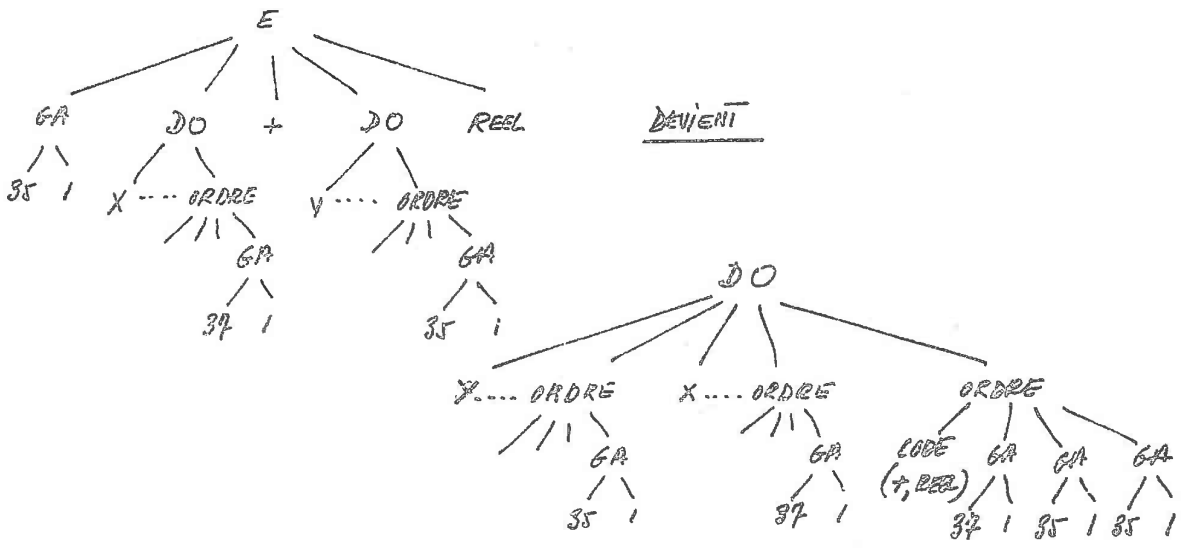


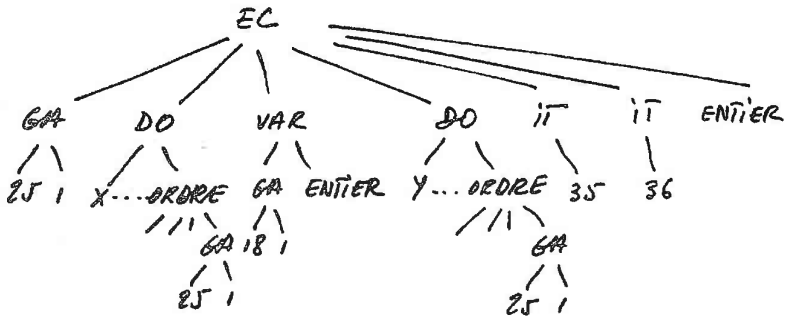
EXEMPLES DE RESULTATS DE B

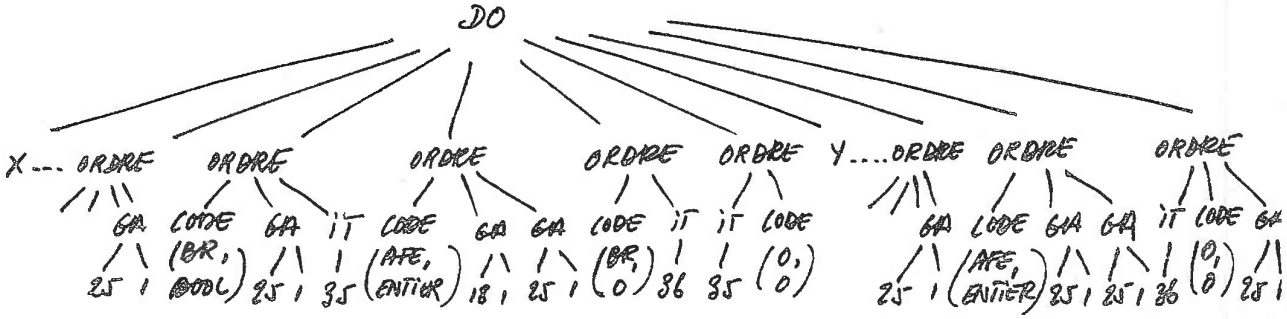


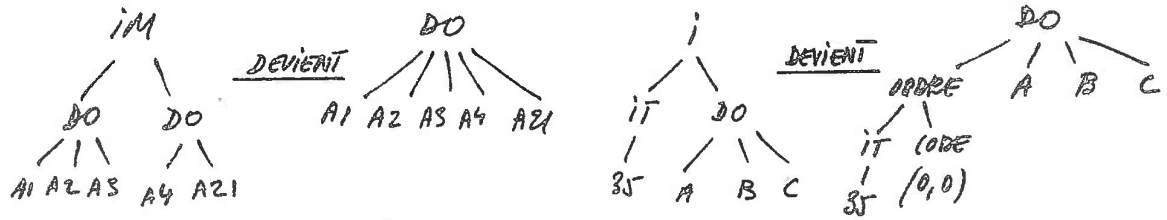


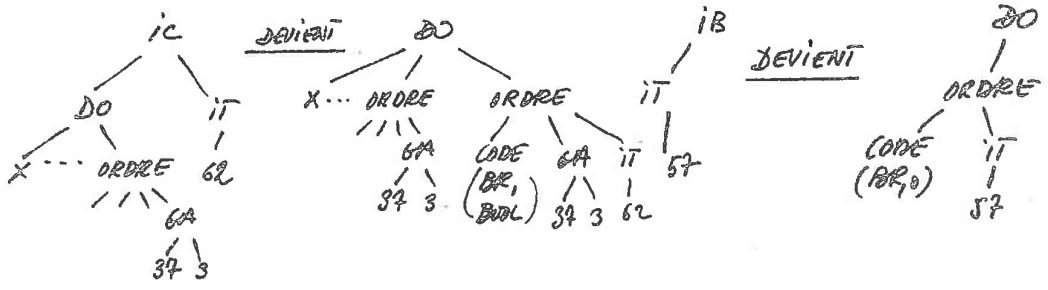


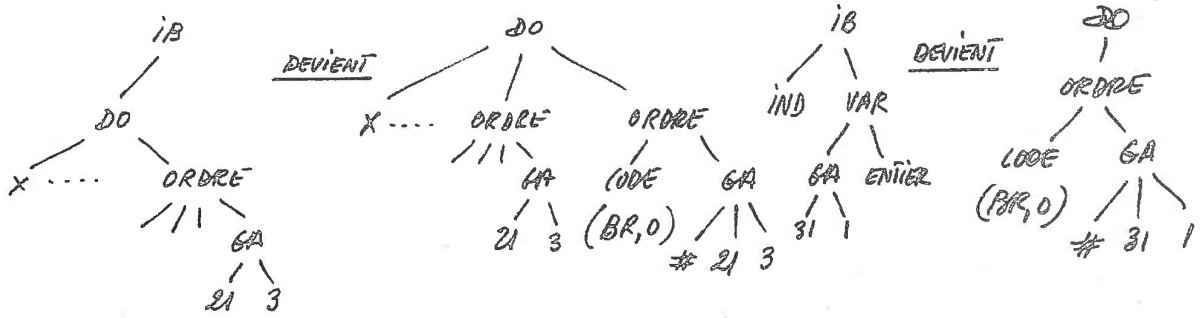


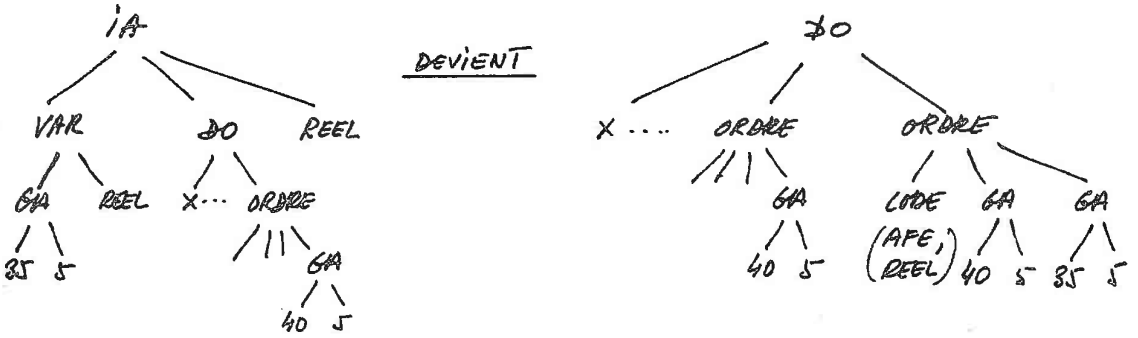

EC
GA
DO
VAR
DO
iT
iT
ENTIER
DEVIENT
25 1
X... ORDRE
GA ENTIER
Y... ORDRE
35
36
GA 18 1
25 1
25 1

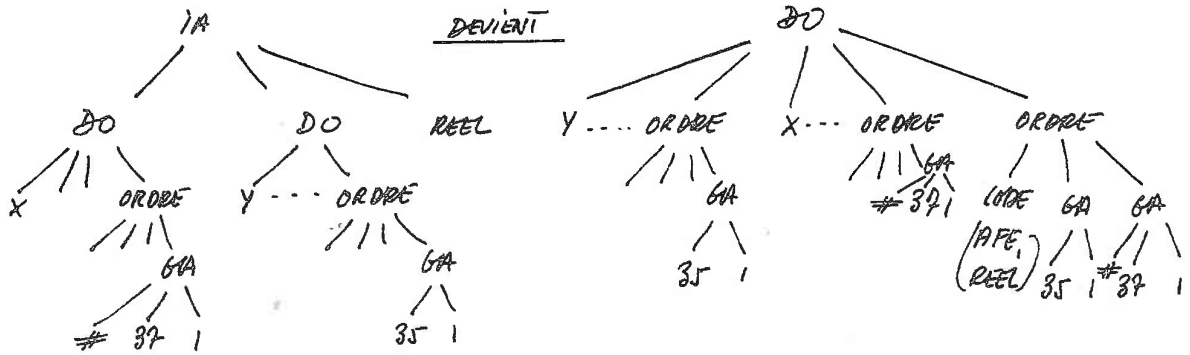

DO
X... ORDRE
ORDRE
ORDRE
ORDRE
ORDRE
Y... ORDRE
ORDRE
ORDRE
GA CODE GA iT CODE GA GA CODE iT iT CODE
25 1 (BR, BOOL) 25 1 35 (AFE, ENTIER) 18 1 25 1 (BR, 0) 36 35 (0, 0)
GA CODE GA GA iT CODE GA
25 1 (AFE, ENTIER) 25 1 25 1 36 (0, 0) 25 1

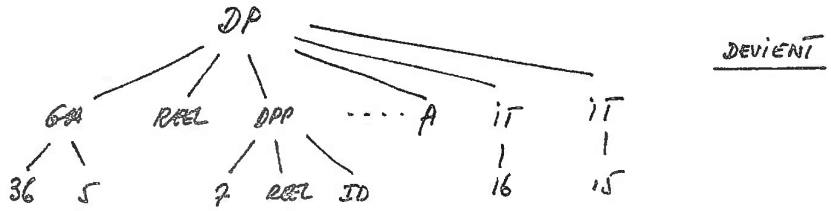

iM
DO DO
A1 A2 A3 A4 A21
DEVIENT
DO
A1 A2 A3 A4 A21
i
iT DO
35 A B C
DEVIENT
DO
ORDRE A B C
iT CODE
35 (0,0)

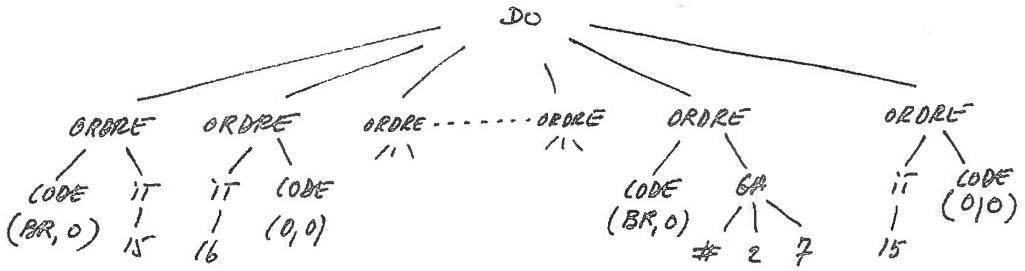

iC
DO iT
X... ORDRE 62
GA
37 3
DEVIENT
DO
X... ORDRE ORDRE
GA CODE GA iT
37 3 (BR, BOOL) 37 3 62
iB
iT
57
DEVIENT
DO
ORDRE
CODE iT
(BR,0) 57

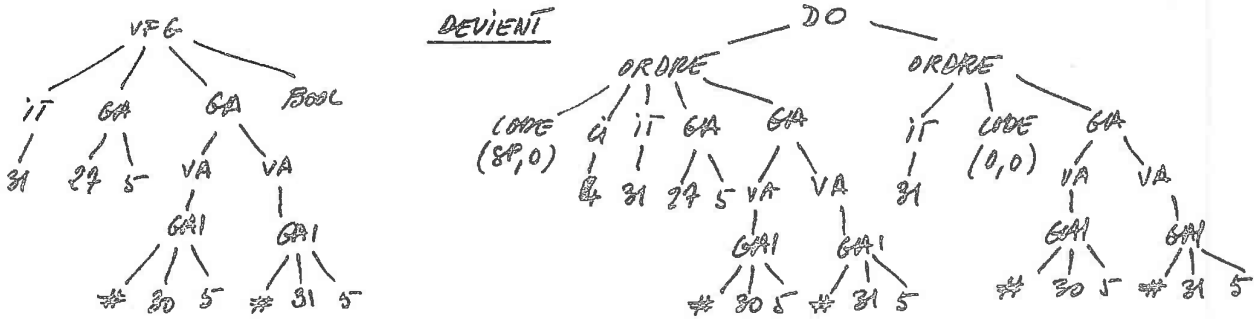

iB
DO
X.... ORDRE
GA
21 3
DEVIENT
DO
X.... ORDRE ORDRE
GA CODE GA
21 3 (BR,0) # 21 3
iB
IND VAR
GA GA ENTIER
31 1
DEVIENT
DO
ORDRE
CODE GA
(BR,0) # 31 1

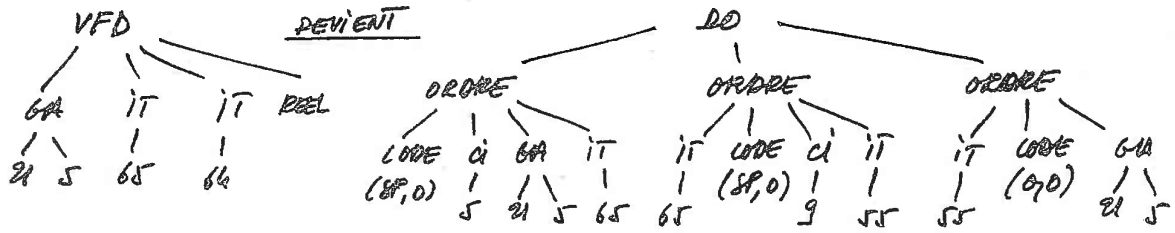

IA
VAR
DO
REEL
GA
REEL
X ...
ORDRE
35 5
GA
40 5
DEVIENT
DO
X ....
ORDRE
ORDRE
GA
CODE
GA
GA
40 5
(AFE, REEL)
40 5
35 5


IA
DO
DO
REEL
X
ORDRE
Y ...
ORDRE
GA
GA
37 1
35 1
DEVIENT
DO
Y ....
ORDRE
X ...
ORDRE
ORDRE
GA
37 1
CODE
GA
GA
35 1
(AFE, REEL)
35 1
37 1

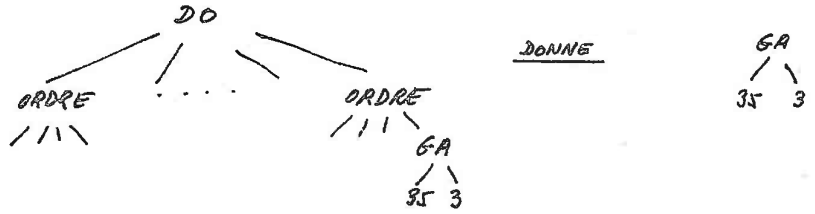

DP
GA
REEL
DPP
....
A
IT
IT
36 5
7 REEL ID
16
15
DEVIENT


DO
ORDRE
ORDRE
ORDRE ..... ORDRE
ORDRE
ORDRE
CODE
IT
IT
CODE
(BR, 0)
15
16
(0,0)
CODE
GA
(BR, 0)
2 7
IT
CODE
15
(0,0)

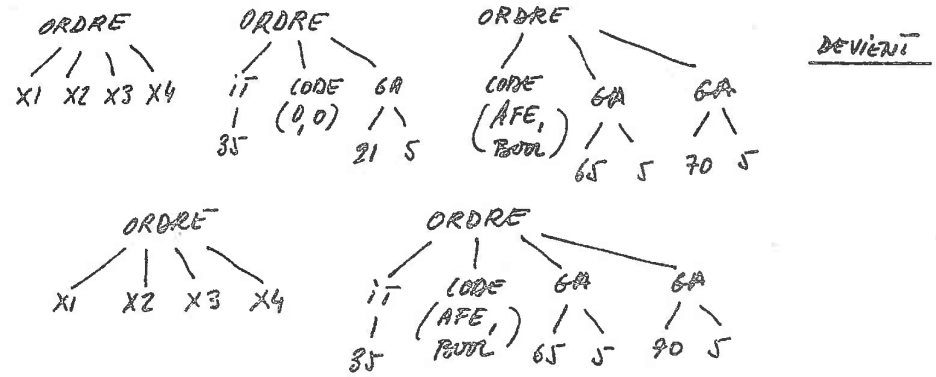

VFG
IT
GA
GA
BOOL
31 24 5
VA
VA
GA1
GA1
30 5
31 5
DEVIENT
DO
ORDRE
ORDRE
CODE
(8P, 0)
IT
GA
GA
IT
CODE
GA
4 31 24 5
VA
VA
31
(0,0)
VA
VA
GA1
GA1
GA1
GA1
30 5
31 5
30 5
31 5

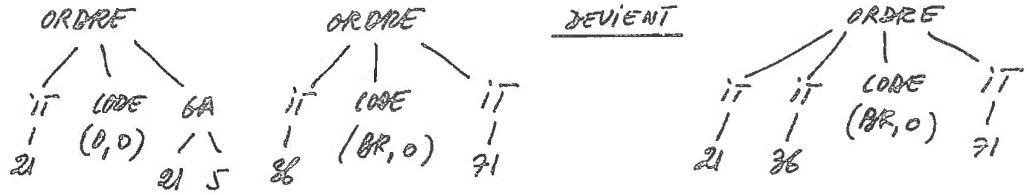


EXEMPLES DE RESULTATS DE DU

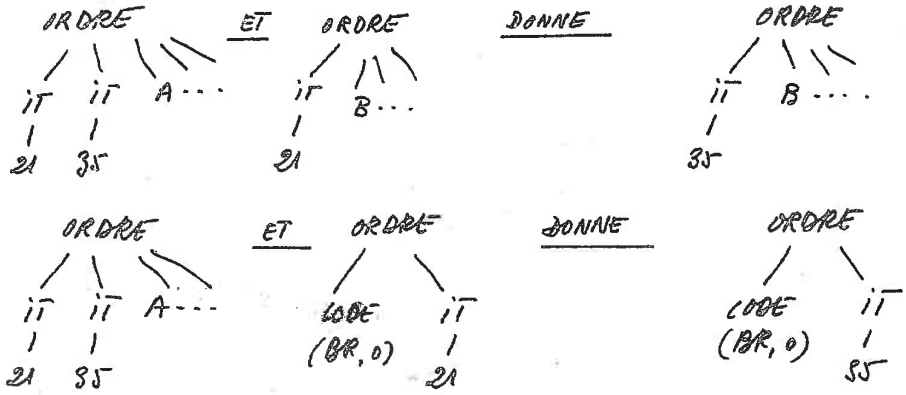


EXEMPLES DE RESULTATS DE TAS

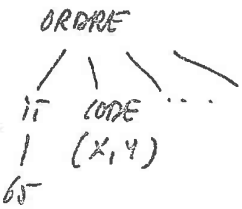




EXEMPLES DE RESULTATS DE ETD ET VER



EXEMPLES DE RESULTATS DE NET



V - CONCLUSION

La description d'un compilateur par une expression fonctionnelle mathématique nous a permis d'étudier, en détails et indépendamment les unes des autres, les diverses transformations effectuées par un compilateur sur un programme.

La rigidité de l'écriture fonctionnelle a cependant oeuvré au détriment de la simplicité de la description et de l'efficacité du compilateur décrit. Elle a conduit à des fonctions assez compliquées, parfois peu naturelles et à une multiplication des fonctions visant à en diminuer la complication.

En particulier, le refus des notions de variable et d'affectation dans le langage de description a entraîné une multiplicité de sous-expressions communes dans la description, ce qui se traduirait, à son exécution, par un considérable gaspillage de temps en opérations inutiles.

Ce refus a également compliqué la simulation des "compteurs" qui ont dû être placés en paramètres, (cf. IV.2 et IV.4). Il nous a surtout éloigné de l'esprit de la programmation, ce qui s'est traduit par un net allongement du temps nécessaire à l'écriture des fonctions. Il semble, en définitive, que l'appareillage mathématique utilisé ait été trop strict pour la description des compilateurs.

Remarquons, enfin, que certaines des fonctions déclarées récursives peuvent être implémentées efficacement, mais pas toutes. (Il s'agit des fonctions qui, appliquées à un produit de deux ramifications, ont, pour résultat, le produit des résultats de l'application de la fonction aux deux ramifications). Cela mène à reprendre les formes de récursivité autorisées dans les fonctions déclarées récursives, de manière à permettre, à la fois, un calcul efficace de ces fonctions et une implémentation peu coûteuse des ramifications.

Des études sont actuellement en cours pour remédier aux inconvénients cités, définir et implémenter un langage de description plus simple et plus efficace.

BIBLIOGRAPHIE

[1] L. BOLLIET - N. GASTINEL - P.J. LAURENT - 1964 - Un nouveau langage scientifique : Algol - Hermann - Paris (1964)

[2] J. FELDMAN - 1966 - A formal semantics for computer languages and its application in a Compiler-Compiler - Comn. ACM. 3-9 (janvier 1966)

[3] J. FELDMAN - D. GRIES - 1968 - Translator Writing Systems - Comn. ACM. 2-11 (février 1968)

[4] M. GRIFFITS - 1966 - Anatomy of a compiler - R.R.E. Memorandum - n° 2 296 (juin 1966)

[5] J.V. GARWICK - 1964 - GARGOYLE - A language for compiler writing - Comn. ACM 7 (janvier 1964)

[6] F.R.A. HOPGOOD - 1969 - Compiling Techniques - Elsevier (1969)

[7] J.M. LECLAIRE - 1970 - Définition de la syntaxe des langages de programmation - Thèse de spécialité mathématiques - Université de Nancy (mai 1970)

[8] J. Mc CARTHY - 1962 - LISP 1.5 - Programmers manual - Computation lab. report - MIT (1962)

[9] P. NAUR - 1963 - Revised report on the algorithmic language Algol 60 - Comn. ACM 6 (janvier 1963)

[10] C. PAIR - 1968 - Préliminaires à l'étude algébrique des ramifications - Communication soumise au Congrès 1968 de l'IFIP

[11] C. PAIR - 1969 - Sur des notions algébriques liées à l'analyse syntaxique - Exposé fait au Centre d'Automatique de l'Ecole des Mines - Fontainebleau (mars 1969)

[12] C. PAIR - A. QUERE - 1969 - Définition et étude des bilangages réguliers - Information and Control - 13 (mai 1968)

[13] A. QUERE - 1969 - Etude des ramifications et des bilangages - Thèse de spécialité mathématiques - Université de Nancy (juillet 1969)

[14] B. RANDELL - D.S. RUSSEL - 1964 - Algol 60 implementation - Academic Press (1964)

[15] J.C. REYNOLDS - 1965 - An introduction to the COGENT programming system - Proc. ACM - 20th Natl. Conf. (1965)

[16] D.V. SCHORRE - 1964 - META II : A syntax-oriented compiler writing language - Proc. ACM - 19th Natl. Conf. (1964).

NOM DE L'ETUDIANT : KHALIL Raymond

Nature de la thèse : Docteur Ingénieur

Vu, Approuvé

et Permis d'imprimer

NANCY, le 17 avril 1970

Le Doyen,

J. AUBRY