Etude de la syntaxe d' Algol - Application à la compilation

Georg Werner

N° d'ordre

T H E S E S

présentées à la

FACULTE DES SCIENCES DE GRENOBLE

pour obtenir

LE GRADE D'INGENIEUR - DOCTEUR

par

Georg WERNER

Dipl. Ing. Abtlg. Kulturtechnik

der Hochschule f. Bodenkultur, Vienne, Autriche

Première thèse :

"ETUDE DE LA SYNTAXE D'ALGOL

APPLICATION A LA COMPILATION"

Deuxième thèse :

PROPOSITIONS DONNEES PAR LA FACULTE

Thèses soutenues le juin 1964 devant la Commission d'Examen :

Jury { MM. J. KUNTZMANN Président
B. VAUQUOIS
N. GASTINEL } Examinateurs
J. ARSAC

PROFESSEURS SANS CHAIRE

M.	LACASE A.	THERMODYNAMIQUE
Mme.	KOFLER L.	BOTANIQUE
MM.	DREYFUS B.	THERMODYNAMIQUE
	VAILLANT F.	ZOOLOGIE ET HYDROBIOLOGIE
	GIRAUD P.	GEOLOGIE
	GIDON P.	GEOLOGIE ET MINERALOGIE
	ARNAUD P.	CHIMIE
	PERRET R.	SERVOMECANISMES
Mme.	LUMER I.	MATHEMATIQUES
Mme.	BARBIER M. J.	ELECTROCHIMIE
Mme.	SOUTIF J.	PHYSIQUE
MM.	BRISSONNEAU P.	PHYSIQUE
	COHEN J.	ELECTROTECHNIQUE
	DEPASSEL R.	MECANIQUE
	GASTINEL N.	MATHEMATIQUES APPLIQUEES

PROFESSEURS ASSOCIES

MM.	LUMER G.	MATHEMATIQUES
	HIGUCHI	BIOSYNTHESE DE LA CELLULOSE
	WAGNER	BOTANIQUE

MAITRES DE CONFERENCES

MM.	ROBERT A.	CHIMIE PAPETIERE
	ANGLES D'AURIAC P.	MECANIQUE DES FLUIDES
	BIAREZ J. P.	MECANIQUE PHYSIQUE
	COUMES A.	ELECTRONIQUE
	DODU J.	MECANIQUE DES FLUIDES
	DUCROS P.	MINERALOGIE ET CRISTALLOGRAPHIE
	GLENAT P.	CHIMIE
	HACQUES G.	CALCUL NUMERIQUE
	LANCIA R.	PHYSIQUE AUTOMATIQUE
	PEBAY-PEROULA J.C.	PHYSIQUE
	KAHANE	PHYSIQUE GENERALE
	DOLIQUE	ELECTRONIQUE
Mme.	KAHANE J.	PHYSIQUE
MM.	DEGRANGE C.	ZOOLOGIE
	GAGNAIRE D.	CHIMIE PAPETIERE
	RASSAT A.	CHIMIE SYSTEMATIQUE
	KLEIN J.	MATHEMATIQUES
	BETHOUX P.	MATHEMATIQUES APPLIQUEES
	POLOUJADOFF M.	ELECTROTECHNIQUE
	DEPOMMIER P.	PHYSIQUE NUCLEAIRE
	DEPORTES C.	CHIMIE
	BARRA J.	MATHEMATIQUES APPLIQUEES
Mme.	BOUCHE L.	MATHEMATIQUES
MM.	PERRIAUX J.	GEOLOGIE
	SARROT-REYNAULD J.	GEOLOGIE
	CAUQUIS G.	CHIMIE GENERALE
	LABBE A.	BOTANIQUE
	BONNET G.	PHYSIQUE GENERALE
	BARNOUD F.	BIOSYNTHESE DE LA CELLULOSE
Mme.	BONNIER M.J.	CHIMIE

MAITRES DE CONFERENCES ASSOCIES

MM.	ISHIKAWA Y.	MAGNETISME
	QUATTROPANI	THERMODYNAMIQUE

A Monsieur le Professeur KUNTZMANN,
Directeur de l'Institut de Mathématiques Appliquées de Grenoble et

A Monsieur le Professeur GASTINEL,
en respectueux témoignage de ma reconnaissance,

A Monsieur le Professeur VAUQUOIS,
Directeur du Centre d'Etudes pour la Traduction Automatique et

A Monsieur ARSAC,
Directeur du Service de Calcul au Laboratoire de Meudon,
qui m'ont fait l'honneur d'être membres du jury.

Je remercie Monsieur BOLLIET, Ingénieur au C.N.R.S.
et le personnel du Laboratoire de Calcul qui ont bien voulu m'aider
à réaliser ce travail.

TABLE DES MATIERES

I - P R E S E N T A T I O N

1.1. - DEFINITION D'UNE CLASSE DE LANGAGES

Un langage formel L_0 [2,3] est défini par un alphabet fini $A = \{a_0, a_1, \ldots, a_n\}$ et par un ensemble fini de règles syntaxiques, la grammaire du langage. On appelle chaîne sur A (de longueur k, $k \geqslant 1$) une séquence ordonnée de k éléments (distincts ou non) de A. La concaténation * de deux chaînes sur A donne encore une chaîne sur A.

Un langage formel L_0 est une certaine partie de l'ensemble de chaînes sur A ; on parle aussi des phrases du langage. Les chaînes de longueur finie appartenant au langage forment une classe d'expressions formelles dites expressions bien formées ou formules bien formées.

On dit qu'une grammaire génère un langage lorsqu'elle fournit un procédé de formation de toutes les expressions formelles bien formées et uniquement de celles-ci. Les définitions suivantes préciseront davantage la notion de règle syntaxique, grammaire et langage formel.

* Cf. § 2.2

DEFINITION 1.1.1.

Soit X, Y,... des variables prenant leurs valeurs dans l'ensemble des chaînes sur un certain vocabulaire V ; (P_i, Q_i) $(i = 1,\ldots,m)$ des couples sélectionnés de chaînes sur V de L_0, le prédicat binaire $\mathcal{P}(X,Y)$ est vrai pour $X = X_0$ et $Y = Y_0$ donnés si et seulement si il existe des chaînes $\overline{P}$, $\overline{Q}$ (éventuellement vides) et un indice $i \in \{1,\ldots,m\}$ tel que :

$$X_0 = \overline{P}\ P_i\ \overline{Q}$$
$$\text{et} \quad Y_0 = \overline{P}\ Q_i\ \overline{Q}$$

On écrit $\overline{P}\ P_i\ \overline{Q} \rightarrow \overline{P}\ Q_i\ \overline{Q}$ et on appelle $\mathcal{P}(X,Y)$ la production définie par (P_i, Q_i).

Soit $\mathcal{P}(X,Y)$ une production et soit X_0 et Y_0 des chaînes telles que $\mathcal{P}(X_0, Y_0)$ est vrai. Dans ce cas on dit que Y_0 est une conséquence de X_0 par rapport à $\mathcal{P}(X,Y)$.

DEFINITION 1.1.2.

Une grammaire $\mathcal{G}(L_0)$ est définie par un vocabulaire V, un nombre fini de productions qui constituent l'ensemble de règles syntaxiques de la grammaire et par un élément distingué de V appelé axiome.

La définition d'un langage formel se fait dans une métalangue. Son vocabulaire comprend un nombre de termes M (simples ou composés, souvent empruntés à une langue courante) dont chacun désigne une famille de mots du langage objet L_0 et un symbole, "$\rightarrow$", dit symbole de réécriture [3] . Ceci permet la définition constructive suivante :

DEFINITION 1.1.3

Un langage L_0 est défini par un alphabet fini A, un ensemble de termes de la métalangue M, un élément distingué de M, l'axiome et par un certain nombre de règles syntaxiques, c'est-à-dire par un ensemble fini de correspondances univoques entre certaines chaînes de longueur n, $n \leq N$, sur A U M et éléments de M.

L' ensemble A U M prend alors le nom de vocabulaire V, les éléments de l'alphabet A s'appellent éléments du vocabulaire terminal V_T et les termes de la métalangue constituent le vocabulaire non - terminal V_N.

Les règles de la syntaxe s'écrivent $X \rightarrow x$, $X \in V_N$, x étant une chaîne sur A U M.

(les règles de la définition 1.1.3. sont moins générales que celles de la définition 1.1.1. et correspondent aux langages dits "context-free").

Soit $\mathcal{G}$ une grammaire, V son vocabulaire et P l'ensemble de ses productions.

La relation binaire $Y \Rightarrow X$ est vérifiée entre deux chaînes sur V : $Y = Y_0$ et $X = X_0$, si $X_0 = Y_0$ ou s'il existe une séquence finie de chaînes

$$Y_0 = Z_0, Z_1, \ldots\ldots, Z_k = X_0 , \qquad k \geq 1$$

telle que Z_i est une conséquence de Z_{i-1} par rapport à une des productions de $\mathcal{G}$, et ceci pour tout i, $1 \leq i \leq k$.

Lorsque Y est un élément de V (c'est-à-dire une chaîne de longueur 1) X est dit un mot pour l'élément syntaxique Y. La relation binaire $\Rightarrow$ ("est un mot pour") est donc une relation transitive et réflexive entre les chaînes d'un langage.

DEFINITION 1.1.4

On appelle dérivation dans une grammaire une séquence finie de mots formés sur le vocabulaire de $\mathcal{G}$

$$X_0, X_1, \ldots, X_k$$

tel que X_0 l'axiome de $\mathcal{G}$ et que pour tout i, $1 \leq i \leq k$, X_i est une conséquence de X_{i-1} par rapport à une production de $\mathcal{G}$.

Un exemple d'un langage formel correspondant à la définition 1.1.3 est celui des nombres décimaux :

Exemple 1.1.1

Grammaire $\mathcal{G}$: $A = \{0,1,2,\ldots,9,.,+,-\}$
$V = \{A ; D, ES, FD, NDS, ND\}$, axiome ND (nombre décimal)
productions (règles syntaxiques)

a) D → 0 D → 1, ... , D → 9 D ... digit
b) ES → D ES ... entier sans signe
c) ES → ESD
d) FD → .ES FD ... fraction décimale
e) NDS → ES
f) NDS → FD NDS ... nombre décimal sans signe
g) NDS → ES FD
h) ND → +NDS ND ... nombre décimal
k) ND → -NDS

Une dérivation d'un mot appartenant à ce langage est donnée par

axiome	ND	Production appliquée :
	- NDS	k)
	- ESFD	g)

- ES.ES	d)
- D . D	2 fois b)
- 1 . 7	2 fois une des dix règles a)

On distingue plusieurs classes de langages formels suivant la structure de leurs règles syntaxiques [3]. Soit $P \rightarrow Q$ une règle,

$A \in V_N$ et B, X, Y des chaînes sur V, B non-vide et supposons que P et Q puissent être écrits comme

$$\left.\begin{array}{l} P = XAY \\ Q = XBY \end{array}\right\} \text{ c'est-à-dire la règle est } XAY \rightarrow XBY.$$

Une grammaire qui possède uniquement des règles de cette forme est une grammaire de <u>type 1</u> (ou "context-dependent").

Lorsque $X = Y = \Lambda$ (Λ la chaîne vide) pour toutes les règles d'une grammaire, c'est-à-dire lorsque toutes les règles sont de la forme $A \rightarrow B$ où A est un seul élément de V_N et B une chaîne sur V non-nulle, on parle d'une grammaire (et d'un langage) du <u>type 2</u> ou "context-free" (langages L_2).

Dans un langage L_2 tous les symboles de V_N peuvent être réécrits indépendamment du "contexte" c'est-à-dire des chaînes voisines.

Un langage dont toutes les règles sont de la forme

$A \rightarrow Ba$ avec $A, B \in V_N$ et $a \in V_T$

$A \rightarrow a$

s'appelle langage de <u>type 3</u> ou d'états finis (cf.ex.1.1.1).

1.2. - LES LANGAGES ALGORITHMIQUES

La grammaire d'un langage formel et les dérivations des formules ou phrases bien formées ont été définies dans le paragraphe précédent. Un tel langage s'appelle aussi système syntaxique lorsqu'il s'agit d'un langage artificiel. Un langage algorithmique (LA) est un système sémantique. Il possède les mêmes règles que le système syntaxique mais on attache un sens, une signification aux éléments du langage. On fait d'ailleurs souvent allusion à la signification des éléments du vocabulaire par un choix particulier de leurs noms (voir exemple 1.1.1).

Les règles syntaxiques d'un langage algorithmique permettent d'obtenir la dérivation d'une formule bien formée. On a vu qu'une dérivation détermine, par définition, la suite ordonnée des productions (c'est-à-dire des règles syntaxiques) appliquées dans les transformations successives des mots qui forment la dérivation. On possède donc avec la dérivation d'un mot une succession de métavariables dont chacune désigne une famille de chaînes sémantiquement équivalentes.

La dérivation d'un mot détermine donc implicitement la signification de ce mot. L'établissement de la dérivation d'une chaîne donnée s'appelle analyse syntaxique.

Exemple 1.2.1

Grammaire $\mathcal{G}$: $A = \{0, 1, \ldots, 9, ., \uparrow, +, -, x, /,), (\}$
$V = \{A ; D, ES, FD, NDS, ND, P, F, T, EX\}$, élément distingué : EX

Les règles sont celles de l'exemple 1.1.1. à l'exception des règles h) et k) qui sont remplacées par ND→NDS, plus les règles suivantes :

l) P→ND m) P→(ES) n) F→P o) F→F↑P
p) T→F r) T→TxF s) T→T/F t) EX→T
u) EX→EX + T v) EX→EX - T

Considérons le mot suivant : (7 - 3)/(2 x 1)↑2
Une dérivation de cette expression (écrite dans l'ordre inverse) est :

Syntaxe	Productions appliquées et signification des chaînes
(7 - 3)/(2 x 1)↑2	a) : former les digits 7,3,2,1,2
(D - D)/(D x D)↑D	b) : former des entiers s.signe corresp.
(ES-ES)/(ESxES)↑ES	e) : transformer ces entiers en nombres déc.
(NDS-NDS)/(NDSxNDS)↑ES	k) : ——
(ND-ND)/(NDxND)↑ND	l) : les opérandes ont des valeurs fixées
(P - P)/(P x P)↑P	n) : ——
(F - F)/(F x F)↑P	p) : ——
(T - T)/(T x F)↑P	t) : ——
(EX - T)/(T x F)↑P	r) : remplacer 2.0 x 1.0 par 2.0
(EX - T)/(T)↑P	u) : remplacer 7.0 - 3.0 par 4.0
(EX) / (T)↑P	t) : ——
(EX) / (EX)↑P	m) : les opérandes ont des valeurs fixées
P / P ↑P	n) : ——
F / F ↑P	p) : ——
T / F ↑P	

T / F
T
EX

o) : remplacer 2.0 par 4.0
s) : remplacer 4.0/4.0 par 1.0
t) : ——

DEFINITION 1.2.1. - REGLE DE REMPLACEMENT

A) Au niveau de la syntaxe on remplace dans un mot donné du langage une occurrence (ou plusieurs occurrences, mais pas nécessairement toutes) du membre droit d'une production par son membre gauche (éventuellement dans un contexte donné).

B) En interprétation, c'est-à-dire au niveau de l'algorithme on remplace (ou évalue) la fonction $F(a_1,...,a_n)$ dans laquelle $(a_1,...,a_n)$ est une liste de variables (ou de constantes) déjà évaluées, par sa valeur F_a dans une formule dont $F(a_1,...,a_n)$ est composante.

La première règle de remplacement ne suffit pour l'analyse syntaxique d'un langage que lorsque celui-ci est complétement spécifié par un vocabulaire et par un nombre de productions. Les langages algorithmiques ne sont en général pas spécifiés complètement par une grammaire d'un langage CF. Les chaînes d'un LA ont deux sortes de constituants :

1) Symboles particuliers appartenant à l'alphabet, donc un nombre limité (par exemple : opérateurs, séparateurs, parenthèses, etc) de symboles distincts.

2) Noms de quantités telles que : constantes, variables, listes de variables, fonctions, etc ; appartenant au vocabulaire V_N, donc un nombre illimité de chaînes distinctes.

Les symboles 1) sont spécifiés une fois pour toutes dans le système sémantique ; chacune des quantités 2) n'est définie que pour un certain algorithme (une phrase d'un LA) ou pour une partie de celui-ci. Les définitions (explicites) font partie d'une phrase d'un LA (les déclarations en Algol 60) et comportent une spécification syntaxique des noms qu'elles introduisent. Un langage algorithmique est donc un langage mixte de chaînes descriptives (déclaratives) et de chaînes du langage objet proprement dit (qui peuvent être interprétées comme des commandes de transformations des quantités variables).

Les chaînes dont la spécification syntaxique est donnée dans les phrases mêmes d'un LA ne sont spécifiées que de façon ambigüe par les productions de la grammaire. Une analyse complète d'une chaîne donnée par l'ensemble des règles d'un LA ne peut-être obtenue que si l'on substitue les spécifications déterminées par l'analyse syntaxique des chaînes déclaratives pour les objets spécifiés dans les chaînes de commandes. Le résultat d'une telle opération est une chaîne dans laquelle les chaînes déclaratives sont remplacées par le terme de la métalangue (un élément de V_N) que leur analyse syntaxique détermine et dans laquelle tous les éléments sont spécifiés de façon non ambigüe. Cette chaîne

peut être considérée comme appartenant à un langage CF dérivé du LA et peut être analysée par une méthode appropriée (voir § 2.2). Les règles de construction d'un tel langage CF "local", c'est-à-dire valable dans la portée d'un ensemble de chaînes déclaratives, seront appelées règles de substitution.

DEFINITION 1.2.2. - REGLE DE SUBSTITUTION

A toute occurrence d'une même chaîne on substitue l'élément du vocabulaire V_N qui lui est attribué dans la chaîne déclarative correspondante (Cf. ex. 2.1.1).

1.3. - A L G O L 60

Algol 60 [1] rentre dans la classe des langages algorithmiques (LA) défini en § 1.2. Les productions sont données en forme normale de Backus (avec les connecteurs métalinguistiques "::=" et " | " et la désignation des éléments de V_N par des crochets "⟨" et "⟩").

On remarque que la structure locale de Algol est celle d'un langage L_2(CF) : les chaînes descriptives du langage spécifient la syntaxe de tous les mots pour ⟨identificateur⟩ (- et des mots pour⟨entier sans signe⟩lorsque ceux-ci correspondent à des étiquettes). Les chaînes font partie d'un mot pour ⟨déclaration⟩ (- on considère ⟨identificateur⟩ : et ⟨entier sans signe⟩ : comme "déclaration" d'étiquette). Chacune des spécifications est valable à l'intérieur d'un segment de la

chaîne donnée ; ces segments s'appellent niveaux de nomenclature. La décomposition d'une chaîne en niveaux définit les règles de substitution (voir § 2.1.).

Les niveaux de nomenclature sont déterminés par les règles suivantes :

R1 : Tout programme correct admet une partition en niveaux de nomenclature (NN).

R2 : Un mot pour ⟨bloc⟩ est un NN.

R3 : Si P est un programme correct, il est aussi un NN

Soit $P = X \varphi Y$ $\quad X \neq \Lambda$, $\quad Y \neq \Lambda$ (la chaîne vide)

et $\varphi = UV$ $\quad$ U étant un mot pour ⟨déclaration⟩

a) si $U = \Lambda$ et V ne contient aucun mot pour ⟨déclaration⟩ , alors P est un (le seul) NN.

b) si $U = \Lambda$ et V contient un mot pour ⟨déclaration⟩ on peut trouver une autre partition

$P = \bar{X} \bar{\varphi} \bar{Y}$ $\quad \bar{\varphi} = \bar{U}\bar{V}$, tel que $\bar{U} \neq \Lambda$

et P est un NN qui contient un mot pour ⟨bloc⟩ , c'est-à-dire d'après R2 un autre NN.

R4 : Soit $\bar{B}$ et $\bar{\bar{B}}$ des mots pour ⟨bloc⟩ (donc des NN)

Si B admet une partition de la forme $B = X \bar{B} Y$ où X et Y sont des chaînes non-vides, alors $\bar{B}$ est un bloc contenu dans le bloc B.

Dans ce cas, si $B = \bar{X} \bar{B} \bar{Y}$ est une partition de B

et $B = \bar{\bar{X}} \bar{\bar{B}} \bar{\bar{Y}}$ est une autre partition de B

alors $\overline{B}$ et $\overline{\overline{B}}$ sont soit disjoints, soit inclus l'un dans l'autre.
R4 détermine un ordre ramifié, ou structure arborescente sur les blocs (et plus particulièrement sur les niveaux de nomenclature).

Certains identificateurs ne sont pas obligatoirement spécifiés et demandent une règle de substitution (c'est-à-dire de spécification de leur syntaxe) particulière. Ce sont les identificateurs énumérés dans la liste des paramètres formels des déclarations de procédure. Celles-ci sont toujours écrites dans la forme :

$$\varphi(x_1, x_2, \ldots, x_n) \quad F(x_1, x_2, \ldots, x_n)$$

où φ est une chaîne de spécification syntaxique qui désigne une certaine chaîne comme ⟨identificateur de procédure⟩ ,

$(x_1, x_2, \ldots, x_n)$ une liste de paramètres formels

et $F(x_1, x_2, \ldots, x_n)$ un mot arbitraire pour ⟨corps de procédure⟩

Celui-ci est toujours un niveau de nomenclature pour les paramètres formels.

Tout identificateur qui n'est pas énuméré dans une liste de paramètres formels désigne une instance de substitution pour l'élément de V_N qui lui est attribué dans une des chaînes de spécification syntaxique.

Mais les déclarations ont deux fonctions. La première est celle d'une spécification locale de la syntaxe, c'est-à-dire relative à un certain niveau de no-

menclature.

Soit I l'ensemble des identificateurs permis ; toute déclaration établit la correspondance entre les éléments d'un ensemble fini :

$$\{I_k\} = \{i_{k_1}, i_{k_2}, \ldots, i_{k_n}\}, \quad I_k \subset I$$

d'identificateurs d'un certain niveau de nomenclature NN et entre un élément $v_k \in V_N$. On dit que $i_{k_1}, i_{k_2}, \ldots i_{k_n}$ sont locaux à ce niveau.

La deuxième fonction des déclarations est la définition d'une règle de valuation pour les quantités variables. Dans le langage interprété une variable correspond à un mot machine, toute valuation implique donc une adresse (d'une mémoire) ou plus généralement une fonction d'adresse, c'est-à-dire une règle pour le calcul d'une adresse. Une règle de valuation définit une certaine fonction d'adresse pour tout couple correspondant (v_k, i_k) ainsi qu'un domaine dans lequel la variable peut prendre ses valeurs. Une telle règle peut contenir une prescription pour le calcul des valeurs que la variable peut effectivement prendre dans le domaine donné.

1.4. - EXPOSE DES PROBLEMES A RESOUDRE

La compilation d'un LA (et de Algol 60 en particulier) est essentiellement basée sur un algorithme d'analyse et de reconnaissance qui décide, pour toute chaîne sur V_T , si cette chaîne appartient ou n'appartient pas au langage et qui donne, dans le cas d'une réponse affirmative, la dérivation complète correspondant à la chaîne donnée (dans le sens de définition 1.1.4.) ainsi que la suite ordonnée des règles employées successivement dans la dérivation.

Le compilateur effectue d'abord les substitutions pour les identificateurs en testant la concordance et la validité des schémas de substitution. La chaîne dans laquelle les substitutions ont été faites étant donnée, les mots successifs de la dérivation sont obtenus par l'application de la règle de remplacement (c'est-à-dire par les règles de réécriture données par l'ensemble des productions).

Une chaîne dérivable de l'axiome par l'application successive des substitutions et des remplacements est appelée programme correct (d'un point de vue purement syntaxique).

On démontre (voir par exemple [11]) que les algorithmes d'analyse dont on vient de parler peuvent être réalisés par une classe d'automates que l'on

appelle automates à pile ou automates PDS (push-down-store). La théorie de ces automates peut être liée à celle des machines de Turing [5] ; une correspondance plus directe entre une forme de dérivations dans les grammaires des langages CF et le calcul des automates à pile sera développée en paragraphe 2.2.

Pour tout programme correct le compilateur établit une liste de règles de valuation pour l'ensemble des variables du programme. Il donne la suite des productions successivement employées dans les opérations de remplacement et il définit l'interprétation de la suite des chaînes correspondantes par une suite de codes machine. Cette interprétation dépend de la sémantique du langage et de la structure de la machine donnée. Elle constitue la partie particulière du compilateur qui dépend de la machine pour laquelle sont destinés les programmes qu'il produit (et qui peut être différente de la machine sur laquelle il est réalisé).

On distingue souvent entre compilateurs interprétatifs ou interpréteurs et compilateurs génératifs qui génèrent un programme en langage machine. Après ce qui a été dit ils peuvent être définis ainsi :

Définition 1.4.1

Un compilateur qui commande immédiatement l'exécution de l'ensemble des codes machine correspondant à la partie droite d'une production qu'il remplace dans la chaîne donnée est un compilateur purement interprétatif ou interpréteur.

Définition 1.4.2.

La génération d'un programme en langage machine est l'assemblage des codes machine correspondant à la suite ordonnée des productions employées dans la dérivation et des codes machine correspondant aux règles de valuation.

On remarque que cette distinction n'affecte pas la partie du compilateur qui effectue l'analyse de la syntaxe d'un programme donné. Cette partie syntaxique du compilateur ne dépend que de la grammaire du langage et on peut, pour une classe de langages comme L_2 ou éventuellement LA, trouver une méthode générale afin de l'établir automatiquement à partir des règles grammaticales données.

II - METHODES GENERALES

2.1. REGLES DE SUBSTITUTION DANS UN LANGAGE ALGORITHMIQUE

Les expressions formelles d'un LA sont générées à partir d'un ensemble d'éléments initiaux $\{A\}$ par les règles de la grammaire du langage. Ces règles consistent en un nombre de productions dont on suppose qu'elles sont du type 2(CF), éventuellement en forme normale de Backus, et en un nombre de règles qui spécifient le langage en tant que langage mixte, elles précisent le rôle des chaînes déclaratives. Ces règles ne peuvent pas être exprimées en forme d'un système de productions et les chaînes d'un LA ne sont par conséquent pas les formules d'un langage CF. On peut distinguer trois classes d'expressions d'un LA et une chaîne contient en général les expressions de ces trois classes.

1) - <u>Expressions dont la syntaxe est spécifiée de façon ambigüe par les productions de la grammaire</u> ;
Leur nombre n'est pas limité dans une phrase du langage, mais elles doivent être introduites et spécifiées dans une des chaînes descriptives, elles sont appelées identificateurs par la suite.

2) - Expressions contenant les identificateurs et un nombre d'éléments de V_T, appelées par la suite formules.

3) - Expressions qui spécifient la syntaxe de chacun des identificateurs d'une suite de formules, appelées chaînes déclaratives.

On montre comment on peut définir un langage CF particulier et valable pour les formules d'une certaine phrase d'un LA par l'analyse des chaînes déclaratives de cette phrase. L'ensemble de règles d'analyse des chaînes déclaratives et de spécification syntaxique des identificateurs, défini pour un certain LA (c'est-à-dire valable pour toute chaîne appartenant à ce langage) sera appelé ensemble de règles de substitution. Toute chaîne particulière d'un langage détermine un certain schéma de substitution qui doit être compatible avec les règles générales.

Exemple 2.1.1. $\mathcal{U}$ = (grammaire d'Algol)

begin { D { Booleen B;integer X,Y;array T[1:3,1:3]; integer procedure F(U,V) ; F:= U + V ;

F { T[X,Y] := if B then X else F(X,X+Y) ; --- end

La chaîne de cet exemple contient les identificateurs B, X, Y, T, F, U et V, les chaînes déclaratives D et les formules F. Les chaînes déclaratives introduisent les identificateurs en définissant leur syntaxe. Cette spécification de la syntaxe des identificateurs résulte de la construction des chaînes déclaratives correspondantes :

⟨déclaration⟩ → ⟨déclaration de type⟩ → ⟨type⟩⟨liste de types⟩ → ⟨type⟩ ⟨variable simple⟩

ou bien ⟨type⟩⟨variable simple⟩,⟨liste de type⟩

Un schéma de substitution peut être établi par l'analyse syntaxique des chaînes déclaratives. Dans l'exemple 2.1.1. les identificateurs B, X et Y sont donc spécifiés comme⟨variable simple⟩ , une analyse des autres chaînes déclaratives donne les spécifications de T et de F ; U et V par contre, qui sont les paramètres d'une procédure ne sont pas spécifiés et leur syntaxe doit être déterminée par leur "contexte" ou par les paramètres effectifs correspondants. Les chaînes déclaratives donnent l'information suivante :

⟨variable simple⟩ : B, X, Y
⟨identificateur de tableau⟩ : T
⟨identificateur de procédure⟩: F

La substitution de ces informations à la place des chaînes spécifiées dans les formules F donne :
⟨ident. de tableau⟩[⟨var.simple⟩,⟨var.simple⟩] : =if⟨var. simple⟩ then ⟨var.simple⟩ else ⟨id.de procéd.⟩(⟨var.simple⟩, ⟨var. simple⟩ + ⟨var. simple⟩)

Cette chaîne peut être considérée comme faisant partie d'un langage CF particulier défini par les régles non ambigües d'Algol et par un nombre de règles supplémentaires de la forme :

⟨variable simple⟩ ::= B|X|Y etc...

Définition 2.1.1.

Soit $\{I\}$ la famille des mots dont chacun désigne un identificateur,

$I \in V_N$ et $i_{k_1}, i_{k_2}, \ldots \in \{I\}$
$V_k \in V_N$, tel que $V_k \Rightarrow I$,

c'est-à-dire I est un mot pour V_k (k = 1,2,...) par les règles de $\mathcal{U}$, alors une correspondance
sont des V_k : $\{i_{k_1}, i_{k_2}, \ldots, i_{k_m}\} = \langle I_k \rangle$, $I_k \subset I$
sont des V_1 : $i_{l_1}, i_{l_2}, \ldots, i_{l_n} = \langle I_1 \rangle$, $I_1 \subset I$

forme un schéma de substitution pour les identificateurs $i_{k_1}, i_{k_2}, \ldots, i_{k_m}, i_{l_1}, \ldots, i_{l_n}$.

En général, ces règles sont liées entre elles :

1) chacun des i_k peut être une chaîne sur $V_T \cup V_N$, un élément de V_N, composante de i_k, peut désigner une chaîne dont une spécification est donnée ailleurs.

2) certains schémas peuvent être définis pour une partie d'une phrase seulement, la même chaîne peut alors apparaître dans les listes d'identificateurs de correspondances différentes.

2.1.1. - LES REGLES DE SUBSTITUTION DANS LE CAS D'ALGOL

Considérons par la suite le cas particulier d'ALgol :

1) les i_k sont des chaînes sur V_T
2) Chacun des i_k est local à un niveau de nomenclature. Soit $N_1, N_2, \ldots, N_k$ une chaîne ordonnée de k niveaux de nomenclature et $I_1, I_2, \ldots, I_k$ les ensembles des quantités locales à $N_1, N_2, \ldots, N_k$ respectivement ; N_k étant le niveau qui inclut tous les autres :

$$N_k \supset N_{k-1} \supset \ldots\ldots \supset N_1$$

On définit les quantités qui sont à la fois locales à N_k et accessibles dans N_1 par $[\delta]$:

$$= \begin{cases} I_k \cap \complement\left(\bigcup_{j=1}^{k-1} I_j\right) & k > 1 \\ I_1 & k = 1 \end{cases}$$

La totalité des quantités accessibles dans N_1 est la réunion des ensembles disjoints des quantités définies ci-dessus.

Un schéma de substitution est admissible, si et seulement si les deux conditions suivantes sont satisfaites :

C1 : Tout identificateur dont l'occurence se situe dans un certain niveau de nomenclature appartient à l'ensemble des quantités accessibles à ce niveau.

C2 : Chaque identificateur n'apparaît qu'une seule fois dans les listes d'un schéma de substitution du même niveau de nomenclature.

Lorsque l'ensemble des identificateurs accessibles à chaque niveau est connu et que les schémas de substitution sont admissibles, on peut effectuer la substitution dans les formules :

Règle de substitution R1

Soit I un identificateur d'une liste d'un schéma, appartenant aux quantités accessibles d'un niveau donné, V l'élément de V_N qui lui correspond par ce schéma

et X une occurence d'un identificateur dans une formule du même niveau ; soit $[V/I]$ X l'opération de substitution de V pour I dans X, alors le résultat de $[V/I]$ X est

$[V/I]$ X $\equiv$ V si X $\equiv$ I et

$[V/I]$ X $\equiv$ X si X $\not\equiv$ I.

R1 est appliqué à l'ensemble de formules d'un niveau de nomenclature pour tout I qui est un élément de l'ensemble des identificateurs accessibles à ce niveau. Le mot obtenu par l'application de la règle du remplacement (analyse syntaxique) aux chaînes déclaratives et ensuite par l'application de R1 contient encore des paramètres formels - (U et V dans l'exemple 2.1.1) - qui n'ont pas pu être spécifiés par R1. Lorsqu'on veut éviter une méthode interprétative on demande que tous les paramètres formels soient mentionnés dans la partie spécification des procédures. Dans ce cas R1 peut être appliqué indifféremment aux paramètres formels et aux autres identificateurs et les chaînes déclaratives peuvent être remplacées entièrement par le terme de la métalangue correspondant.

Lorsque la partie spécification des paramètres formels est vide (ou incomplète) on peut souvent déterminer cette spécification par une analyse du contexte des paramètres. Une telle analyse demande un ensemble de règles du type 1, c'est-à-dire une transformation de la grammaire.

2.2. - LA SYNTAXE DES LANGAGES "CONTEXT-FREE"

Ce paragraphe contient quelques résultats établis de la théorie de langages formels CF. La première section précise la définition de ces langages et de leurs grammaires et donne certaines propriétés remarquables et importantes pour la compilation. Les définitions sont celles données par Y.Bar-Hillel.

La deuxième section présente un algorithme d'analyse (et de compilation) valable pour une grande classe de langages CF. Il est basé sur un théorème de Bar-Hillel et il suit les lignes exposées par M. Paul. Cet algorithme permet par la suite de définir une classe de langages CF particuliers et de distinguer certaines ambiguïtés d'une grammaire donnée.

2.2.1. - DEFINITIONS ET PROPRIETES DE LANGAGES CF

L'ensemble des chaînes de longueur finie formées sur l'alphabet A (respectivement sur le vocabulaire V) sera désigné par Γ_A (respectivement Γ_V). Les éléments de Γ_A sont $\alpha, \beta, \ldots, \xi, \ldots$ et ceux de Γ_V sont a, b,..., x,.... Le vocabulaire est un ensemble fini de symboles A, B, ..., $V = V_T \cup V_N$, $V_T = A \cup \Lambda$.

On définit la concaténation (ou juxtaposition) de deux éléments de Γ_V x et y , $x = X_{i_1} \ldots X_{i_m}$ et $y = Y_{j_1} \ldots Y_{j_n}$ par $x \circ y = xy = X_{i_1} \ldots X_{i_m} Y_{j_1} \ldots Y_{j_n}$. La concaténation est associative = $x(y\ z) = (x\ y)\ z$.

Un élément x de Γ_V qui peut être écrit comme $x = x'\ y\ x''$ est dit composé de x', de y et de x" ; y est un segment (composante) de x. C'est un segment propre si $x \neq y$.

Lorsque $x' = \Lambda$ (Λ la chaîne vide), y est un segment initial de x, c'est un segment final si $x'' = \Lambda$.

On définit des relations de génération sur Γ_V par un ensemble fini de règles, les productions (voir § 1.1 pour la définition générale).

- Définition 2.2.1 (Bar-Hillel) :

Un SPS (en anglais : simple phrase structure system) est un couple ordonné (V,P) V étant le vocabulaire et P un ensemble fini de productions de la forme

$$X \longrightarrow x \qquad (x \neq X,\ x \in \Gamma_V,\ X \in V - V_T)$$

- Définition 2.2.2 (Bar-Hillel)

Une grammaire SPG (en anglais : simple phrase structure grammar) est un quadruplet ordonné $\mathcal{G} = (V, P, V_T, S)$, (V,P) étant un SPS, $V_T \subset V$ est le vocabulaire terminal et S le symbole distingué de $V - V_T$.

- Définition 2.2.3

Un langage SPL (en anglais : simple phrase structure langage) L_A ($L_A \subset \Gamma_A$) est défini par

a) une relation entre chaînes x et y, $x, y \in \Gamma_V$

$$\{(x \curvearrowright y) \mid \exists (z', z'') \wedge (U \to u) \in P,\ x,y,z',z'' \in \Gamma_V,\ x = z' U z'' \quad \text{et} \quad y = z' u z''$$

b) un ensemble $\Phi(S)$

$$\{\xi \in \Gamma_A \mid \exists (x_i) \ , \ 1 \leqslant i \leqslant n \ , \ (x_i \curvearrowright x_{i+1}),\ x_1 = S \text{ et } x_n = \xi\}$$

c) $L(\mathcal{G}) = L_A = \Phi(S) \cap \Gamma_A$

- Définition 2.2.4.

Le produit de deux langages L_1 et L_2 est le langage $L_1.L_2$ défini par les chaînes $L_1.L_2 = \{x.y \mid x \in L_1 \text{ et } y \in L_2\}$.

On remarque la stricte équivalence entre la définition d'un langage CF ou de type 2 (Chomsky § 1.1) et un langage SPL. La définition 2.2.3 choisit certaines chaî-

nes sur A comme les mots d'un langage $L(\mathcal{G})$ pour une grammaire $\mathcal{G}$ donnée, c'est-à-dire qu'un mot appartient au langage si, et seulement si il existe une dérivation $\Phi(S)$ dont il fait partie (définition 2.2.3.b). On n'est, par contre, pas assuré que tout $X \in V-V_T$ puisse être réellement employé dans une dérivation (connexité), ni qu'il existe au moins une chaîne $\xi \in \Gamma_A$ pour tout X, telle que $X = z_0, z_1, \ldots, z_n = \xi$ et $z_i \; r \; z_{i+1}$), c'est-à-dire $X \Rightarrow \xi$.

On est donc amené à restreindre la classe des langages (respectivement des grammaires) de la définition 2.2.3. de la façon suivante :

- Définition 2.2.5.

Une grammaire $\mathcal{G} = (V, P, V_T, S)$ est dite réduite si A) il existe au moins un mot ξ , élément de Γ_A pour tout $X \in V-V_T$, c'est-à-dire $X \Rightarrow \xi$

B) il existe au moins une paire de chaînes z', z" (éventuellement vides), telles que $S \Rightarrow z' \; X \; z"$ pout tout $X \in V$.

On donne d'abord un procédé de construction d'une grammaire $\mathcal{G}'$ satisfaisant au point 2.2.5.A) à partir d'une grammaire donnée $\mathcal{G}$.

Définissons la chaîne ascendante d'ensembles $V_1 \subseteq V_2 \subseteq \ldots$ par $V_1 = A$ c'est-à-dire que le premier ensemble comprend exactement les éléments de l'alphabet donné.

$$V_{k+1} = V_k \cup \{X \mid X \in V-V_T \text{ et } (\exists x)[(x \in \Gamma_{V_k}) \text{ et } (X \rightarrow x) \in P]\}$$

$$k = 1, 2, \ldots$$

c'est-à-dire que l'ensemble V_{k+1} diffère de l'ensemble V_k par les éléments de $V - V_T$ qui sont déterminés par les productions dont la partie droite est une chaîne sur V_k (mais qui n'est pas une chaîne sur V_{k-1}). ?

$V_{k+1} \supseteq V_k$ et $(\exists n)\ V_{n+1} \equiv V_n$, c'est-à-dire que l'algorithme de construction est terminé après un certain nombre de constructions et qu'il ne peut pas excéder le nombre d'éléments du vocabulaire.

Posons $\mathcal{G}'(V', P', V_T, S)$ où $V' = V_n$ et P' l'ensemble de productions employées dans la construction de V_n ; $\mathcal{G}'$ satisfait à la définition 2.2.5.A. On supposera par la suite toute grammaire dans sa forme $\mathcal{G}'$. Un procédé d'élimination des éléments syntaxiques qui ne satisfont pas à la définition 2.2.5.B) peut être trouvé de façon analogue (voir la construction de théorème 2.2.2)

La relation de la définition 2.1.3.a) entre chaînes de Γ_V permet d'ordonner celles-ci ; soit $\mathcal{R} = \bigcup_{n=1}^{\infty} r^n$. $\mathcal{R}$ est la fermeture transitive de r, elle est vérifiée entre deux chaînes x et y ($x \mathcal{R} y$) lorsqu'il existe une suite de transformations d'un mot x, telle que :

$x = z_0, z_1, \ldots, z_n = y$ et $z_i \; r \; z_{i+1}$ (déf. 2.2.3.b) .

La relation $\mathcal{R} \cup I$ était introduite au § 1.1, elle se note " $\Rightarrow$ " ($x \Rightarrow y$, y est un mot pour x), c'est une relation d'ordre partiel entre chaînes.

La suite des mots z_o, z_1,..., z_n est une séquence de génération de y. Dans le cas où $y \in \Gamma_A$, la séquence est terminée. Une séquence terminée s'appelle dérivation. Etant donné $x = x_o$ et $y = y_o$, $x_o \mathcal{R} y_o$, la séquence de génération correspondante n'est pas déterminée de façon unique en général.

Soit $x = x_o$ un mot donné $x \in \Gamma_V$, c'est-à-dire tel que sa séquence de génération ne soit pas terminée et soit $\{U_i \rightarrow u_i\}$, i = 1, 2, ...,k l'ensemble de productions applicables à x.

On peut distinguer un mode de génération de $y = y_o$ à partir de $x = x_o$ faisant correspondre à x_o et y_o une séquence de génération particulière, dite dérivation séquentielle (ou dérivation ordonnée). Dans une telle dérivation le symbole réécrit dans chaque mot de la suite z_i peut être connu sans comparaison avec le mot suivant z_{i+1} et par conséquent la dérivation est déterminée de façon univoque par une suite de règles de la grammaire.

- Définition 2.2.6.

Soit $x = z' U z''$ un mot de $L(\mathcal{G})$, $U \in V - V_T$ $U \in \{U_i\}$, z', $z'' \in \Gamma_V$ et soit z" un segment final (propre ou non) d'une dérivation (c'est-à-dire un segment terminal)

a) Dans ce cas, U est le premier symbole transformable (PST) de x par l'application de la production $(U \rightarrow u) \in (U_i \rightarrow u_i)$

b) Soit $x \mathcal{R} x_1$, $x_1 \tau y$. Si y peut être obtenu par une application de la production $U \rightarrow u$ à partir d'un x_1, tel que $x \mathcal{R} x_1$, alors U est le premier symbole non terminé (PSN) de x.

3) Expressionsde désignation

Les expressions de désignation sont toujours des segments finaux propres d'une instruction go to ou d'un élément de liste de switch . Les expressions non-simples ne peuvent être situées dans une liste de paramètres effectifs

L est une étiquette (identificateur en position d'une expression de désignation simple).

S → SI[
EDS → SE] | L | (ED)
ED1 → PIF1 EB THEN
ED → EDS | ED1 EDS ELSE | ED ED

σ_I est la catégorie d'un identificateur en position d'indice dans un indicateur de fonction. Cet état n'est maintenu que si c'est une variable non-locale et si l'indicateur de fonction est spécifié.

4) Instructions simples

GTS → go to ED

instruction go to (IGT) . L'étiquette correspondante se trouve en α [a] .

AST → σ := partie gauche d'une instruction d'affectation. La variable se trouve en α [a] , fonction d'adresse évaluée. Une liste de partie gauche compte autant de AST qu'elle compte de parties gauche. L'instruction est un mot de la forme

$\{AST\}^{n} AST\ E$

La conséquence suivante est évidente : soit $x = z' U z''$ et soit U le P S N de x. Le P S T est U ou il est segment de z'.

- Définition 2.2.7.

Une dérivation séquentielle est une séquence de génération, telle que tout $x = x_{i+1}$ est une conséquence d'un x_i par rapport à une production qui transforme le P S T de x_i.

- Théorème 2.2.1.

Dans toute grammaire du type S P G il existe une dérivation séquentielle pour tout $x \in L(\mathcal{G})$ [5].
Soit n le nombre de transformations dans une dérivation de $x \in L(\mathcal{G})$ et supposons toutes les dérivations possibles de x non-séquentielles.

On peut choisir, parmi toutes les dérivations de longueur n, les séquences de construction $x = x_1, x_2, \ldots, x_i, \ldots$ telles que x_i, $1 \leqslant i < n$ est le premier mot de la séquence dans lequel le P S T n'est pas le symbole transformé. Soit i maximum parmi toutes ces séquences ainsi choisies. Si les mots ont la forme $x_j = z'_j U_j z''_j$, $1 \leqslant j \leqslant n$, $U_j \in V-V_T$ pour tout j $1 \leqslant j < i$, U_j est le symbole transformé de x_j.

Comme $x_n = y$ est le dernier élément d'une dérivation, il existe un k, $i < k < n$, tel que U_i est le symbole transformé de x_k. Par définition, $x_i = z'_i U_i z''_i$ et $x_k = z'_k U_i z''_i$, mais $x_i \mathcal{R} x_k$, donc $(z'_i U_i z''_i) \mathcal{R} (z'_k U_i z''_i)$ d'où $z'_i \mathcal{R} z'_k$.

Par l'application de $U_i \rightarrow u_i$ à z_k on obtient $z_{k+1} = z'_k u_i z''_k$ et si l'on remplace x_{i+1} par x_{k+1}, les autres transformations restant inchangées, x_{k+1} reste inchangée et la dérivation est séquentielle jusqu'au terme x_{i+1}. Donc i n'est pas la longueur maximale d'une séquence de construction séquentielle.

On considère donc par la suite des mots faisant partie d'une séquence de construction séquentielle sauf exception mentionnée.

Un mot y pour un $X \in V_N$ est déterminé de façon unique par X et par une séquence de construction, c'est-à-dire par un n-tuplet ordonné de productions. D'autre part, si deux séquences de construction différentes (mais toutes deux séquentielles) et partant du même $X \in V_N$ donnent un mot identique, le langage (respectivement la grammaire du langage) est dit ambigu. On considère ici en principe des langages sans ambiguïtés structurelles, c'est-à-dire des langages dans lesquels tout mot possède essentiellement une seule dérivation.

2.2.2. - ANALYSE DES LANGAGES CF.

A toute dérivation dans une grammaire S P G d'un langage S P L (équivalent : CF) correspond une formule bien formée de ce langage. Dans le paragraphe précédent il a été montré que toute dérivation (et plus généralement toute séquence de génération) peut être donnée dans une forme particulière que l'on a appelée dérivation séquentielle. Celle-ci est parfaitement définie par une suite ordonnée de règles de la grammaire dont chacune indique la réécriture d'un symbole non-terminal d'une des chaînes de la dérivation. Une telle suite de productions définit une formule bien formée si et seulement si le P S T de chaque mot de la dérivation est identique au symbole transformé par la prochaine production.

Dans ce paragraphe on se pose le problème suivant : trouver un algorithme qui décide si une chaîne donnée (terminée ou non) appartient au langage et qui donne la suite ordonnée des règles appliquées successivement dans la dérivation séquentielle de ce mot si et seulement si la solution du problème est unique, c'est-à-dire si il n'y a pas plusieurs dérivations pour la même chaîne donnée.

Y. Bar-Hillel a démontré que, pour toute grammaire réduite du type S P G, c'est-à-dire satisfaisant à un certain nombre de propriétés, il existe un algorithme qui répond à la question $x \in L(\mathcal{G})$. La deuxième partie du problème par contre, qui est l'unicité de la solution trouvée (aussi appelée problème d'ambiguïté des grammaires) s'avère être une propriété indécidable d'une grammaire SPG.

Soit $\mathcal{G} = (V, P, V_T, S)$ une grammaire réduite et telle, qu'elle ne contient aucune règle de la forme $X \rightarrow \Lambda$, c'est-à-dire l'application d'une règle de réécriture à un symbole non-terminal d'une chaîne ne peut pas diminuer la longueur de cette chaîne. Dans ce cas, on a le théorème suivant :

- Théorème 2.2.1.

Soit $A \in V - V_T$ et $u, v, x \in \Gamma_V$; pour tout x $(\exists u)(\exists v)(A \mathcal{R} u x v)$ est effectivement décidable.

On construit pour le démontrer les ensembles suivants de chaînes :

Soit $l(z) = m = l(x)$ la longueur de la chaîne x, alors

$$E_m^1(A) = \{ z \mid l(z) \leqslant m \text{ et } (\exists u)(\exists v)(A \vdash uzv) \}$$

$$E_m^k(A) = \{ z \mid l(z) \leqslant m \text{ et } (z \in E_m^{k-1}(A) \vee (\exists u)(\exists v)(\exists y)(y \in E_m^{k-1}(A) \text{ et } y \vdash uzv) \}$$

Ces ensembles ont les propriétés suivantes :

P1 : $E_m^{k-1}(A) \subseteq E_m^k$

P2 : $E_m^{k-1}(A) \equiv E_m^k(A)$ implique $E_m^k \equiv E_m^{k+1}$

P3 : le nombre total de chaînes non-vides sur $V(V_1, \ldots, V_n)$ de longueur inférieure ou égale à m est

$$n + n^2 + \ldots + n^m = \frac{n^{m+1} - 1}{n - 1} - 1 = l ,$$

$(\exists u)(\exists v)(A \mathcal{R} u x v)$ est donc équivalent à

$\{(A = x) \vee (\Lambda = x) \vee (\exists k)(x \in E_m^k(A))\}$ et encore à

$\{(A = x) \vee (\Lambda = x) \vee (x \in E_m^1(A))\}$

Corollaire :

Pour tout x $(\exists u)\ (\exists v)\ (S\ \mathcal{R}\ u\ x\ v)$

$(\exists u)\ (A\ \mathcal{R} u x\ )$

$(\exists v)\ (A \mathcal{R} x v)$

est effectivement décidable.

Cet algorithme génère donc toutes les chaînes de longueur inférieure ou égale à une certaine longueur donnée et cherche le mot donné parmi ces chaînes. Il assure la validité du théorème 2.2.1. pour une classe de langages S P L (sans qu'il soit toutefois utilisable à un emploi pratique).

Les ensembles $E_m^1\ (X)$ peuvent servir à la définition d'un nombre de relations d'ordre entre chaînes d'un langage respectivement entre éléments du vocabulaire. Ces relations seront utilisées par la suite dans l'algorithme d'analyse.

- Définition 2.2.8.

La relation entre X et Y, notée $X \geqslant Y$ est définie par $\{(X \geqslant Y) \mid X, Y \in V \quad (X = Y \vee Y \in E_1^1\ (X)\}$
c'est-à-dire Y est segment d'un mot pour X.

C'est une relation de préordre : elle est réflexive et transitive, car $Y \in E_1^1\ (Y)$ et $Y \in E_1^1\ (X)$ et $Z \in E_1^1 (Y)$ impliquent $Z \in E_1^1\ (X)$.

De même on définit :

$X \geqslant_1 Y$ par $\{(X \geqslant_1 Y) \mid X, Y \in V \quad [X = Y \vee (\exists u)\ (X \mathcal{R}_u Y)$

$X \geqslant_2 Y$ par $\{(X \geqslant_2 Y) \mid X, Y \in V \quad [X = Y \vee (\exists v)\ (X \mathcal{R} Yv)$

avec $u, v \in \Gamma_V$.

On désigne maintenant par $C_1(Y)$ l'ensemble des X_i, tel que $X_i \geqslant_1 Y$ pour tout i , $i = 1,2,\ldots, n$, par $C_2(Y)$ l'ensemble des X_i, tel que $X_i \geqslant_2 Y$ pour tout i, $i = 1,2,\ldots, n$ et par $C(Y)$ l'ensemble des X_i, tel que $X_i \Rightarrow Y$.

- Définition 2.2.9.

L'ensemble d'éléments de $V - V_T$ $V(X, Y)$ est défini par $\{Z \mid (Z \rightarrow AB) \in P,\ A \in C_1(X) \text{ et } B \in C_2(Y)\}$.
$V(X,Y) \neq \emptyset$ est une conditions pour l'existence d'un mot du langage de la forme u X Y v, c'est-à-dire pour $X\,Y \in E_2^1(S)$.

Soit H et A des éléments fixes de $V - V_T$ et $\Rightarrow$ la relation d'ordre partiel du paragraphe 2.2.1.

- Définition 2.2.10.

Les chaînes x_i respectivement y_i pour lesquelles

$$H \Rightarrow x_i A \qquad i = 1, 2, \ldots$$

$$H \Rightarrow A\, y_j \qquad j = 1, 2, \ldots$$

s'appellent segments complémentaires à gauche respectivement à droite de A par rapport à H.

Dans l'ensemble de segments complémentaires de A par rapport à H on peut former des classes $x_i(y_j)$:

$$x_{i_1} \uparrow x_{i_2} \uparrow \ldots\ldots \uparrow x_{i_{n_i}} \qquad i = 1, 2, \ldots, m$$

$$y_{j_1} \tau\ y_{j_2} \tau\ \ldots\ldots \tau\ y_{j_{n_j}} \qquad j = 1, 2, \ldots\ldots, n$$

où les $x_{i_{n_i}}$ et les $y_{j_{n_j}}$ sont des chaînes sur V_T et les x_{i_1} (et les y_{j_1}) des chaînes dérivables d'aucune autre chaîne sur V par les règles de réécriture de la grammaire.

Soit maintenant $S \Rightarrow xAy$, alors on associe un automate à pile [5,11] à cette forme du mot : x correspond au contenu de la pile, y au contenu de la bande d'entrées et A à un état de l'automate.

Supposons que xAy fasse partie d'une dérivation séquentielle, alors y est de forme d'un $y_{j_{n_j}}$ (chaîne sur V_T) et x est de la forme d'un x_{i_1} ; A correspond au PST du mot xAy.

A l'aide des relations définies ci-dessus, on caractérise maintenant une classe de grammaires respectivement de langages par un certain nombre de conditions. On obtient ainsi un propre sous-ensemble des langages CF que l'on peut appeler "séquentiellement analysable". Lorsque l'une ou l'autre de ces conditions n'est pas satisfaite on trouve des langages ambigus ou des classes de langages, ne pouvant pas être analysés "séquentiellement" (et qui sont éventuellement en même temps ambigus).

On suppose par la suite que les grammaires en question possèdent uniquement des productions de la forme $X \rightarrow x$

où x est une chaîne de longueur 1 ou 2 ; ceci n'est pas une nouvelle restriction.

- Définition 2.2.11.

Soit $S \Rightarrow xXYy$, $x \in \Gamma_V$, $y \in \Gamma_A$.

Les conditions suivantes définissent un langage séquentiellement analysable et non-ambigu :

a) Lorsque $x \equiv \Lambda$ (XY est un segment initial propre d'un mot pour S) :
Quel que soit $B \in C_2(Y)$, il existe un seul segment complémentaire à gauche A (de longueur 1), $A \in C(X)$, tel que $H \rightarrow AB$ $(H_2 \leqslant S)$ est une règle de la grammaire.

b) Lorsque $y \equiv \Lambda$ (XYest un segment final d'un mot pour S) :
Soit xX dans la forme d'un segment complémentaire à gauche, alors quel que soit $B \in C(Y)$, il existe un seul $G_1 \leqslant S$ (c'est-à-dire $S \Rightarrow xG$), tel que $G \rightarrow XB$ est une règle.

c) Soit $x \not\equiv \Lambda$, $y \not\equiv \Lambda$ et $S \Rightarrow xXYZy$; $A \in C(Y)$ et $B \in C_2(Z)$ l'une des deux alternatives doit être satisfaite :
1) Quel que soit H, $H \rightarrow AB$: $V(X, H) = \emptyset$ mais $G \rightarrow XA$ est unique et $V(G, Z) \neq \emptyset$; ou bien
2) Quel que soit G , $G \rightarrow XA$: $V(G,Z) = \emptyset$ mais $H \rightarrow AB$, A est unique et $V(X, H) \neq \emptyset$.

On construit un automate à pile à l'aide de la définition 2.2.11. qui accepte(traduit) une chaîne d'un langage CF.

Un tel automate consiste en trois éléments de base :
1) une tête de lecture
2) une tête d'écriture
3) une unité de contrôle.

La tête de lecture peut lire sur un premier ruban ou ruban d'entrée, sur lequel la chaîne donnée est écrite de gauche à droite dans l'alphabet V_T. Ce ruban est lu de gauche à droite et on ne peut lire qu'un symbole à la fois. On ne peut pas revenir sur un symbole qui a été déjà lu.

La tête de lecture peut aussi lire sur un deuxième ruban (la pile). Ce ruban est vide avant et après un calcul de l'automate. Pendant le calcul une section est remplie par une chaîne de symboles sur V. On convient de dire que c'est le symbole le plus à droite qui est exposé à la tête de lecture qui fait avancer le ruban à droite en exposant ainsi un nouveau symbole après avoir lu le symbole exposé auparavant.

Le premier casier vide, à droite de la tête de lecture, est exposé à une tête d'écriture qui fait reculer le ruban à gauche après avoir marqué un symbole dans le casier qui lui a été exposé. Sur ce ruban on ne peut lire à tout moment que le dernier symbole écrit (en principe du "last in - first out").

L'unité de contrôle réalise un nombre (fini) de décisions qui déterminent l'action de l'automate à l'aide de l'information reçue par la tête de lecture et par un nombre fini de règles que l'on suppose enregistrées dans l'unité de contrôle.

Une situation de l'automate est définie par un triplet $(S_k, E_l, E[i])$ où $S_k \in V_T$, $E_l \in V$ "l'état" de l'automate et $E[i] \in V$ le symbole accessible de la pile, i étant le nombre d'éléments contenus dans la pile à l'instant donné. Les actions de l'automate sont

1) Les mouvements des rubans
2) L'écriture sur le deuxième ruban (la "pile")
3) Le changement de l'état E_l.

Les règles de calcul (ou règles de transition) sont de la forme

$$(S_k, E_l, E[i]) \xrightarrow{T} (\alpha, E_m, E[\beta])$$

où α indique le mouvement du ruban d'entrée,
β celui de la pile.

1) Une règle $(S_k, E_l, E[i]) \xrightarrow{T} (S_k, G, E[i-1])$ correspond à chaque situation c) 1) de la définition 2.2.11.
2) Une règle $(S_k, E_l, E[i]) \rightarrow (S_k, A, E[i])$ correspond à chaque situation c) 2) de la définition 2.2.11.
3) Lorsque $E_l \equiv A$, E_l est écrit dans la pile et la bande d'entrée avance :

$$(S_k, E_l, E[i]) \xrightarrow{T} (S_{k+1}, S_k, E[i+1]).$$

Un calcul de l'automate se termine par l'acceptation de la chaîne donnée lorsque la situation finale est $(\#, S, \#)$ où $\#$ est un signal de fin de bande et S le symbole distingué de la grammaire.

On examine maintenant un cas où la définition 2.2.11. n'est pas satisfaite :

. Exemple 2.2.1

Exemple d'une grammaire ambigue :

$$\mathcal{G} = (V, P, V_T, S)$$

$V_T = \{\underline{PIF}, \underline{PFOR}, \underline{IBA}, \underline{ELSE}\}$

$V = \{V_T ; IFOR, IC, ICO, IIC, IIF, I\}$

$S = I$

P = 1 $\langle I \rangle = \langle IIC \rangle \mid \langle ICO \rangle$ Instruction

2 $\langle ICO \rangle = \langle IIF \rangle \langle IC \rangle \langle I \rangle$ instruction conditionnelle

3 $\langle IC \rangle = \langle IIF \rangle$ ELSE instruction conditionnelle suivie par else

4 $\langle IIF \rangle = \underline{PIF} \langle IIC \rangle$ instruction IF

5 $\langle IIC \rangle = \langle IFOR \rangle \mid \underline{IBA}$ instruction inconditionnelle

6 $\langle IFOR \rangle = \underline{PFOR} \langle I \rangle$ instruction FOR

7 IBA : instruction de base

8 PFOR : proposition FOR

9 PIF : proposition IF

Les matrices de relations sont : $C_1(X)$, $C_2(X)$, $C(X)$

C_1 :

	1	2	3	4	5	6	7	8	9	10
1	v	v		v	v	v	v			
2	v	v		v	v	v	v			
3			v						v	
4	v	v		v	v	v	v			
5	v	v		v	v	v	v			
6	v	v		v	v	v	v			
7							v			
8								v		
9									v	
10										v

C :

	1	2	3	4	5	6	7	8	9	10
1	v	v		v	v	v	v			
2		v		v						
3			v							
4				v						
5					v	v	v			
6						v				
7							v			
8								v		
9									v	
10										v

C_2:	1	2	3	4	5	6	7	8	9	10
1	v	v	v	v	v	v	v			
2		v	v	v						
3			v							
4				v					v	
5					v	v	v			
6						v		v		
7							v			
8								v		
9									v	
10										v

La chaîne donnée est

PIF PFOR PIF IBA else IBA , elle peut être réécrite comme

PIF PFOR ⟨IIF⟩ else IBA

Le triplet $\{X = \underline{PFOR}\ ,\ Y = \langle IIF\rangle\ \ Z = \underline{else}\}$ crée une situation ambigüe ;

1) $C(Y) = \{\langle I\rangle\ ,\langle ICO\rangle\ ,\langle IIF\rangle\}$, d'où $G = \{\langle IFOR\rangle\}$

et avec

$C_1(\langle IFOR\rangle) = \{\langle I\rangle\ ,\ \langle ICO\rangle\ ,\ \langle IIF\rangle\ ,\ \langle IIC\rangle\ ,\ \langle IFOR\rangle\}$ et

$C_2\ (\underline{else}) = \{\underline{else}\}$

$V\ (G,\ Z) = V(\langle IFOR\rangle\ ,\ \underline{else}) = \{\langle IC\rangle\} \neq \emptyset$

2) $C(Y) = \{\langle I\rangle\ ,\ \langle ICO\rangle\ ,\langle IIF\rangle\}$ d'où $H = \{\langle IC\rangle\}$

et avec $C_1\ (\underline{PFOR}) = \{\underline{PFOR}\}$ et $C_2(\langle IC\rangle) = \{\langle I\rangle\ ,\ \langle ICO\rangle\}$

$V(X,\ H) = V(\underline{PFOR}\ ,\ \langle IC\rangle) = \{\langle IFOR\rangle\} \neq \emptyset$

Le calcul se termine dans les deux cas par l'acceptation de la chaîne donnée.

En général, chaque fois que la définition 2.2.11. n'est pas satisfaite, il faut continuer plusieurs analyses.

Lorsque plus d'une entre elles conduisent à l'acceptation de la chaîne donnée, il y a ambiguité, c'est-à-dire qu'il existe plus d'une dérivation pour ce mot dans la grammaire donnée.

III - COMPILATION D'ALGOL 60

3.1. - ANALYSE SYNTAXIQUE D'UN PROGRAMME DONNE

Le compilateur a la structure d'un automate à pile (automate croissant). Les actions du compilateur sont déterminées par l'état interne dans lequel il se trouve, par les entrées ou données (c'est-à-dire par le programme écrit en Algol 60), qu'il reçoit et par une suite d'états internes enregistrés dans une pile (ou push-down - store, P D S) au cours de son histoire. Ces actions sont donc liées aux configurations internes qui se traduisent par une liste d'un nombre de triplets $\{S, E, E[i]\}$ où $E[i]$ est l'état au sommet de la pile, E l'état interne actuel et S un élément du vocabulaire d'entrée. Une action du compilateur est la mise en route de l'ensemble des programmes liés à chaque triplet $\{S, E, E[i]\}$.

3.1.1. - LES ELEMENTS D'ENTREE

Le programme donné est perforé sur un ruban qui est présenté à l'organe d'entrée (lecteur). Ce ruban ne peut être lu que de gauche à droite.

Les différents éléments du vocabulaire V_T qui se trouvent sur le rubant d'entrée consistent en général en un nombre différent de codes machine. Tous ces codes se

suivent sur la bande ; les blancs ne sont pas admis à l'intérieur d'un mot. La lecture est dirigée par le programme de lecture. Celui-ci a plusieurs fonctions : d'abord il effectue la dé-concaténation des symboles de V_T; ensuite il effectue une normalisation et une re-codification dans cet ordre pour chacun des symboles de la chaîne donnée. Ce programme s'arrête après la lecture d'un seul symbole. Il transmet le code qui corrèspond au symbole lu dans une mémoire M qui le conserve jusqu'au prochain appel du programme de lecture.

Tous les mots pour ⟨identificateur⟩ ou ⟨nombre⟩sont remplacés par une codification de l'élément de V_N correspondant, mais les mots d'origine sont normalisés et conservés dans certaines mémoires.

Les fonctions du programme de lecture sont donc d'une nature préparatoire et servent à constituer un "ruban idéalisé" qui peut-être obtenu si l'on place les contenus successifs de M les uns après les autres chacun étant séparé du suivant par un trait de séparation.

Il convient de noter que le symbole représenté par le contenu de M est, à un instant donné, la seule donnée accessible. C'est exactement celle, qui se trouve sous la "tête de lecture" du "ruban idéalisé".

3.1.1.1. - LA REPRESENTATION CODIFIEE DES SYMBOLES DE BASE

Le code tient dans un mot machine. Il consiste en deux parties : la première, cadrée à droite, est une adresse qui indique une section de la table de syntaxe. La deuxième partie contient une spécification lorsque plusieurs symboles de base sont groupés dans une même catégorie de base (ce qui se traduit par une partie adresse identique pour chacun des membres de cette catégorie). Si une catégorie ne comporte qu'un seul symbole, la partie spécification de la codification de ce symbole est vide.

Les catégories ayant plus d'un membre sont :

- les opérateurs arithmétiques, booléens et de relation.
- les déclarateurs de type real, integer, Booleen
- les spécificateurs label, value, string

Exemple :

En-tête d'une déclaration de procédure :

1) chaîne d'entrées d'origine (formé sur V_T)
2) "ruban idéalisé" (formé par les contenus successifs de M)

1) 'DEBUT' 'REEL' 'PROCEDURE' 'ROMBERG (FCT, A, B, ORD) Δ

'REEL' A, B Δ 'ENTIER' ORD Δ 'REEL' 'PROCEDURE' FCT Δ

2)

5100	16782100	1053877	4 895	5249	4895	
5214	4895	5214	4895	5214	4895	
5269	4982	16782100	4895	5214	4895	
4982	8393492	4895	4982	16782100	1053877	4895

3.1.2. - REGLES DE CALCUL DU COMPILATEUR - TABLE DE SYNTAXE

Chaque pas d'un calcul syntaxique (dérivation) est en général déterminé par un triplet $\{S_k, E_1, E[i]\}$ la configuration interne du compilateur et par une des règles de transition

$$\{S_k, E_1, E[i]\} \xrightarrow{T} \{\alpha, E_m, E[\beta]\}$$

où $S_k \in S$, $S \subset V$ et $V_T \subset S$,

E_1 est l'état interne actuel, $E[i]$ l'état au sommet de la pile (i est l'indice variable qui indique le sommet).

α peut prendre les deux valeurs : S_k ou S_{k+1}

E_m est le nouvel état

β peut prendre les trois valeurs i -1, i, i+1
lorsque $\beta = i+1$, la règle transition a la forme :

$$\{S_k, E_1, E[i]\} \xrightarrow{T} \{S_{k+1}, E_m, E_1 (= E[i+1])\}$$

Chacune des configurations internes possibles ne peut apparaître qu'une seule fois comme membre gauche dans une des règles de transition. Les états internes E_i sont étroitement liés à la syntaxe d'Algol. Chaque état interne correspondant à une certaine catégorie syntaxique. Ces catégories sont définies en section 3.1.5 -
L'ensemble des règles de transition peut être dérivé de l'ensemble de productions. Ces règles sont codifiées en forme d'une table, divisée en chapitres. Chaque chapitre correspondant à une certaine classe d'éléments du vocabulaire d'entrée S.

En tête de chaque chapitre on trouve l'ensemble des états internes présents dans le chapitre - ceci désigne explicitement les paires $\{S_k, E_j\}$ admissibles.

Si, pour un chapitre C_k correspondant à un S_k lu, l'état E_j actuel ne se trouve pas dans l'en-tête du chapitre, la chaîne d'entrée n'est pas acceptée par le compilateur et la traduction s'arrête sur "erreur de syntaxe". La configuration interne ne correspond dans ce cas à aucune des règles syntaxiques.

Chaque ligne d'un chapitre consiste en deux sortes de mémoires. La première donne l'ensemble des états pour lesquels un certain programme est à exécuter, la deuxième donne l'adresse de ce programme. Un tel programme comporte :

A) les opérations de transition
B) les opérations sur les listes (opérations de substitution)
C) les programmes de sortie - c'est-à-dire de génération -

Remarque : B) et C) peuvent être vides.

3.1.3. - REGLES DE TRANSITION ET PRODUCTIONS

Tout triplet $\{S_k, E_l, E[i]\}$ détermine le prochain état interne du compilateur, les opérations avec la bande d'entrées (décalage à gauche ou maintien) et les opérations sur la pile. Une règle de transition est liée à une production. Elle établit la correspondance entre les mots d'une chaîne donnée et les éléments de V_N qu'ils

représentent en effectuant le remplacement chaque fois que c'est possible. Il y a quatre règles type dont trois seulement sont essentielles, la quatrième pouvant être composée à partir de deux autres règles.

1) - Le changement d'état interne CH(X), qui a l'effet

$$\{S_k, E_1, E[i]\} \xrightarrow{T} \{S_{k+1}, X, E[i]\}$$

c'est-à-dire l'état actuel est changé en X. Le prochain élément de S est amené sous la tête de lecture mais $E[i]$, le contenu de la pile reste inchangé.

Cette règle correspond à une production de la forme $X \rightarrow E_1 S_k \qquad X \neq E_1$

ou plus généralement à une production de forme

$A \rightarrow E_1 S_k Z$ où $X, E_1 \in \{E\}$, $S_k \in S$

et Z est une chaîne sur S.

Une telle règle est généralement décomposée en

$A \rightarrow X Z$, $X \rightarrow E_1 S_k$,

Ce procédé pouvant être itéré.

Exemple :

$(BI, \omega) \rightarrow \sigma\omega$ où ω est un opérateur binaire, σ un opérande et (BI, ω) l'état d'opération binaire ω . La transition CH(X) remplace une chaîne $E_1 S_k$ par $X \in \{E\}$ lorsqu'elle est constituante d'une partie droite d'une des productions.

2) L'entrée d'un état dans la pile : ENT(X)

$$\{S_k, E_1, E\ [i]\} \xrightarrow{T} \{S_{k+1}, X, E_1 = E\ [i+1]\}$$

Ce type de transition ajoute l'état actuel à la pile, choisit un nouvel état et amène le prochain élément de S. Il intervient chaque fois que $E\ [i]\ E_1 S_k$ ou $E_1 S_k$ ne peut pas être remplacé par un $X \in \{E\}$.
C'est le cas dans les productions telles que

$A \rightarrow BC$, $C = S_k Z$ (et $B = E_1$)

et α) $C \Rightarrow S_k$ S_k est un mot pour C
β) $C \not\Rightarrow S_k$ S_k n'est pas un mot pour C
} mais $C \not\rightarrow S_k$

• <u>Exemples</u>

α) soit B = (BI, ω = +) et S_k = VId (variable id.), alors C = term et $C \Rightarrow S_k$, mais $C \not\rightarrow S_k$ et il existe Z, tel que $C \Rightarrow S_k Z \rightarrow S_k$.

β) Soit B et C comme avant mais S_k = "(" , alors il existe Z, tel que $C \Rightarrow S_k Z$, mais $C \not\Rightarrow S_k$.

3) La répétition avec le même S_k qui fait intervenir la pile : $\{S_k, E_1, E\ [i]\} \xrightarrow{T} \{S_k, E\ [i]\ , E\ [i-1]\}$

La bande de lecture est maintenue dans sa position, l'état au sommet de la pile prend le contrôle et la pile est "décalée" à gauche.
Cette transition est appliquée dans un cas tel que

$A \rightarrow BC$, $B = E\ [i]$
et α) $C \Rightarrow E_1$, mais $E_1 S_k Z \overset{Y}{\Rightarrow} C$
ou β) $C \Rightarrow E_1$, mais $E_1 S_k Z \overset{Y}{\not\Rightarrow} C$
} $C \not\Rightarrow E_1 S_k Z$

Exemples :

α) soit B = (BI, ω = X) , C = facteur
$S_k = \oplus$ (opérateur d'addition) , E_1 = V (variable)
alors avec Z = V,
facteur $\Rightarrow$ V variable est un mot pour facteur mais
V $\oplus$ V = S A E (simple arithmetic expression) $\Rightarrow$ facteur

β) soit B = E [i] , et C et E_1 comme avant, et S_k = ;
alors C $\Rightarrow E_1$ mais pour aucun Z C $\Rightarrow E_1\ S_k\ Z$.

4) - LA sortie S $\left\{S_k, E_1, E\,[i]\right\} \xrightarrow{T} \left\{S_{k+1}, E\,[i], E\,[i-1]\right\}$
peut être composée par R et CH

3.1.4 - LES LISTES ET LES OPERATIONS DE LISTES

Le compilateur construit plusieurs listes de structure différente. Toutes les listes servent au transfert de l'information d'un point du programme à un autre.

Distinguons les listes en forme de pile ou P.D.S. (push-down-store) qui fonctionnent strictement suivant le principe du "last-in-first-out". Elles effectuent un rangement des éléments syntaxiques déduits des chaînes du texte de la langue source suivant un ordre d'inclusion.

Les listes croissantes (et décroissantes) des variables suivent la structure des blocs. Elles définissent

les ensembles de variables accessibles, c'est-à-dire les variables à substituer dans les formules d'un bloc. Ces listes sont donc constituées par des suites d'éléments équivalents entre eux dans un certain sens.

Les listes strictement croissantes, sans structure particulière (c'est-à-dire dont la représentation dépend uniquement de la machine), constituent la sortie du compilateur ("programme généré" et tables annexes).

3.1.4.1. - L E S P I L E S

Deux piles sont formées pendant la compilation : celle des états syntaxiques E [i] et celle des opérandes α[a] i et a désignant la "hauteur" des piles, c'est-à-dire pratiquement leur dernier élément.

E [i] règle le processus d'analyse syntaxique. A tout moment E [i] est le seul élément de la pile qui intervient dans cette analyse.

La pile des opérandes contient les arguments des opérateurs et elle reçoit aussi leur valeur. Cette pile

peut donc contenir les codifications pour le type et l'adresse d'une expression quelconque.

Il est possible de confondre ces deux piles ; c'est le cas dans la notation post-fixée. Dans le cas de deux piles distinctes, il y a une correspondance bi-univoque entre éléments de E [i] et de α [a] .

3.1.4.2. - LES LISTES CROISSANTES

La liste des déclarations ν [d] contient la codification des quantités déclarées dans l'en-tête d'un bloc ainsi que l'identificateur qui s'y attache dans une suite de groupes de plusieurs mémoires. Une autre liste ε [e] contient les noms et adresses des étiquettes.

Ces listes sont divisées en niveaux de nomenclature. Elles définissent les ensembles d'éléments accessibles et d'éléments non-accessibles pour les opérations de substitution.

3.1.4.3. - LES LISTES AUXILIAIRES

Le compilateur doit avoir accès à tout élément d'une telle liste à tout moment. Ces listes sont :
- celle des constantes et valeurs booléennes
- celle des instructions générées en langage machine
- celle des tables d'adresses auxiliaires : aiguillages, instructions après FOR, etc...

NOM DE LA LISTE	CONTENU STATIQUE	DYNAMIQUE
E pile d'états syntaxiques	Etats syntaxiques	-
α pile d'opérandes	variables	résultats anonymes
$\mathcal{L}$ liste de déclarations	identificateur et codification (type +adr.)	valeurs des variables
ε liste d'étiquettes	identificateur et adresses correspondantes	-
C liste des constantes	valeurs et codification (type+adr)	constantes
γ programme généré	instructions machine	

3.1.5. - LES ETATS SYNTAXIQUES

Cette section établit la correspondance entre les états internes du compilateur $\{E\}$ et certaines classes de métavariables, c'est-à-dire d'éléments de V_N.

1) Variables et indicateurs de fonction

Tout identificateur (ou nombre ou valeur booléenne) correspond à l'état σ (opérande) et à un élément de la pile des opérandes $\alpha[a]$. Le couple $(\alpha[a] , \sigma)$ définit, par une codification appropriée de $\alpha[a]$, les identificateurs suivants :

VI , AI , SI , PI

identificateur de variable, de tableau, d'aiguillage et de procédure respectivement, ainsi que les métavariables

SV (= VI) , IV , FD , SD

variable simple, indicée, indicateur de fonction, d'aiguillage.

Les productions sont :

$(\sigma, \alpha[a]) \equiv$ IV $\rightarrow$ $\{EA,\}^{n}$ EA] $\quad \{EA,\}^{o} = \Lambda$

$\sigma_1 \rightarrow$ AI [

$(\sigma, \alpha[a]) \equiv$ FD $\rightarrow$ σ_2 $\{L2PD\}^{n}$ L2)

$\sigma_2 \rightarrow$ PI (

PD $\rightarrow$) $\{1\}^{n}$ 1 : (

PD $\rightarrow$,

L2 $\rightarrow \sigma$ | E | ED | L

2) <u>Expressions arithmétiques et expression booléennes</u>

Les productions sont :

$\sigma \rightarrow$ (BI, ω) σ , $\sigma \rightarrow$ (UN, ω) σ

(BI, ω) $\rightarrow \sigma\omega$ où ω est un opérateur binaire

(UN, ω) $\rightarrow \omega$ où ω est un opérateur unaire

$\sigma \rightarrow (\sigma)$ | (E) | N | IV | FD | LV

PIF $\rightarrow$ <u>IF</u>

E1 $\rightarrow$ PIF EB <u>THEN</u>

E $\rightarrow$ E1 σ <u>ELSE</u>

E $\rightarrow$ EE | Eσ

EB = (α [a] = bool., E)

(REMARQUE : à l'intérieur d'une expression simple, la distinction entre primaires, facteurs, etc.. est basée sur l'élément syntaxique précédent, si celui-ci est (BI, ω) ou (UN, ω)).

3) Expressionsde désignation

Les expressions de désignation sont toujours des segments finaux propres d'une instruction go to ou d'un élément de liste de switch . Les expressions non-simples ne peuvent être situées dans une liste de paramètres effectifs.

L est une étiquette (identificateur en position d'une expression de désignation simple).

S → SI[
EDS → SE] | L | (ED)
ED1 → PIF1 EB THEN
ED → EDS | ED1 EDS ELSE | ED ED

σI est la catégorie d'un identificateur en position d'indice dans un indicateur de fonction. Cet état n'est maintenu que si c'est une variable non-locale et si l'indicateur de fonction est spécifié.

4) Instructions simples

GTS → go to ED
instruction go to (IGT) . L'étiquette correspondante se trouve en α [a] .
AST → σ:= partie gauche d'une instruction d'affectation. La variable se trouve en α [a] , fonction d'adresse évaluée. Une liste de partie gauche compte autant de AST qu'elle compte de parties gauche. L'instruction est un mot de la forme

$$\{\text{AST}\}^{n} \text{AST E}$$

5) Instructions et blocs

⟨program⟩ → ⟨unlabelled block⟩

Un programme doit être un mot de la forme d'un bloc :

BEGIN $\{(DE, X)\ ;\}^{l}$ (DE, X) ; $\{\{E :\}^{m_n}\ S\ ;\}^{n}\{E:\}^{\sigma}$ S END

D → BEGIN

DB → D(DE, X) | DB $\{(DE, X)\ ;\}^{n}$ (DE, X)

où (DE, X) est une déclaration

E: une étiquette et S une instruction

DB est le début de bloc (⟨blockhead⟩)

ø → S ; | E: Cet état correspond à un élément d'un ⟨compound tail⟩, il prend le contrôle avant et après chacune des instructions d'un bloc ou d'une instruction composée.

6) Instructions conditionnelles

IFS → PIF EB THEN

(Cet état suit une proposition if qui n'est pas le début d'une expression. Il ne peut être précédé que par un autre état d'instruction non simple.).

CS → IFS $\{E:\}^{l}$ S ELSE | CS $\{E:\}^{n}$ CS

Les mots suivants sont aussi des instructions conditionnelles IFS S | IFS FS | $\{E:\}^{n}$ CS

où FS est une instruction for et S une instruction arbitraire à l'exception d'un FS ou d'un CS.

7. Les déclarations et les états de liste

Les états de listes réunissent des éléments qui ont une certaine propriété en commun

a) - liste d'identificateurs dans les déclarations : L1

b) - liste de paramètres formels LFP

c) liste d'expressions en indice L2

d) liste d'éléments d'aiguillage ESL

e) éléments d'une liste de FOR : EFL1, EFL2, EFL3

3.1.6. - CODIFICATION DES ETATS INTERNES ET DES ELEMENTS DE L'ALPHABET

Chaque état est codifié comme une puissance entière de 2, 2^n , $0 \leqslant n \leqslant 30$, le signe pouvant être + ou -, ce qui donne 62 codifications différentes.

Les états pour lesquels $0 \leqslant n \leqslant 5$ possèdent en plus une "zône de spécification" qui peut contenir un certain nombre de bits 2^m , $20 \leqslant m \leqslant 29$.

Chaque élément $S \in V_T$ est codé dans une seule mémoire. Celle-ci contient l'adresse de l'en-tête du chapitre correspondant cadrée à droite et une zône de spécification $20 \leqslant m \leqslant 29$ lorsque le chapitre groupe plusieurs éléments. Cette spécification est "héritée" par l'état correspondant. (Exemple : " ↑ " et " × " sont groupés dans le même chapitre. L'état syntaxique correspondant est celui des opérations binaires. Cet état portera donc toujours la spécification d'un certain opérateur binaire).

Les spécifications servent surtout dans les programmes de génération. Elles sont en principe ignorées par les règles syntaxiques de transition.
Voir en annexe 1 la table de codification des états internes des symboles de base et des identificateurs.

TABLE SYNTAXIQUE

(matrice de transition)

OPERANDES

	σ	$\sigma 1$	$\sigma 2$	σI
IDENT				
(	DIP | CH($\sigma 2$) | ENT (L2)			ST | DIP | CH($\sigma 2$) | ENT(L2)
)	R		IP | CH(σ)	
,	R			
ω	EV | CH(BI,ω)	CH(BI,ω)		ST | CH(BI,ω)
:=	ST | CH(AST)	CH(AST)		
IF				
THEN	R			
ELSE	R			
BEGIN				
END	R			
[	STI | CH($\sigma 1$) | ENT(L2)			STI | CH($\sigma 1$) | ENT(L2)
]	R	ADR | CH(σ)		
TYPE				
ARRAY				
,	R	INDICE | ENT(L2)	PE | ENT(L2)	DSP | R
SWITCH				
PROCEDURE				
FOR				
STEP	R			
UNTIL	R			
WHILE	R			
DO	R			
COMMENT				
SPECIFIC.				
NUMBER				
REAL. VALUE				
;	R			
STRING				
STEP				

ETATS D'EXPRESSIONS

	PIF	E1	E	E1D	ED	PD
ENT	ST|ENT(σ) ENT(P)	ST|ENT(σ) ENT(P)	ST|ENT(σ) ENT(P) SAUT2|R SAUT2|R	ENT(L) ENT(PD)	ENT(L) ENT(PD) SAUT^2|R SAUT^2|R	ENT(L) CH(ED)|S
W	ENT(UN,ω)	ENT(UN,ω)	ENT(UN,ω)			
IF	ENT(PIF)		ENT(PIF) SAUT2|R		ENT(PIF1)	ENT(PIF1
HEN	TEST|CH(IFS/E1/E1D)					
SE		SAUT1|CH(E)	SAUT2|R	SAUT^1 |CH(ED)	SAUT^2|R	
GIN						
ND			SAUT2|R		SAUT^2|R	
			SAUT2|R			
TE						
RAY						
	ENT(EFL2)|ENT(GT) ENT(L)|PDL|R		SAUT2|R		SAUT^2|R	
ITCH						
DURE						
			SAUT2|R			
P			SAUT2|R			
IL			SAUT2|R			
N	CH(FS)|ENT(EFL1) PDL|R		SAUT2|R			
NT						
IC.						
	STO|ENT(σ)	STO|ENT(σ)	STO|ENT(σ)	ENT(L)	ENT(L)	ENT(L)
VALUE	STA| ENT(σ)	STA|ENT(σ)	STA|ENT(σ)			

ETATS D'INSTRUCTIONS

	Ø	D	AST	GT	IFS	CS
	CH(σ)	ENT(σ)	ST|ENT(σ) ENT(P)	ENT(L) ENT(PD)	ENT(σ)	ENT(σ)
	R	ENT(Ø)	AFFECT|R ENT(UN)	SAUT|R	SAUT^{1}|R	SAUT^{2}|R
	CH(PIF)	ENT(PIF)	ENT(PIF)	ENT(PIF1)		ENT(PIF)
	CH(D) R	ENT(D) FEBL|S	AFFECT|R AFFECT|R	SAUT|R SAUT|R	SAUT^{1}|CH(CS) ENT(D) SAUT^{1}|R	ENT(D) SAUT^{2}|R
	SP|CH(LFP)|ENT(PDC)	OUBL|ENT(DE,T)				
	SP|CH(LFP)|ENT(PDC)	OUBL|ENT(DE,A)				
	SP|CH(LFP)|ENT(PDC)	OUBL|ENT(DE,S)	AFFECT|R			
E	SP|CH(LFP)|ENT(PDC)	OUBL|ENT(DE,P)				
	CH(FS)|ENT(EFL1)	ENT(FS)|ENT(EFL1)			ENT(FS)|ENT(EFL1)	EFS|E(EFL1)
			AFFECT|R	SAUT|R		
			AFFECT|R			
			AFFECT|R			
			AFFECT|R			
	CH(GT)	ENT(GT) CNT1			ENT(GT)	ENT(GT)
C.	SP|CH(LFP)|ENT(PDC)					
	CH (σ)	ENT(σ)	STO|ENT(σ) STA|ENT(σ)	ENT(L)		
LUE		ET|ENT(Ø)			ET|ENT(Ø)	ET|ENT(Ø)
				SAUT|R	SAUT^{1}|R	

ELEMENTS DE LISTE DE FOR

	FS	EFL1	EFL2	EFL3
NT	ENT(σ)		ST|ENT(σ) ENT(P)	
	FEBL|ET|R		ENT(UN,ω)	
IF	ENT(PIF)		ENT(PIF)	
EN				
SE				
IN	ENT(D)			
D	FEBL|ET|R			
E				
AY		ENT(GT)|ENT(L)|PDL|R	CH(DI,-)|ENT(BI,×)|ENT(BI,⩽)	
TCH				
URE				
R		ENT(EFL2)|ENT(L)|PDL|R	ET|ENT(AST)|ENT(BI,+)|ENT(P) ET	
IL				
LE		ENT(PIF)		
	FF|ET|ENT(∅)	PDL|R	CH(BI,-)|ENT(BI,+)|ENT(BI,⩽)	
TO	ENT(GT)			
T				
IC.				
R	ENT(σ)		STO|ENT(σ)	
VALUE				
G		ET|ENT(AST)	CH(EFL3)|PDL|R	F

ETATS DES LISTES ET DE DECLARATIONS

	L1	L2	LFP	ESL	DE	PDC
NT		ST|ENT(σ)/L/σ1	DCP| ENT(σ)	ENT(L)	DECLA|ENT(L1)	SPEC|ENT(L1)
	PF|S	ENT(P)		ENT(PD)		
	ELIMIN|R	R			NUMER|S	
	R		FCP|S	FS|R	NUMER|S	S
		ENT(UN,ω)				
	DDS|CH(ELS)					
F		ENT(PIF)	DCP|ENT(PIF)	ENT(PSI)		
N						
E						
IN			DCP| ENT(D)			
D			FCP| S			
	CH(L2)					
		R				
E			SP|ENT(PDC)			
AY			SP| ENT(PDC)		CH(DE,TA)	SPEC|R
	S	R		DS|ENT(ESL)	BS|ENT(L2)	
CH			SP|ENT(PDC)			
URE			SP|ENT(PDC)		CH(DE,P)	SPEC|R
L						
E						
O						
T			CNT1			
IC.			SP|ENT(PDC)			
		STO| ENT(σ)		ENT(L)		
LUE		STA| ENT(σ)				
		R	ET|ENT(Q)		BI|ENT(L2)	
O						

EXPRESSIONS SIMPLES

	BI,	UN,	P	L	S
IDENT	ST|PRE|ENT(σ) PRE|ENT(P)	ST|PRE|ENT(σ) PRE|ENT(P)	ST|ENT(σ) ENT(P) CH(σ)	 PDL|DES|R	ST|ENT(σ) ENT(P)
(				PDL|DES|R	
)					
;					
ω	PRE|ENT(UN,ω)		ENT(UN,ω)		ENT(UN,ω)
:=					
IF			ENT(PIF)		ENT(PIF)
THEN					
ELSE				PDL|DES|R	
BEGIN					
END				PDL|DES|R	
[				ST|CH(S)	
]					ISW|S
TYPE					
ARRAY					
,				PDL|DES|R	
SWITCH					
PROCEDURE					
FOR					
STEP					
UNTIL			ET|CH(σ)|R		
WHILE					
DO					
GO TO					
COMMENT					
SPECIFIC.					
NUMBER					STO|ENT(σ)
BOOL.VALUE					
:					
STRING					
REP				PDL|DES|R	

3.2. - DECLARATIONS ET BLOCS

3.2.1. - LES PRINCIPES DE L'ATTRIBUTION D'ADRESSE

ADRESSES FIXES ET ADRESSES VARIABLES

Parmi les quantités définies dans l'en-tête d'un bloc on peut distinguer celles qui occupent un nombre de mémoires variables au cours du déroulement du programme : tableaux à bornes variables et procédures. Si l'on sépare en deux zônes distinctes l'ensemble des mémoires nécessaires aux variables d'un certain bloc, il est toujours possible de désigner chaque variable déclarée par une adresse fixe relative au bloc dans lequel cette variable est déclarée de la façon suivante :

1) - Tout identificateur de variable simple, de tableau, de procédure ou d'aiguillage occupe un nombre fixe de mémoires (ce nombre pouvant être différent pour une variable simple, un tableau, etc..) dans l'une de deux zônes, dite "zône d'adresses (relatives) fixes".

2) - Les mémoires de cette zône contiennent une valeur(numérique ou booléenne) dans le cas d'une variable simple ou bien une (ou plusieurs) référence sur l'emplacement en "zône d'adresse variable" de l'ensemble des valeurs.

Il convient de noter que l'on peut incorporer dans la zône d'adresses fixes les tableaux à bornes constantes, le nombre de leurs éléments étant déterminé.

Ces deux zônes distinctes peuvent être placées en mémoire de façon à former un seul "vecteur". Dans ce cas le nombre total des identificateurs déclarés doit être connu avant la détermination de la zône d'adresse variable. La position séparée a l'avantage de n'exiger aucun ordre dans la suite de déclaration mais elle rend plus difficile la meilleure utilisation de la mémoire - surtout en cas de zônes distinctes pour les variables déclarées remanent.

3.2.2 - ADRESSES RELATIVES FIXES ET BLOCS

Après réduction à une ou plusieurs adresses fixes de toute quantité déclarée à l'intérieur d'un seul bloc l'adressage d'une quantité ne dépend que de son adresse relative au début du bloc et de l'ordre dynamique des blocs (insertion d'un bloc par une instruction procédure). Chaque variable est repérée par un couple d'entiers (i, j) , i étant le niveau de bloc déterminé par l'ordre statique sur les blocs et j l'adresse relative fixe de la variable. L'adresse définitive d'une variable A_{ij} est donnée par

$$Adr\ (A_{ij}) = j + \langle h_i \rangle$$

où $\langle h_i \rangle$ est l'adresse de A_{io}. Cette adresse dépend de l'activation des niveaux de nomenclature. Celle-ci peut-être exprimée en forme d'une table

Niveaux de nomenclature :	N_k	N_{k-1}	N_{k-2}	N_{k-3}	N_{k-4}
	$h_{o,k}$	$h_{o,k-1}$	$h_{o,k-2}$	$h_{o,k-3}$	$h_{o,k-4}$
première insertion d'une série					
de blocs par une inst.procédure :		$h_{1,k-1}$	$h_{1,k-2}$	$h_{1,k-3}$	
deuxième insertion :			$h_{2,k-1}$	$h_{2,k-2}$	

Chaque activation de procédure ajoute un élément dans les colonnes correspondants aux N_j qu'elle contient. A chaque niveau, c'est le dernier h d'une colonne qui est utilisé - sauf dans les sous-programmes pour les paramètres effectifs appelés par nom. Chaque appel d'un tel sous-programme substitue $h_{j-1,k}$ pour $h_{j,k}$ pour l'ensemble des N_k de la procédure. Les h_k antérieurs doivent être gardés en zône de mémoires d'adresses fixes.

Ce schéma résout le problème d'attribution de mémoire notamment pour les procédures définies récursivement.

3.2.3 - ADRESSAGE SIMPLIFIE - ADRESSES FIXES ABSOLUES

Le procédé d'adressage par une fonction d'adresse qui dépend de l'histoire du programme en cours n'est réalisable sans ralentissement considérable de la totalité des opérations, que si l'on dispose d'un certain nombre de registres d'indexe et d'un dispositif d'adressage indirect.

Une autre version pour l'attribution d'adresses qui donne à chaque variable une adresse fixe et absolue, doit nécessairement recourir à la réécriture (copie) de parties du programme. Si l'appel par nom d'un paramètre est supprimé dans les procédures définies récursivement, il suffit de copier, c'est-à-dire de transférer en zône d'adresses variables l'ensemble des variables d'une procédure chaque fois qu'elle est activée dans une série de

récursions. Les variables transférées ne sont pas adressables directement, elles sont seulement conservées pendant l'insertion de la même procédure à un niveau supérieur. La dernière activation d'une procédure utilise toujours les mémoires d'adresses fixes pendant que les valeurs des variables des mêmes adresses, mais provenant d'activations précédentes, se trouvent en mémoire d'adresse variable. A chaque fin d'activation de la procédure un ensemble de variables est retransféré dans les mémoires d'adresse fixes de la procédure. Pour les détails voir les paragraphes sur la déclaration de tableau et sur l'appel de procédure. Les avantages de cette version simplifiée résident en l'absence des opérations fréquentes sur les $\langle h_k \rangle$. L'adressage des variables simples peut être totalement incorporé dans les opérations sur ces variables.

3.2.4. - LISTE D'IDENTIFICATEURS

Toutes les quantités définies dans l'en-tête d'un bloc sont incorporées dans une liste de déclarations. Chaque quantité y occupe trois mémoires dont deux sont prises par l'identificateur correspondant (plus précisément par ses dix premiers signes - lettres ou chiffres - significatifs) et servent à la reconnaissance de la quantité. La troisième mémoire contient la codification de la quantité et une adresse.

En tête de la liste viennent les noms et les codifications des procédures standard, puis classées par niveau de nomenclature (programme - bloc - corps de procédure) et séparées par des marques de niveau, les quantités une par une telles qu'elles ont été trouvées dans les "en-tête de bloc".

Chaque fois qu'un identificateur est rencontré au cours de l'analyse d'un programme donné, il est retrouvé dans la liste et la codification de la quantité correspondante est substituée. La recherche dans la liste commence par le niveau de nomenclature le plus récent et elle est terminée lorsque le nom cherché est rencontré la première fois. Toute autre quantité de même nom de la liste a été déclarée antérieurement et n'est pas accessible.

A chaque fin de bloc la section de la liste correspondante, qui est toujours la dernière qui a été remplie, est "effacée" et les mémoires correspondantes sont réutilisées lors des prochaines déclarations rencontrées dans l'analyse séquentielle du programme donné.

3.2.5 - LA DECLARATION DE PROCEDURE

Deux mémoires d'adresses fixes sont réservées pour un identificateur de procédure. La première reçoit la valeur (lorsqu'il s'agit d'un indicateur de fonction), la deuxième reçoit l'adresse de la première instruction du corps de la procédure ("l'adresse" de la procédure) lorsque le bloc correspondant est activé.

Les mémoires "en-tête" du programme généré correspondant contiennent la codification et l'adresse de chacun des paramètres formels, un emplacement pour l'adresse du sous-programme des paramètres effectifs correspondants, le nombre des paramètres formels et le nombre de mémoires nécessaires pour les variables locales.

Avant l'activation d'un corps de procédure un programme interprétatif est appelé pour le calcul des valeurs des expressions des paramètres spécifiés value. Ces valeurs sont obtenues et transférées une par une dans les mémoires correspondantes des paramètres formels. Ces variables ne se distinguent par la suite pas des variables locales, elles sont considérées locales au corps de la procédure.

Un paramètre formel qui n'est pas appelé par valeur correspond à un saut vers un sous-programme fermé dont l'adresse se trouve dans la mémoire correspondante à ce paramètre. Lorsqu'un tel sous-programme est activé l'état d'encombrement des divers registres doit être enregistré.

3.2.6. - LA DECLARATION DE TABLEAU

A chaque entrée dans un bloc qui contient une ou plusieurs déclarations de tableau, emplacement et nombre d'éléments doivent être déterminés dynamiquement pour chaque tableau, c'est-à-dire que le compilateur doit fournir un programme et des données, incorporés dans le programme généré. La première partie de ce programme prend la forme d'un sous-programme ouvert. Il calcule les composantes du "vecteur d'information" qui contient les paramètres de la fonction d'attribution d'adresse pour chaque SECTION de TABLEAU et il présente ce vecteur, l'adresse fixe du premier tableau dans la section et le nombre de tableaux de la section comme paramètres effectifs à un sous-programme fermé qui copie le vecteur d'information pour chaque tableau de la section, réserve le nombre de mémoires nécessaires dans la zône d'adresses variables et transfère l'adresse de la première mémoire du vecteur d'information ("l'adresse" du tableau) dans la mémoire d'adresse fixe correspondante.

3.2.6.1. - GESTION DYNAMIQUE DE LA MEMOIRE

Chaque déclaration de tableau dans un bloc B_i change le contenu de la mémoire d'encombrement F_i correspondante qui indique à tout moment la première adresse libre en zône d'adresses variables. Lorsqu'un bloc est activé, le contenu de la mémoire F_{i-1} du bloc englobant (prédécesseur immédiat dans le sens d'ordre statique sur les blocs) est transféré en F_i et l'indicateur d'adresse est positionné sur F_i.

Le sous-programme fermé de copie et de réservation trouve alors à chaque instant l'adresse de F_i à l'aide de l'indicateur d'adresse et augmente son contenu du nombre de mémoires nécessaires pour chaque tableau et pour son vecteur d'information.

3.2.6.2. - LE VECTEUR D'INFORMATION

Le vecteur d'information pour un tableau de dimension N contient N + 2 mots. L'adresse de sa première mémoire constitue en quelque sorte l'adresse du tableau.

Les N + 2 mots du vecteur contiennent dans l'ordre :

Le nombre de dimensions du tableau, L, le nombre total d'éléments du tableau, dans la première mémoire.

$$L = \prod_{i=1}^{n} n_i$$, où n_i est la longueur de l'ième arête, i = 1,2,..., n

La deuxième, troisième,..., (n + 1) - ième mémoire contient

$$n_i = BS_i - BI_i + 1$$ BS ..... borne supérieure
BI ..... borne inférieure

La (n + 2) -ième mémoire contient l'adresse de $A[0, 0,..., 0]$ (cet élément du tableau n'est pas toujours contenu dans les bornes. Cette adresse est fixe pour les tableaux à bornes constantes et peut être incorporé dans une instruction générée).

- Calcul du vecteur d'information

Soit BS_i et BI_i la borne supérieure et la borne inférieure respectivement, de la dimension i, $1 \leqslant i \leqslant n$.

Adr(a $[k_1, \ldots, k_n]$), l'adresse variable indicée $a[k_1, \ldots, k_n]$, $k_1, \ldots, k_n$ donnés, est déterminée de la façon suivante :

$$Adr(a[k_1, \ldots, k_n]) = Adr(a[BI_1, \ldots, BI_n]) + (\ldots(k_1 - BI_1)(BS_2 - BI_2 + 1) + k_2 - BI_2)\ldots)(BS_n - BI_n + 1) + k_n - BI_n$$

ou

$$Adr(a[k_1, \ldots, k_n]) = Adr(a[BI_1, \ldots, BI_n] + (\ldots(k_1 - BI_1) n_2 + k_2 - BI_2) n_3 + k_3 - BI_3)\ldots) n_n + k_n - BI_n$$

$$Adr(a[k_1, \ldots, k_n]) = Adr(a[BI_1, \ldots, BI_n]) + (\ldots(k_1 n_2 + k_2) n_3 + k_3)\ldots) n_n + k_n - ((\ldots(BI_1 n_2 + BI_2) n_3 + BI_3) \ldots) n_n + BI_n$$

et avec $k_1 = k_2 = \ldots = k_n = 0$ on obtient l'adresse réduite:

$$Adr(a[0, \ldots, 0]) = Adr(a[BI_1, \ldots, BI_n]) - ((\ldots(BI_1 n_2 + BI_2) n_3 + BI_3)\ldots) n_n + BI_n$$

et par substitution de la formule pour l'adresse réduite on obtient l'adresse d'une variable indicée :

$$Adr(a\left[k_1,\ldots,k_n\right]) = Adr(a\left[0,\ldots,0\right]) + (\ldots(k_1 n_2 + k_2) n_3 + k_3)\ldots) n_n + k_n$$

- Paramètres formels spécifiés "tableau" .

Il n'est pas nécessaire de faire une différence entre un identificateur de tableau déclaré et un paramètre formel spécifié tableau lors du calcul d'adresse d'une variable indicée. Dans les deux cas l'adresse fixe de la variable indiquera la première mémoire d'un vecteur d'information. La seule différence réside dans le fait que cette adresse est déterminée lors de la déclaration du tableau correspondant dans le premier cas et qu'elle est déterminée par le paramètre effectif de l'instruction procédure dans le deuxième cas.

3.2.7. - ETIQUETTES ET EXPRESSIONS DE DESIGNATION

D E F I N I T I O N

Les étiquettes ne sont pas déclarées explicitement au début des blocs. La définition d'un identificateur ou d'un nombre sans signe comme étiquette est donnée implicitement par son emploi. Distinguons d'abord

A) l'occurrence de l'étiquette dans une expression de désignation

B) la définition (par rapport à un niveau de nomenclature) par les chaînes :

⟨identificateur⟩ : ⟨instruction⟩ ou bien
⟨entier sans signe⟩ : ⟨instruction⟩

qui équivaut en fait à une déclaration explicite dans l'en-tête du bloc.

Supposons que le programme généré ne subisse aucune modification pendant son exécution et faisons exception pour A) des étiquettes faisant partie d'un élément d'une liste d'aiguillage. Dans ce cas, la valeur d'une étiquette est fixe. Elle est déterminée par B) et elle sera connue à un certain moment pendant la compilation. Ce fait permet d'évaluer les étiquettes statiquement, c'est-à-dire de remplacer pendant la compilation, la même étiquette partout dans les occurrences suivant A) par la valeur dont il s'agit.

Cette opération peut-être effectuée à la fin du plus grand bloc dans lequel l'étiquette est locale. Lors de l'exécution du programme, aucune mémoire ne contient la valeur d'une étiquette, à l'exception de paramètres formels et des éléments de listes d'aiguillages, elles sont incorporées dans les instructions du programme généré.

Pendant la compilation, une liste d'étiquettes rencontrées est établie. Cette liste, comme celle des quantités déclarées, contient des groupes de trois mémoires consécutives par étiquette. Les deux premières contiennent le nom, la troisième l'adresse que désigne l'étiquette suivant B) (c'est-à-dire sa valeur), un numéro courant et l'adresse de la dernière instruction machine qui se réfère à cette étiquette.

A chaque début d'un niveau de nomenclature, mais aussi au début de l'instruction qui suit une proposition FOR, une marque de séparation de niveaux est introduite dans la liste. Chaque fois qu'une étiquette est rencontrée, elle est comparée au contenu de la liste correspondant au niveau le plus récent, c'est-à-dire jusqu'à la dernière marque de niveau. Si elle ne s'y trouve pas, un nouveau groupe de trois mémoires est rajouté, sinon la nouvelle occurrence est marquée dans la mémoire adresse.

A la fin d'un niveau, toutes les étiquettes de ce niveau sont examinées. Deux cas sont possibles : l'étiquette est définie dans ce niveau, ou bien dans un niveau inférieur. Dans le premier cas, elle est locale à ce niveau et la partie B) de la mémoire adresse contient une

adresse (sa valeur). Dans le deuxième cas, la partie B) est vide.

Dans le premier cas, l'adressage est définitif c'est-à-dire la substitution d'adresse dans toutes les instructions du programme généré qui se réfèrent à cette étiquette peut être effectuée et le groupe de mémoires affecté à l'étiquette est "effacé". Un chaîne de reprise, qui utilise la partie adresse des instructions générées intéressées par cette étiquette permet la compression de l'information dans un seul groupe de trois mémoires par étiquette, ce qui signifie un certain gain de place et de temps de recherche.

La partie A) peut être vide sans qu'il s'agisse d'une erreur. Une telle étiquette, définie mais jamais utilisée peut être interprétée comme une sorte de commentaire qui n'a pas de répercussion sur le programme généré.

Si, par contre, la partie B) est vide, le groupe de mémoires de la liste est incorporé dans le niveau inférieur.

3.2.7.1 - LA DECLARATION D'AIGUILLAGE

La déclaration d'aiguillage correspond à un sous-programme qui a autant de sorties que la liste d'aiguillage compte d'éléments. Pour chaque indicateur d'aiguillage dans un élément d'une liste d'aiguillages, le programme se réfère récursivement à un nouveau sous-programme.

Chaque étiquette dans un élément de la liste correspond à l'appel de sa valeur dans un registre. C'est le contenu de ce registre qui figure comme valeur de l'indicateur d'aiguillage. Pour chaque déclaration d'aiguillage le compilateur génère une table avec les adresses de début des programmes correspondant aux différents éléments de la liste. Si la valeur de l'expression en indice est N, c'est la N-ième adresse de cette table qui est choisie. Cette table fait partie du programme généré et doit l'accompagner. Les valeurs courantes des étiquettes d'une liste d'aiguillage doivent être recalculées au début de chaque bloc dans lequel la déclaration est valable.

3.3.1. - L ' I N S T R U C T I O N FOR

La proposition FOR est l'écriture condensée d'une séquence d'instructions plus simples : instructions d'affectation, instructions GO TO et instructions IF. Cette définition de la proposition FOR (PF) par d'autres éléments de la syntaxe sera ici appelée son développement. Celui-ci n'est pas déterminé de façon unique pour une instruction FOR donnée, mais on choisit un développement déterminé et distinct pour chaque forme d'élément de la liste FOR. Les états syntaxiques d'une PF sont composés par trois espèces d'éléments différents : ceux qui correspondent aux expressions qui forment les éléments de la liste, ceux des instructions du développement et finalement les états qui correspondent aux règles de succession des instructions d'un développement. Ces derniers sont appelés EFL (états syntaxiques attachés aux éléments d'une liste de FOR).

Le développement prend une des formes suivantes :

⟨for list element⟩ ::= ⟨label⟩ : ⟨assignement statement⟩ ;
⟨go to statement⟩ |
| ⟨label⟩ : ⟨assignement statement⟩ ; ⟨if statement⟩ |
| ⟨assignement statement⟩ ; ⟨go to statement⟩ ; ⟨label⟩ :
⟨assignement statement⟩ ; ⟨label⟩ : ⟨if statement⟩

Les instructions GO TO et les étiquettes du développement demandent l'introduction d'un certain nombre

d'étiquettes artificielles. Ces étiquettes sont construites à l'aide d'un compteur. Le nom d'une étiquette artificielle est composé par le contenu du compteur suivi d'une lettre. Cette combinaison ⟨chiffre⟩ ⟨lettre⟩ étant interdite pour les identificateurs, on évite ainsi la confusion avec un identificateur du programme et les étiquettes artificielles peuvent être incorporées dans la liste des étiquettes et traitées normalement sans distinction.

Au début de l'instruction qui suit la proposition un niveau de "bloc" est ouvert dans la liste des étiquettes. Ainsi, deux effets sont obtenus : les sauts conduisant à l'intérieur d'une telle instruction sont supprimés dans le programme généré, et les étiquettes correspondantes apparaissent comme erreurs à la fin du procédé de génération. Le deuxième effet réside dans le fait, que les étiquettes artificielles créées dans la PF précédente, peuvent être réemployées dans le développement d'une PF imbriquée.

Dans le détail, le développement d'une proposition FOR prend une des formes suivantes :

FOR V: = E1, E2,..., EN DO

sera défini par

```
Begin  switch T := 1,2,...,N, SUITE ;
       integer i ;
                i := 1 ;
1:   V:= E1 ; go to S ;
2:   V:=E2 ;  go to S ;
              - - -
N:   V:=EN ;
 S: γ ;   i:=i+1 ; go to  T[i]   ;  SUITE : end
```

La séquence en langage machine peut être quelque peu simplifiée, elle ne contient pas un switch, mais un saut vers un sous programme :

V:= E1 ; go to S ; (rupture de séquence avec conservation du compteur ordinal : RCV)
V:= E2 ; go to S ;
- - -
V:= EN ; (chargement du C.O. avec SUITE) ;
S: γ ; (instruction γ en forme de sous-programme avec renvoi du C.O. au début et saut vers cette adresse à la fin) ; SUITE :

Elément avec WHILE

FOR V:= E1 WHILE B1, ..., EN WHILE BN DO

devient

```
IE :    V := E1 ;                    (I = 1)
    IF B1 THEN
                BEGIN
                        GO TO S ;    (RCV)
                        GO TO IE
                END ;
```

```
IE:     V:=EN                        (I = N)
    IF BN THEN  (chargement du C.O. par IE) ;
        S = γ ;  (instruction γ en forme de sous-programme)
```

Les étiquettes artificielles sont IE (i = 1, 2, ..., N) et S.

Elément avec STEP et UNTIL

FOR V:=E11 STEP E12 UNTIL E13,...,EN1 STEP EN2 UNTIL EN3 DO

On peut distinguer les trois cas suivants :

a) EI2 est une constante positive

b) EI2 est une variable simple

c) EI2 est une expression arithmétique plus compliquée

V:=E11 ; (I = 1)

STEP⟨constante positive⟩	STEP ⟨variable simple⟩	STEP⟨expression arithmétique⟩
		Chargement du C.O. par IF
IG: GO TO IF ;	IG: GO TO IF ;	IG: W:= (E12) ; (sous-programme)
IE: V:=V+(E12);	IE: V:=V+(E12);	IE: GO TO IG (RCV); V:=V + (W) ;
IF: IF(V-(E13))⩽0 THEN	IF: IF(E12)x(V-(E13) ⩽0 THEN	IF: IF(W)x(V-(E13) ⩽0 THEN

BEGIN GO TO S ; (RCV)

GO TO IE END ;

---

ou, au lieu de cette dernière instruction composée, on a pour le dernier élément d'une liste :

(chargement du C.O. par IE) ;

S: γ (- instruction γ en forme de sous-programme -) ;

IE, IF, IG (i = 1,2,...,N) et S sont des étiquettes artificielles.

Remarque :

La valeur des variables de la proposition peut être changée par un effet de bord d'une procédure ou dans l'instruction qui suit la proposition.

Notamment le calcul du "STEP" et de l'expression suivant le séparateur UNTIL peut changer la valeur de la variable contrôlée. Dans le cas d'une variable indicée, la variable même peut changer, sauf si c'est la variable contrôlée. La fonction d'adresse de cette dernière sera déterminée une seule fois au début de la proposition pour tous les éléments de la liste. Sa valeur numérique par contre, peut être changée à tout moment par un effet de bord.

Exemple 3.3.1.

Génération d'un programme en langage machine pour une ⟨for-clause⟩ : voir annexe 2.

3.3.2. - L'APPEL DE PROCEDURE

3.3.2.1. - G E N E R A L I T E S

Du point de vue de la syntaxe, un appel de procédure peut être une instruction (instruction-procédure) ou une expression (indicateur de fonction). L'un des deux problèmes principaux, celui de la transmission d'information concernant les paramètres effectifs, reste inchangé dans les deux cas. L'autre problème est celui d'attribution d'adresse pour les quantités définies dans le corps de la procédure. Le cas de l'indicateur de fonction paraît alors comme le plus général à cause de la présence possible d'un certain nombre de résultats intermédiaires (ou valeurs anonymes).

Dans ce qui suit on ne fait pas de différence entre les deux cas. L'instruction procédure sera considérée comme un cas particulier (et plus simple) d'un appel de procédure.

Pour le reste le schéma d'un appel est imposé par certaines caractéristiques des procédures en Algol :

A) Remplacement des paramètres formels par les paramètres effectifs partout dans le corps de la procédure (appel par nom des paramètres).

B) Règles de correspondance entre paramètres formels et paramètres effectifs.

C) L'effet de bord (side effect) résultant d'un appel de procédure.

D) Procédures récursives.

Les deux derniers points se rattachent au problème d'attribution d'adresse.

Il est bien connu [1,12] , que le point A) nécessite - en général - une compilation indépendante de l'appel d'une procédure, c'est-à-dire sans référence à la déclaration correspondante, la liaison entre l'appel et la déclaration ne pouvant être réalisée que dynamiquement. La compilation de l'appel de procédure génère donc uniquement des morceaux de code, qui doivent être reliés correctement par un programme interprétatif. Celui-ci doit donc avoir accès à la déclaration de procédure correspondant à un appel donné, ce qui implique le maintien dans le programme généré d'un certain nombre de renseignements concernant la déclaration.

Ce programme de liaison peut aussi tester la compatibilité entre le nombre et les types des paramètres formels et des paramètres effectifs.

Il peut finalement être chargé - entièrement ou partiellement - de la gestion dynamique de la mémoire en cas d'appels récursifs - c'est-à-dire dans tous les cas où une procédure est de nouveau appelée avant que le, ou les appels précédents soient définitivement terminés. On peut distinguer deux cas : l'appel récursif (une procédure constitue un, ou plusieurs, de ses propres paramètres effectifs) et la définition récursive (appel d'une procédure à l'intérieur de son corps), ce dernier cas étant le plus général.

Le programme interprétatif se compose de plusieurs parties plus ou moins indépendantes et attachées aux divers problèmes mentionnés ci-dessus.

3.3.2.2. - COMPILATION D'UN APPEL DE PROCEDURE L'APPEL PAR NOM DES PARAMETRES FORMELS

Les instructions qui composent un appel de procédure forment une série de sous-programmes, un pour chaque paramètre effectif. Ces sous-programmes sont entourés d'un certain nombre de codes auxiliaires qui, d'une part, fournissent une description des paramètres effectifs et d'autre part, appellent le programme interprétatif.

Le point A) - appel par nom - exige, que le calcul d'un paramètre effectif puisse être demandé à partir de n'importe quel endroit dans le corps de la procédure. Ce calcul doit être fait par un sous-prgramme fermé qui range le résultat de son calcul dans un certain registre standard. La première possibilité est choisie pour les expressions de désignation, la deuxième pour toutes les autres expressions.

Dans le cas d'appel par valeur, ces sous-programmes sont appelés par le programme interprétatif une seule fois avant d'entrer dans le corps de la procédure et les valeurs obtenues sont transférées dans les mémoires des paramètres formels correspondants.

Dans les sous-programmes pour les paramètres effectifs on distingue essentiellement trois niveaux d'action suivant la nature du paramètre :

1) le calcul d'une expression
2) le transfert d'un résultat (valeur numérique d'une expression) dans une mémoire intermédiaire.
3) le chargement d'un certain registre (registre d'index) par l'adresse du paramètre effectif.

Les paramètres effectifs peuvent être groupés en quatre catégories :

Paramètre effectif		Niveau d'action du sous-programme
A) Expression arithmétique ou booléenne, indicateur de fonction		1 - 2 - 3
B) Expression de désignation		1 - 3
C) Variable simple, déclarée ou spécifiée, nombre ou valeur booléenne, variable indicée identificateur de tableau identificateur d'aiguillage	déclaré ou spécifié	3 (après évaluation de la fonction d'adresse si c'est une variable indicée)
D) Identificateur de procédure, déclaré ou spécifié		1 - 2 - 3

Ce dernier cas demande des explications supplémentaires. Au moment de la compilation d'un appel de procédure on n'a pas d'information sur les paramètres formels de

cette procédure (c'est vrai seulement, si l'identificateur de la procédure que l'on appelle est lui-même le paramètre formel d'une autre procédure). Si l'on trouve à la place d'un paramètre effectif un identificateur deprocédure, il s'agit soit d'un identificateur de fonction sans paramètre, soit du nom d'une procédure. La compilation le traite comme un indicateur de fonction sans paramètre. Dynamiquement, les deux interprétations sont possibles et le choix s'effectue suivant la spécification du paramètre formel correspondant. Si celui-ci est spécifié TYPE le sous-programme pour le paramètre effectif est exécuté tel que, si par contre la spécification est PROCEDURE ou TYPE PROCEDURE, la partie 3 (appel d'une adresse dans un registre standard) seulement est exécutée.

- LE TRANSPORTEUR DES REGISTRES

Au moment d'entrer dans un des sous-programmes pour les paramètres effectifs, les registres peuvent contenir un certain nombre de résultats qui doivent être conservés. Le nombre et l'emplacement de ces résultats varie évidemment d'un appel à l'autre. Le calcul d'un paramètre effectif fait donc intervenir un sous-programme standard TR(R) qui fait partie de l'interpréteur et qui transfère les contenus de tous les registres, qui lui sont indiqués par un des registres réservés à ce but, dans la zône d'adresses variables. En remplaçant l'adresse de retour du sous-programme par sa propre adresse, il s'appelle lui-même à la sortie du S.P. pour le paramètre effectif et l'état précédent des registres est rétabli.

3.3.2.3. - CORRESPONDANCE ENTRE PARAMETRES EFFECTIFS ET PARAMETRES FORMELS - LE PROGRAMME ADMINISTRATIF

La catégorie et le type d'un paramètre effectif doivent être compatibles avec la catégorie et le type du paramètre formel correspondant. Les catégories de deux paramètres correspondants doivent être strictement identiques. A l'intérieur de chaque catégorie (si la catégorie admet un type), les types réel et entier (respectivement entier - réel) sont considérés comme compatibles.

Les transferts de type sont traités statiquement, c'est-à-dire les opérations de conversion de virgule flottante en virgule fixe et de fixe en flottante, sont déterminées au moment de la compilation et incorporées dans le programme généré. Les transferts de type entre paramètres effectifs et paramètres formels sont donc entièrement à la charge du programme interprétatif. Deux cas doivent être distingués :

A) le paramètre formel fait partie d'une PARTIE GAUCHE

B) le paramètre formel fait partie d'une expression.

Les opérations à performer sont respectivement :

	A	B
1)	Effectuer le transfert de type du contenu du registre indiqué à l'appel de TR(R)	Effectuer le transfert de type du contenu de la mémoire indiquée par le registre d'indexe.
2)	—	Transfert du résultat de 1) dans une mémoire intermédiaire
3)	—	Changement du contenu du registre d'indexe.

(Dans une version simplifiée ce transfert de type n'est pas possible).

Avant d'entrer dans le corps de la procédure, les adresses de début des sous-programmes pour les paramètres effectifs sont transférées une par une dans les mémoires qui ont été prévues pour elles au début du programme généré lors de la déclaration de procédure.

Les liaisons sont établies par la première partie du programme interprétatif appelé par la suite programme administratif, P.A.

Le P.A. teste d'abord la compatibilité des catégories et types en question puis il transfère l'adresse de début du sous-programme de chacun des paramètres de la liste.

Les changements de type sont effectués par le programme TR(R) de la façon suivante : A l'entrée dans un sous-programme le type du paramètre formel est marqué dans un registre. TR(R) compare ce type au type du paramètre effectif et charge l'adresse de sortie de ce sous-programme par l'adresse de sortie ordinaire - d'après le résultat de la comparaison des types.

3.3.2.4. - L'EFFET DE BORD DES PROCEDURES

Toute procédure peut changer la valeur d'une variable non-locale au corps de cette procédure. Les indicateurs de fonction, contenus dans une expression, changeront en général la valeur des autres variables contenues dans la même expression.

La règle suivante a été adoptée : les variables qui se trouvent à gauche de l'indicateur de procédure ne peuvent être changées si elles sont mises entre parenthèse, autrement leurs valeurs ne sont pas définies. Les variables à droite de l'indicateur de fonction seront toujours affectées par un effet de bord éventuel.

3.3.2.5.- PROCEDURES RECURSIVES

Un certain schéma de gestion dynamique de la mémoire par un programme interprétatif est présenté dans ses détails. Ce schéma permet la compilation d'une grande classe d'algorithmes définis récursivement.

Nous rappelons d'abord quelques conventions adoptées :

1) l'ensemble de variables d'une procédure, les paramètres formels y inclus, correspond à un certain vecteur V en zône d'adresses fixes. Ces adresses figurent dans les instructions machine du corps de procédure. Ces instructions ne sont jamais modifiées et tout calcul fait inter-

venir uniquement ce vecteur.

2) l'ensemble de résultats intermédiaires du corps de la procédure (variables anonymes) correspond à un autre vecteur $\overline{V}$ à adresses fixes.
3) Les renseignements concernant les paramètres effectifs sont placés dans les mémoires dites "en tête de procédure".
4) L'ensemble des mémoires des points 1) à 3) est conservé en zône adresses variables pendant l'activation de la procédure.

3.3.2.5.1. - LA ZONE D'ADRESSES VARIABLES

Chaque début de bloc ou de corps de procédure B_i correspond à un certain "vecteur" d'adresses fixes. La première mémoire de ce vecteur s'appelle F_i. C'est une mémoire auxiliaire qui contient à tout moment de l'activation de ce bloc l'adresse de la première mémoire libre en zône d'adresses variables (mémoire d'encombrement). Une mémoire distincte J, qui joue le rôle d'un registre d'index, indique l'adresse de la mémoire F_i correspondant au dernier bloc activé.

Les opérations de gestion de mémoire sont donc les suivantes

1) Au début d'un bloc B_i

$\langle F_{i-1} \rangle \Longrightarrow \langle F_i \rangle$ transfert du contenu de F_{i-1} en F_i

$F_i \Longrightarrow \langle J \rangle$ transfert de l'adresse de F_i dans J.

2) Au début d'un corps de procédure :

F_{i-1}, c'est-à-dire la mémoire F du dernier bloc activé, est variable :

$$\langle\langle J \rangle\rangle \Rightarrow \langle F_i \rangle$$

$$F_i \Rightarrow \langle J \rangle$$

Il résulte de 1) et 2) que tout encombrement dû aux appels récursifs d'une procédure est uniquement reflèté par la mémoire F qui correspond au corps de cette procédure (et ensuite par les blocs intérieurs à ce corps de procédure), et non par la mémoire F du bloc dans lequel cette procédure a été activée.

Si un corps de procédure est quitté par une instruction GO TO, le contenu de J doit être rectifié avant d'entrer de nouveau dans une procédure quelconque.

3) Au début d'un appel de procédure qui ne fait pas partie d'un paramètre effectif :

$$F_i \Rightarrow \langle J \rangle$$

3.3.2.5.2. - LES OPERATIONS DE TRANSFERT

La première action du programme interprétatif est le transfert des adresses des sous-programmes pour les paramètres formels décrits ci-dessus. La dernière partie du P.A. fait d'abord le test indiqué en 3.3.2.2.D), puis il

appelle TR(R) qui libère tous les registres qui contiennent des résultats intermédiaires obtenus avant l'appel de procédure. Puis on effectue l'entrée dans le corps de procédure.

A l'entrée dans le corps de procédure, l'opération 5.1.2) est effectuée, puis la partie TRANSPORTEUR du programme interprétatif est activée. Elle transfère les vecteurs V et $\overline{V}$, correspondant aux variables locales au corps de la procédure et aux résultats intermédiaires respectivement, et finalement elle transfère les adresses des sous-programmes pour les paramètres formels et l'adresse de retour une par une en zône adresses variables. L'encombrement qui en résulte est transféré dans la mémoire F du corps de procédure.

La dernière action est le test des paramètres formels : pour tout paramètre appelé par valeur, le sous-programme correspondant est activé et le résultat obtenu est transféré dans la mémoire correspondante. Si le paramètre est appelé par nom, cette mémoire est chargée par l'adresse du sous-programme qui donne la valeur (ou l'adresse) du paramètre effectif correspondant.

A la sortie d'une procédure tous les transferts sont effectués dans le sens inverse.

Voir aussi exemple 3.2.5. et annexe 2.

IV - LE CALCULATEUR ELECTRONIQUE CAB 500 ET SON COMPILATEUR ALGOL 60

4-1 - LE CALCULATEUR

La CAB 500 est une machine binaire à tambour magnétique qui comporte 128 pistes à 128 mots par piste. Chaque mot est directement adressable, le temps moyen d'accès est de 10 ms.

Il y a aussi 16 registres (mémoires rapides) Chacun peut servir comme accumulateur. Ces registres sont en principe indépendants ; certains ont des fonctions particulières : registre d'index, registres de résultats des opérations logiques, de décalages, etc...

La structure du mot est la suivante :
32 bits (dont un pour le signe) + 1 bit de parité pour les mémoires magnétiques et pour les registres. Les bits 32 (signe) et 33 (parité) sont couplés, ils peuvent indiquer le débordement d'un registre :

mot positif et non-débordé	transfert registre
mot négatif et non-débordé	tambour possible
registre positif et débordé	transfert registre
registre négatif et débordé	mémoire tambour impossible

La virgule flottante utilise 6 bits + 1 (signe) pour l'exposant et le reste du mot (24 bits + 1 + 1) pour la mantisse. L'exposant est donc limité à $2^{\pm 63}$.

La structure des instructions :

Les instructions de base suivantes sont câblées : addition et soustraction (en virgule fixe), décalages, intersection, disjonction(addition sans report), ruptures de séquence, stop. Toutes les autres opérations sont microprogrammées.

Les instructions sont en général à deux adresses dont l'une porte sur un registre qui reçoit aussi le résultat de l'opération ; l'autre adresse pouvant être soit un registre, soit une mémoire magnétique. Les instructions de décalage et logiques font exception. Le résultat de ces opérations se trouve dans l'un de deux registres particuliers.

Les instructions de rupture de séquence n'ont qu'une adresse.

Certaines instructions admettent une forme conditionnelle.

On ne dispose que de deux facilités concernant l'adressage des mémoires : il y a un registre d'index, mais pas d'adressage indirect, à l'exception de l'instruction de rupture de séquence, qui peut aussi conserver le compteur ordinal dans un des registres.

L'entrée et la sortie se fait par flexowriter à la vitesse de $\sim$ 5 caractères/seconde. Un lecteur optique de ruban perforé ($\sim$50 caractères/Seconde) et une perforatrice rapide peuvent complèter l'équipement. Les temps d'exécution des opérations sont les suivants : 0,3 ms pour les opérations de base en virgule fixe, environ 60 ms pour les opérations arithmétiques microprogrammées + temps d'accès.

4.2. - PARTICULARITES DU COMPILATEUR ALGOL 60 SUR CAB 500

4.2.1. - TRADUCTION SEQUENTIELLE PAR PASSAGE UNIQUE

Le dispositif d'entrée ainsi que le temps d'accès important et le nombre réduit de mémoires disponibles après l'implantation des microprogrammes et autres fonctions standard sur le tambour demandent une optimisation rigoureuse des opérations sur les données, c'est-à-dire sur le programme en ALGOL à compiler. Dans la traduction séquentielle en un seul passage, la bande perforée avec le programme constitue en quelque sorte une mémoire externe supplémentaire. La chaîne des symboles d'ALGOL n'est pas entrée sur le tambour, un seul élément de la chaîne est accessible à tout moment. La codification est ainsi réduite à un minimum - elle reste nécessaire uniquement pour l'empilage des états internes du compilateur et pour la conservation des déclarations.

L'analyse de la syntaxe est toujours basée sur les deux opérations de substitution et de remplacement, mais elles sont appliquées simultanément.

4.2.2. - MEMOIRES NON-HOMOGENES L'ADRESSAGE DES VARIABLES

Le petit nombre de mémoires rapides disponibles doit être exploité au maximum afin de réduire le nombre des opérations de transfert dans le sens tambour-registres. Le schéma d'adressage dans le cas d'une instruction à deux

adresses fait intervenir plusieurs types de quantités.

Toute variable (simple) ou constante possède à tout moment une adresse sur le tambour (T), mais elle peut aussi se trouver dans un des registres (par exemple lorsqu'une valeur lui a été affectée). Dans ce cas, elle possède aussi une adresse ferrites (F). Il y a donc des variables FT et FT. Les résultats intermédiaires ou variables anonymes sont gardés autant que possible dans les registres jusqu'à leur emploi définitif, ils sont transférés sur le tambour uniquement si leur nombre dépasse celui des registres.

Pour les deux opérandes d'une instruction, il y a donc quatre cas possibles : (F,F), (F, T), (T,F) et (T, T).

Les deux premiers cas peuvent être traités tout de suite.

L'image des registres est donnée par le vecteur $\beta[b]$, b varie de 1 à 8. Dans le cas (T,T) le premier opérande est transféré dans le premier registre dont l'adresse est supérieure à b et qui ne contient pas une variable anonyme AF. Si tous les registres sont occupés par des AF, un transfert est effectué préalablement et la variable correspondante devient AT. Le cas (T,T) est alors réduit au cas (F,T)

Dans le cas (T,F) les opérandes peuvent être permutés lorsque l'opération est symétrique sinon le premier opérande est appelé dans un registre ; ce cas est donc réduit

soit à (F, T) , soit à (F, F).

Toute variable non simple devient une variable anonyme AF après son évaluation(ou après le calcul d'adresse qui s'y attache).

4.2.3. - TRANSFERTS DE TYPE

Les variables déclarées integer et les nombres entiers correspondent aux nombres en virgule fixe (convention CAB 500), les variables déclarées real et les constantes contenant une partie décimale ou un facteur de cadrage correspondent aux nombres en virgule flottante (convention CAB 500, décrit en 4.1. ci-dessus).

La convention virgule fixe suppose la virgule toujours à l'extrême droite d'un mot machine - le bit de poids le plus faible représente donc toujours 1 - La convention CAB 500 du nombre fractionnaire, qui suppose la virgule toujours à l'extrême gauche d'un mot n'est pas utilisée dans le compilateur.

Les transferts de type sont déterminés statiquement, c'est-à-dire par le compilateur pendant la compilation. Chaque transfert de type correspond à l'appel d'un microprogramme. L'instruction machine correspondante est incorporée dans le programme généré.

Les nombres entiers désignant une adresse peuvent être appelés directement dans le registre d'index sans transformation préalable.

4.3. - PERFORMANCES DU COMPILATEUR

4.3.1. - L'OCCUPATION DE LA MEMOIRE PENDANT LA COMPILATION

Le compilateur occupe les pistes 42 à 66
71 à 73
et 112 à 117

soit 34 pistes contenant environ 4 300 instructions.

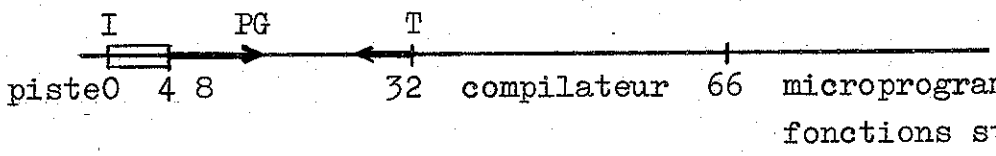
Les autres pistes du tambour, à partir de 67, contiennent en permanence les divers microprogrammes et les fonctions standard.

Les pistes 35 à 41 sont chargées par les tables auxiliaires du compilateur, notamment par la table syntaxique.

Les pistes 33 et 34 sont utilisées comme piles (p.d.l) des états syntaxiques et d'opérandes respectivement. Quatre pistes au début du tambour (de 0 à 3) sont occupées par l'interpréteur des appels de procédures et par des programmes de lancement.

Toutes les pistes restantes reçoivent les instructions générées ou sont occupées par les tables d'identificateurs.

Le programme généré et les tables d'identificateurs peuvent donc occuper 29 pistes au total (3 712 mots).



PG : programme généré

T : table d'identificateurs

I : interpréteur

La longueur maximum du programme généré dépend du nombre d'identificateurs présents dans la liste T. Cette liste est de longueur variable, elle contient à tout moment uniquement les identificateurs dont la déclaration reste valable. Une répartition appropriée des déclarations peut donc minimiser la longueur de T au moment où le programme généré s'approche de la piste 30. Dans ces conditions le programme généré peut être de 24 pistes (3 072 instructions en langage machine), ce qui permet la compilation de programmes d'une certaine complexité.

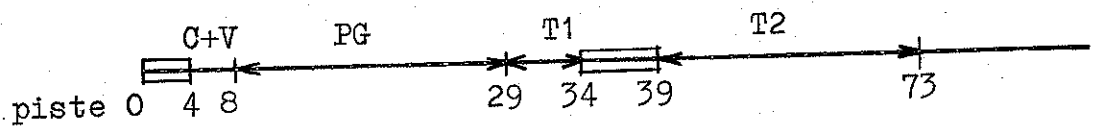
4.3.2. - L'OCCUPATION DE LA MEMOIRE PENDANT L'EXECUTION D'UN PROGRAMME GENERE

Avant l'exécution d'un programme généré les
constantes numériques se trouvent en piste 4
les mémoires réservées pour les identificateurs en piste 5 - 8
et le programme généré en piste 8 (6) - 29

Le programme généré utilise également les tables (piste 35 à 38). Les tableaux et éventuellement les copies

successives provenant d'appels récursifs des procédures peuvent être placés entre le programme généré et la piste 35. Dans ce cas, le compilateur n'est pas altéré et on peut passer à une autre compilation après l'exécution d'un premier programme.

Si le programme est trop long et si l'on a de grands tableaux, ceux-ci peuvent être placés dans les pistes qui contiennent le compilateur (39 - 66 ou 39 - 73).



C+V = constantes et variables

PG = programme généré

T1, T2 = zônes pour les tableaux

4.3.3. - VITESSE DE COMPILATION EFFICACITE DU PROGRAMME GENERE

La vitesse de compilation dépend largement de la structure du programme ; elle varie entre 30 et 60 instructions en langage machine par minute lorsqu'on emploie le lecteur photoélectrique. Une statistique portant sur 30 programmes de 200 à 2 600 instructions (générées) donne une vitesse moyenne de 180 × le temps d'accès par instruction générée, ce qui semble être l'ordre de grandeur pour un compilateur ALGOL 60 de structure simple.

Cette performance est moitié moins grande si l'on se sert de laflexowriter comme organe d'entrée. Il semble qu'un organe d'entrée plus rapide diminuerait fortement le temps de compilation.

Les performances d'un programme généré sont essentiellement fonction de l'adressage des éléments des tableaux et d'emploi des procédures - notamment des procédures récursives.

Le programme pour une intégrale double suivant la méthode de Romberg * par une procédure définie récursivement occupe 409 mots machine, il est généré en 10 minutes environ, son exécution prend 15 minutes.

4.3.4. - DETECTION D'ERREURS

Les erreurs dans un programme soumis au compilateur sont détectées à quatre niveaux différents de la compilation, elles sont distinguées par quatre "libellés erreur" différents. Les erreurs de perforation et de présentation machine des symboles de base et des nombres sont détectées par le programme de lecture.

Parmi les erreurs de déclaration : la non - déclaration, la non - spécification, la spécification d'un paramètre qui n'est pas dans la liste des paramètres formels et le double emploi des étiquettes, sont détectées.

Les erreurs de syntaxe sont détectées par la matrice de transition si c'est une erreur de succession, autrement elles le sont par les programmes de génération.

* annexe au chapitre IV

```
,1   8,32        1065
     8,33        T12EM8,35
     8,34        RCV1          appel transporteur des registres
     8,35        X             adr. de retour du s/progr. paramètre effec.
     8,36        A4EM5,3
     8,37        S4M4,111
     8,38        T4M34,11      sous-programme paramètre effectif
     8,39        A15EV4363
     8,40        RM8,35
     8,41        A9EV8   ----> protection du registre R3 occupé par 1
     8,42        RCV97         appel transporteur des registres, entréecorps
     8,43        RCV2,34       retransfert
     8,44        A5EM0+
     8,45        M5M5,3
     8,46        IOQW49152
     8,47        VI3QR14
     8,48        A5ER14
     8,49        A3ER5
     8,50        T3M5,1        transfert du résultat.
     8,51        AOEVI039
     8,52        RM8,14        fin du corps de procédure.
     8,53        AOEVI039      activation de la déclaration.
     8,54        TOEM5,2
     8,55        A1OEM5,2
     8,56        AOEV639       F1 ⇒ <J>
     8,57        TOEM38,18
     8,58        RCVI,42       appel du P.A.
     8,59        1089
     8,60        TI2EM8,62
     8,61        RCVI          appel transporteur des registres.
     8,62        X
     8,63        A15EV622      adresse du paramètre effectif dans R15.
,1   8,64        RM8,62
     8,65        A9EV0
     8,66        RCV97         appel transporteur des registres, entrée corps
     8,67        RCV2,34       retransfert.
     8,68        F
```

ANNEXE 2

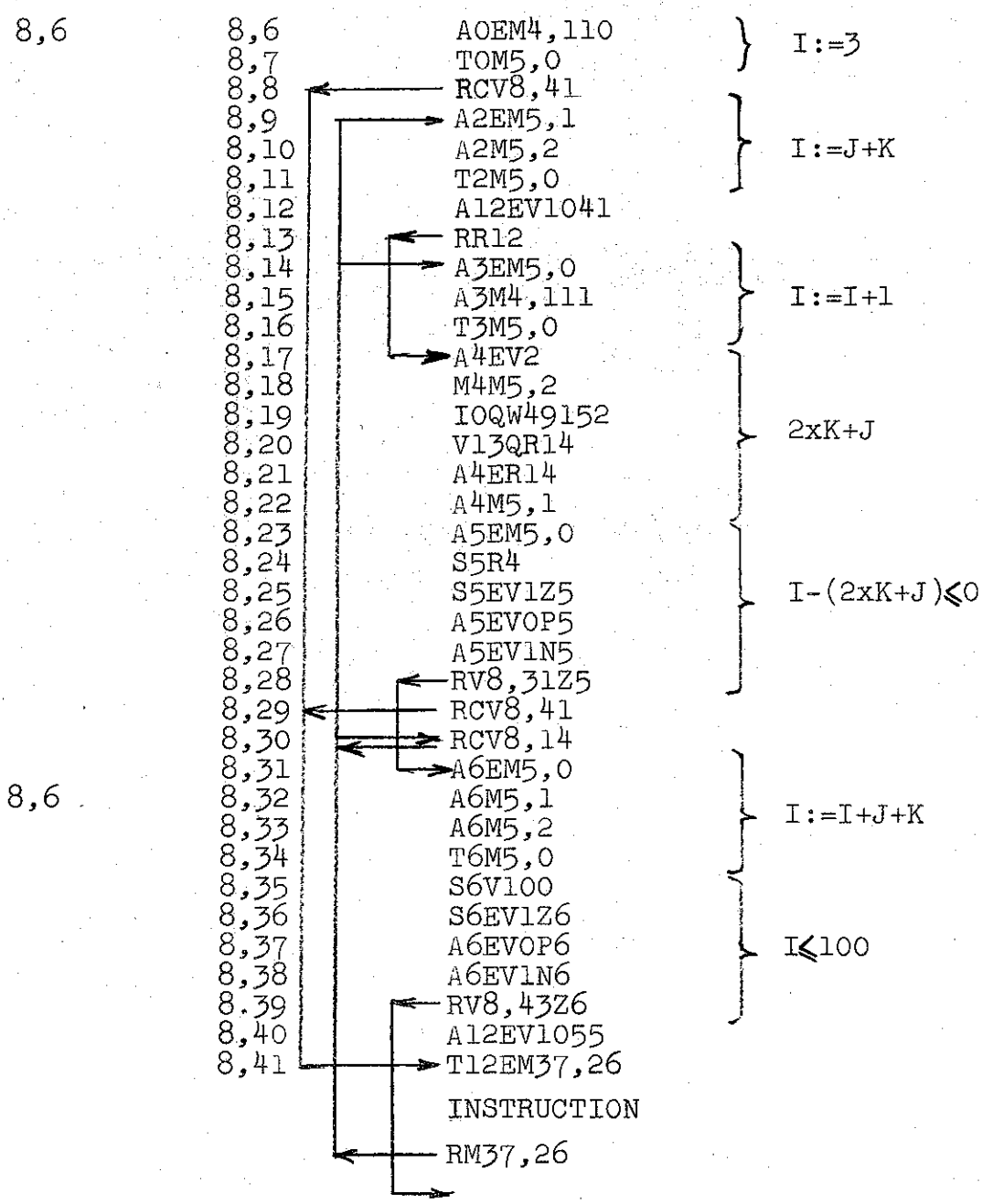
PROCEDE DE GENERATION DE L'EXEMPLE 3.2.5. (Voir table syntaxique)

PILE DES ETATS SYNTAXIQUES	PILE DE OPERANDES	M : CARACTERE LU	SP GENERATION	SP DE CONTROLE
∅	-	'BEGIN'	-	CH(D)
D	-	'INTEGER'	OUBL	ENT(DE,T)
D (DE,T)	-	'PROCEDURE'	-	CH(DE,PR)
D (DE,PR)	-	FACT	DECLA	ENT(L1)
D (DE,PR)L1	-	(	PF	SORTIE
D (DE,PR)	-	N	DECLA	ENT(L1)
D (DE,PR)L1	-	)	ELIMIN	REP
D (DE,PR)	-	)	NUMER	SORTIE
D	-	;	-	ENT(∅)
D ∅	-	'VALUE'	SP	ENT(DEC)
D ∅ DEC	-	N	SPEC	ENT(L1)
D ∅ DEC L1	-	;	-	REP
D ∅ DEC	-	;	-	REP
D ∅	-	'INTEGER'	SP	CH(LFP)|ENT(DEC)
D LFP DEC	-	N	SPEC	ENT(L1)
D LFP DEC L1	-	;	-	REP
D LFP	-	FACT	DCP	ENT(σ)
D LFP σ	-	:=	ST	CH(AST)
D LFP AST	FACT	'IF'	-	ENT(PIF)
D LFP AST PIF	FACT	N	ST	ENT(σ)
D LFP AST PIF σ	FACT|N	=	-	CH(BI,)
D LFP AST PIF (BI,=)	FACT|N	0	STO	ENT(σ)
D LPF AST PIF (BI,=) σ	FACT|N|0	'THEN'	EXU	REP
D LPF AST PIF σ	FACT|N|0	'THEN'	-	REP
D LPF AST PIF	FACT|I1	'THEN'	TEST 1	CH(E1)
D LPF AST E1	FACT	1	STO	ENT(0)
D LPF AST E1 σ	FACT|1	'ELSE'	-	REP
D LPF AST E1	FACT|1	'ELSE'	SAUT 1	CH(E)
D LFP AST E	FACT|1	N	ST	ENT(σ)
D LFP AST E σ	FACT|1|N	X	-	CH(BI,X)
D LFP AST E (BI,X)	FACT|1|N	FACT	ST	ENT(σ)
D LFP AST E (BI,X) σ	FACT|1|N|FACT	(	DIP	CH(σ_2)|ENT(L_2)
D LFP AST E (BI,X) σ_2 L_2	FACT|1|N|FACT	N	ST	ENT(σ)
D LFP AST E (BI,X) σ_2 L_2 σ	FACT|1|N|FACT|N	-	-	CH(BI,-)
D LFP AST E (BI,X) σ_2 L_2 (BI,-)	FACT|1|N|FACT|N	1	STO	ENT(σ)
D LFP AST E (BI,X) σ_2 L_2 (BI,-)0	FACT|1|N|FACT|N|1	)	EXU	REP
D LFP AST E (BI,X) σ_2 L_2 σ	FACT|1|N|FACT|I_1	)	-	REP
D LFP AST E (BI,X) σ_2 L_2	FACT|1|N|FACT|I_1	)	-	REP
D LFP AST E (BI,X) σ_2	FACT|1|N|FACT|I_1	)	PE	CH(σ)
D LFP AST E (BI,X) σ	FACT|1|N|I_1	;	EXU	REP
D LFP AST E σ	FACT|1|I_1	;	-	REP
D LFP AST E	FACT|1|I_1	;	SAUT 2	REP
D LFP AST	FACT|I_1	;	AFFECT	REP
D LFP	-	;	FCP	SORTIE
D	-	;	-	ENT(∅)
D ∅	-	FACT		CH(σ)
D σ	-	(	ST|DIP	CH(σ_2)|ENT (L_2)
D σ_2 L_2	FACT	3	STO	ENT(σ)
D σ_2 L_2 σ	FACT|3	)	-	REP
D σ_2 L_2	FACT|3	)	-	REP
D σ_2	FACT|3	)	PE	SORTIE
D	-	'END'	FEBL	SORTIE
∅	-	-	-	-

Annexe 2

Ex.3.31 'FOR'I:=3,J+K'STEP'1'UNTIL'J+2xK,I+J+K'WHILE'I 100'DO'

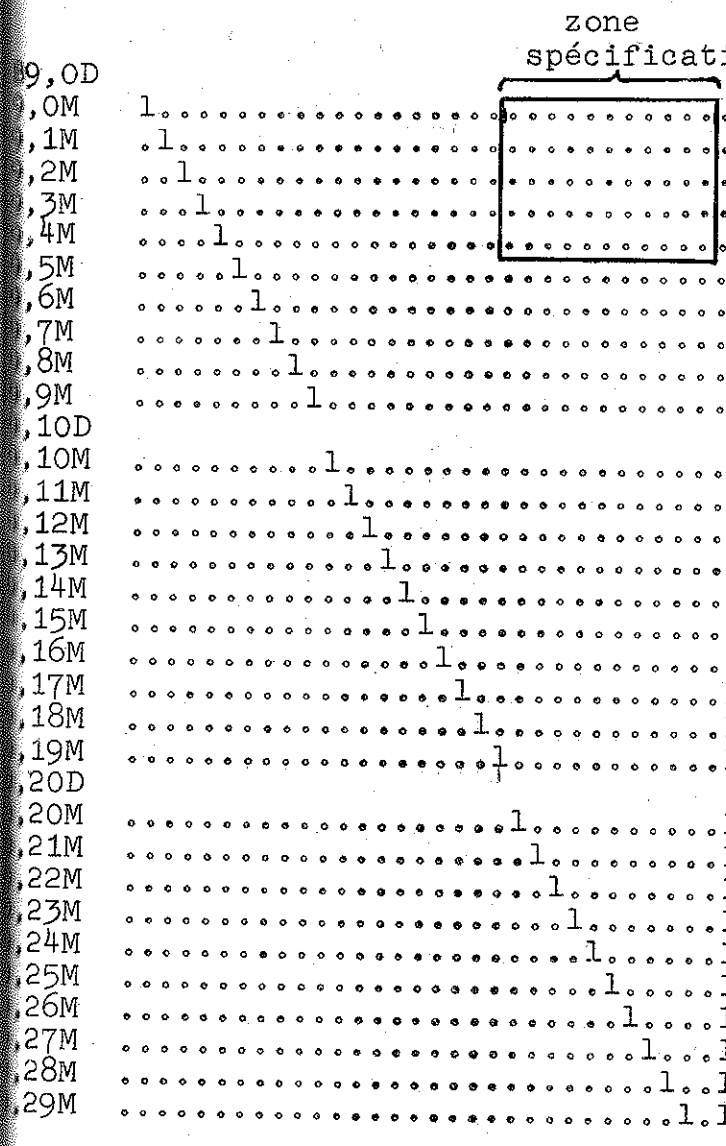
*



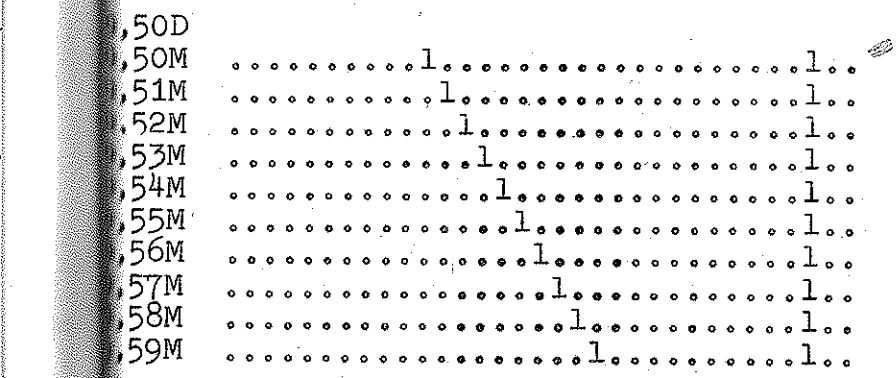
```
'DEBUT' 'REEL' 'PROCEDURE' ROMBERG (FCT, A, B, ORD)Δ
                                   'VALEUR' A, B, ORDΔ
        'REEL' A, BΔ'ENTIER' ORD Δ 'REEL' 'PROCEDURE' FCT Δ
        'DEBUT' 'REEL' 'TABLEAU' T.|1:ORD+1|.Δ 'REEL' L, U, M Δ
                                    'ENTIER' F, H, J, N Δ
                L:=B-AΔT.|1|. : = (FCT(A) + FCT(B) )/2.0 Δ N:=1Δ
                'POUR' H:=1 'PAS' 1 'JUSQUA' ORD 'FAIRE'
                'DEBUT' U:=0Δ M:=L/(2xN) Δ
                        'POUR' J:=1 'PAS' 2 'JUSQUA' 2xN-1 'FAIRE'
                          U:=U + FCT(A + J x M)Δ
                          T.|H +1|.:= (U/N + T.|H|.)/2.0ΔF:=1Δ
                        'POUR' J:=H 'PAS' -1  'JUSQUA' 1 'FAIRE'
                        'DEBUT' F:=4xFΔT.|J|.:=T.|J+1|.
                                        +(T.|J+1|.-T.|J|.)/(F-1)  'FIN'Δ
                          N:=2xN
                'FIN'Δ
                ROMBERG:= T.|1|. x L
        'FIN'Δ
        'REEL' 'PROCEDURE' F(X)Δ 'VALEUR' X Δ'REEL' X Δ
        'DEBUT' 'REEL' 'PROCEDURE' G(Y)Δ 'VALEUR' Y Δ'REEL' Y Δ
                                    G:=60.0xX*2xY
                        F:= ROMBERG(G, 0, 1-X, 3)
        'FIN' Δ 'REEL' J Δ
        J:=ROMBERG (F,0, 1.0, 3) Δ
        IMPRESSION (J, 5, 10)

'FIN'
```

ANNEXE 1

Codification des états syntaxiques



9,0D			
,0M	1 ... 11	(DE,XX)	déclaration
,1M	.1 ... 11	(BI,X)	opération binaire
,2M	..1 ... 11	(UN,X)	opération unaire
,3M	...1 ... 11	(DE)	début
,4M	....1 ... 11	(PIF)	proposition IF
,5M	....1 ... 11		
,6M	.....1 ... 11	(I.F)	instruction FOR
,7M	.....1 ... 11	(DEP)	déclaration PROCEDURE
,8M	.....1 ... 11	(LFP)	liste de paramètres formels
,9M	.....1 ... 11	(Ø)	état zéro
,10D			
,10M	.....1 ... 11	(O)	opérande
,11M	.....1 ... 11	(L1)	liste d'identificateurs
,12M	.....1 ... 11	(P)	parenthèse
,13M	.....1 ... 11	(E1) }	expression
,14M	.....1 ... 11	(E) }	
,15M	.....1 ... 11	(01)	variable indiciée, tableau
,16M	.....1 ... 11	(ASG)	affectation
,17M	.....1 ... 11	(IST)	instruction SI
,18M	.....1 ... 11	(CST)	instruction conditionnelle
,19M	.....1 ... 11	(L2)	liste d'expressions
,20D			
,20M	.....1 ... 11	(02)	indicateur de fonction
,21M	.....1 ... 11	(OI)	opérande indéfini
,22M	.....1 ... 11		
,23M	.....1 ... 11		
,24M	.....1 ... 11		
,25M	.....1 ... 11		
,26M	.....1 ... 11		
,27M	.....1 ... 11	(EFL 3)	élément d'une liste FOR
,28M	.....1 ... 11	(EFL 2)	
,29M	.....1 ... 11	(EFL 1)	
40M	1 ... 1 ..	(ESL)	élément d'une liste d'aiguillage



(L) étiquette

(E1D) }
(ED) } expression de désignation

(IGT) instruction ALLER A

TABLES DES SYMBOLES

DELIMITEURS

	Adresse	Codification
BEGIN	36,29M	5100
END	36,30M	4950
FOR	36,31M	5473
STEP	36,32M	5332
UNTIL	36,33M	5439
DO	36,34M	5352
REAL	36,35M	16782100
INTEGER	36,36M	8393492
IF	36,37M	5046
THEN	36,38M	5060
ARRAY	36,39M	18879686
PROCEDURE	36,40M	1053877
VALUE	36,41M	268440494
ELSE	36,42M	5071
GOTO	36,43M	5161
BOOLEEN	36,44M	4199188
FALSE	36,45M	5136
TRUE	36,46M	529424
COMMENT	36,47M	5114
SWITCH	36,48M	134222900
INTER	36,49M	15209288
UNION	36,50M	9966408
IMPLI	36,51M	5247816
NOT	36,52M	19927880
IDENT	36,53M	529224
LABEL	36,54M	529326
STRING	36,55M	
WHILE	36,56M	5459

Remarque : Les symboles α correspondant qui servent à la reconnaissance se trouvent dans le même ordre dans les mémoires de 36,0 à 36,28

CODIFICATION DE LA TABLE SYNTAXIQUE

5100	BEGIN	
B5100M	---1--1-11-------11----------	
B5101M	----------------------------	
B5102M	---------1------------------	
B5103M	----------------------------	
D5104M	55,95	CH(D)
B5105M	---1--1----------11----------	
B5106M	----------------------------	
D5107M	55,97	ENT(D)
B5108M	--------1-------------------	
B5109M	----------------------------	
D5110M	112,0	DCP|ENT(D)

5046	IF	
B5046M	---11-1-11--1-1-1-11--------1----	
B5047M	1----1--------111----------------	
B5048M	---1--1--1--------1--------------	
B5049M	----------------------------	
D5050M	59,104	ENT(PIF)/CH(PIF)
B5051M	----1-------1-1-1--1--------1----	
B5052M	---------------1----------------	
D5053M	59,100	ENT(PIF^1)
B5054M	----------------------------	
B5055M	1----1--------1-1--------------	
D5056M	59,102	ENT(PIF^2)
B5057M	--------1-------------------	
D5059M	112,0	DCP|ENT(PIF)

5060	THEN	
B5060M	----1-----1---1----------------	
B5061M	----------------------------	
B5062M	----------1-------------------	
B5063M	----------------------------	
D5064M	44,100	REP
B5065M	----1-----------------------	
B5066M	----------------------------	
D5067M	57,0	TEST1|CH(.)
B5068M	---------------1--------------	
B5069M	----------------------------	
D5070M	58,16	SAUT2|REP

4982	;

```
B4982M      1--1--111111--1-111---------1---
B4983M      1---------1---111---------------
B4984M      1-------------------------------
B4985M      --------------------------------
D4986M 44,64                                  NUMER|S
B4987M      ---------111-----------------1---
B4988M      ---------------1----------------
D4989M 44,100                                 REP
B4990M      ----------------1---------------
B4991M      --------------------------------
D4992M 54,0                                   AFFECT|REP
B4993M      ---1----------------------------
B4994M      --------------------------------
D4995M 47,120                                 ENT(Ø)
B4996M      --------------1-----------------
B4997M      --------------------------------
D4998M 58,16                                  SAUT2|REP
B4999M      -----------------1--------------
B5000M      --------------------------------
D5001M 58,72                                  SAUT1|REP
B5002M      ------------------1-------------
B5003M      --------------1-----------------
D5004M 58,108                                 SAUT2|REP
B5005M      --------------------------------
B5006M      ----------------1---------------
D5007M 43,94                                  SAUT|REP
B5008M      --------------------------------
B5009M      ----------1---------------------
D5010M 56,100                                 PLD|DES1|REP
B5011M      --------------------------------
B5012M      1-------------------------------
D5013M 61,114                                 FS|REP
B5014M      -------1------------------------
B5015M      --------------------------------
D5016M 44,87                                  S
B5017M      --------1-----------------------
B5018M      --------------------------------
D5019M 112,78                                 FCP|S
B5020M      ------1------------------------
B5021M      --------------------------------
D5022M 63,91                                  P14|REP
```

SYMBOLES HAUT

	Adresse	Codification
+	4666M	27792200
-	4667M	28316488
,	4668M	5214
x	4669M	29889352
=	4670M	23587896
	4671M	-929946694
/	4672M	30413640
Δ	4673M	4982

SYMBOLES BAS

•	4674M	100667150
o (en indice) = ∗	4675M	31462216
ERREUR	4676M	4922
)	4677M	5269
ERREUR	4678M	4922
ERREUR	4679M	4922
(	4680M	5249
<	4681M	22025032
≠	4682M	24646472
ERREUR	4683M	4922
ERREUR	4684M	4922
<	4685M	20976456
ERREUR	4686M	4922
ERREUR	4687M	4922
ERREUR	4688M	4922
ERREUR	4689M	4922
⩾	4690M	23073608
:=	4691M	4924
:	4692M	5144
π	4693M	-939440124
*	4694M	32510792
[	4695M	5180
]	4696M	5197
>	4697M	24122184
	4698M	-741294136

4950	END	
B4950M	---1--1-111---1-111----------1---	
B4951M	----------1---1-1---------------	
B4952M	---------11------------------1---	
B4953M	--------------------------------	
D4954M	44,100	REP
B4955M	---1----------------------------	
B4956M	--------------------------------	
D4957M	47,88	FEBL|S
B4958M	----------------1---------------	
B4959M	--------------------------------	
D4960M	54,0	AFFECT|REP
B4961M	--------------1----------------	
B4962M	--------------------------------	
D4963M	58,16	SAUT2|REP
B4964M	----------------1---------------	
B4965M	--------------------------------	
D4966M	58,72	$SAUT^1$|REP
B4967M	------------------1-------------	
B4968M	--------------1----------------	
D4969M	58,108	$SAUT^2$|REP
B4970M	--------------------------------	
B4971M	----------------1---------------	
D4972M	43,94	SAUT|REP
B4973M	--------------------------------	
B4974M	----------1---------------------	
D4975M	56,100	PDL|DES1|REP
B4976M	---------1----------------------	
B4977M	--------------------------------	
D4978M	112,78	FCP|S
B4979M	------1-----------------------	
B4980M	--------------------------------	
D4981M	63,91	P14|REP

4936	ω :tout opérateur arithmétique,de relation et Booléen	
B4936M	-1--1-----1-111-1--1-1------1----	
B4937M	----------------1---------------	
B4938M	----------1---------------------	
B4939M	--------------------------------	
D4940M	113,44	CH(BI,ω)
B4941M	----1-------111-1--1--------1----	
B4942M	----------------1---------------	
D4943M	44,119	ENT(UN,ω)
B4944M	-1--------------------------------	
B4945M	--------------------------------	
D4946M	56,36	PRE|EXU|ENT(UN,ω)
B4947M	--------------------1----------	
B4948M	--------------------------------	
D4949M	113,49	

Codification

HAUT

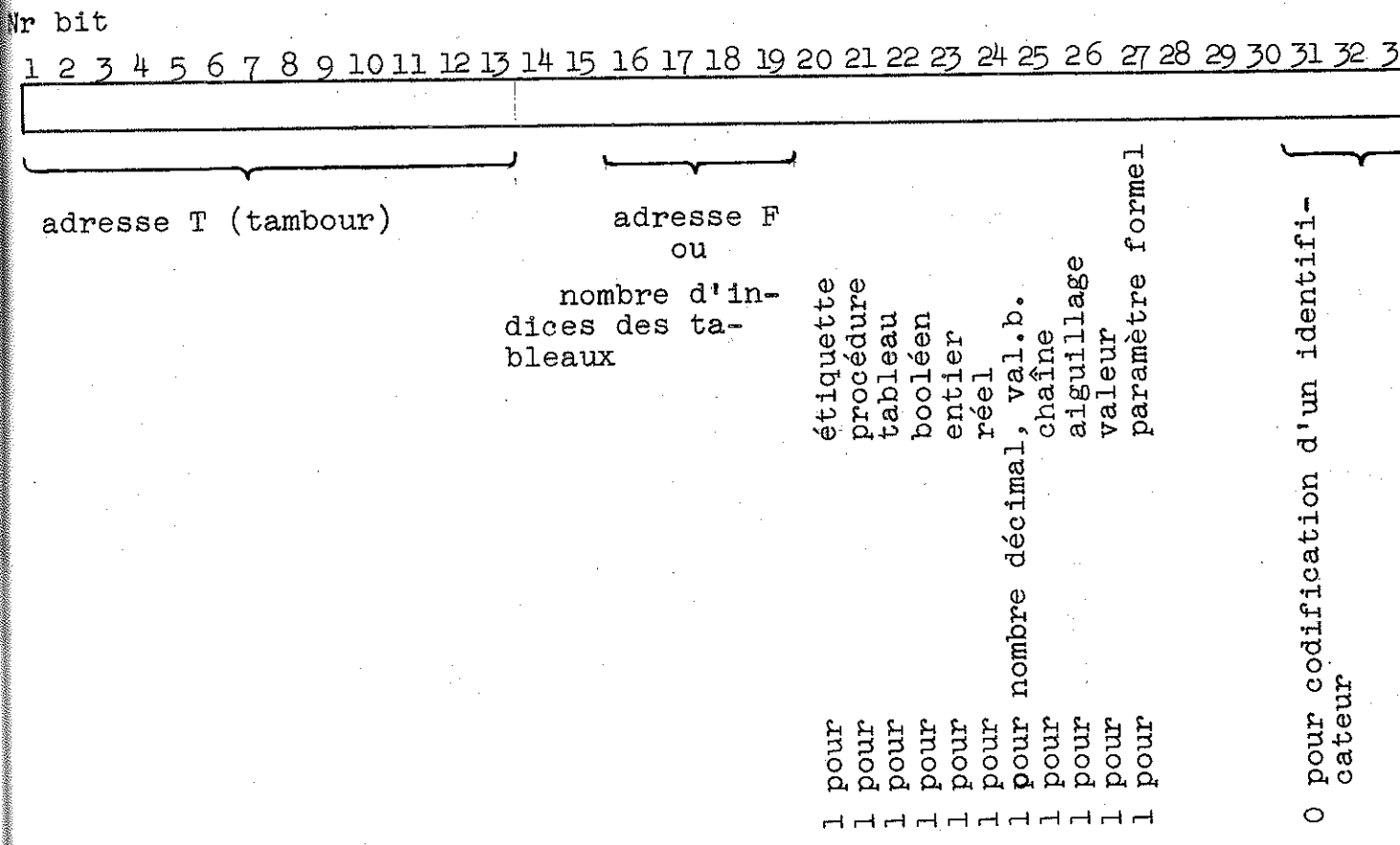
+ B4666M	---1--1-11--1------1-1-11--------
- B4667M	---1--1-11--1------11-11--------
x B4669M	---1--1-11--1------1--111--------
= B4670M	---1--1-11--1-----1-11-1--------
/ B4672M	---1--1-11--1------1-111--------

BAS

• B4674M	-111----1111----------11------
÷ B4675M	---1--1-11--1-------1111-------

≤ B4681M	---1--1-11--1------1-1-1--------
≠ B4682M	---1--1-11--1------1111-1--------
< B4685M	---1--1-11--1---------1-1--------
≥ B4690M	---1--1-11--1--------11-1--------
↑ * B4694M	---1--1-11--1------11111--------
> B4697M	---1--1-11--1------111-1--------

Codification des identificateurs



5180	C
B5180M	----------11---------1----------
B5181M	----------1----------------------
B5182M	----------------------------------
B5183M	----------1----------------------
D5184M	42,0
B5185M	----------1----------------------
B5186M	----------------------------------
D5187M	114,60
B5188M	----------1----------------------
B5189M	----------------------------------
D5190M	114,64
B5191M	---------------------1----------
B5192M	----------------------------------
D5193M	113,40

ST|CH(S)

CH(L2)

TFT|ENT(L2)

CX|REP

5071	ELSE
B5071M	----------1--11-11----------------
B5072M	----------1-11-11----------------
B5073M	----------1----------------------
B5074M	---------------1-----------------
D5075M	44,100
B5076M	---------------1-----------------
B5077M	----------------------------------
D5078M	57,52
B5079M	----------------1-----------------
B5080M	---------------1-----------------
D5081M	58,72
B5082M	---------------1-----------------
B5083M	----------------------------------
D5084M	58,16
B5085M	----------------1-----------------
B5086M	----------------------------------
D5087M	54,0
B5088M	----------------------------------
B5089M	----------------1-----------------
D5090M	43,94
B5091M	----------------------------------
B5092M	----------1----------------------
D5093M	56,100

REP

SAUT1|CH(E)

SAUT[1]|CH(IS)/CH(ED)

SAUT2|REP

AFFECT|REP

SAUT|REP

PPL|DES1|REP

5038	Spécificateur : LABEL, VALUE
B5038M	--------11----------------------
B5039M	----------------------------------
B5040M	--------11----------------------
B5041M	----------------------------------
D5042M	73,105

SP|ENT(DEC)

5214	, VIRGULE

```
B5214M       l---l-----ll--lll--lll------ll---
B5215M       l---------l---ll----------------
B5216M       ----------l--------------------
B5217M       ------------------------------
D5218M 44,87                                  S
B5219M       ----------l--------l------------
B5220M       ---------------l----------------
D5221M 44,100                                 REP
B5222M       -------------l-----------------
B5223M       ------------------------------
D5224M 58,16                                  SAUT2|REP
B5225M       ------------------------------
B5226M       l-----------------------------
D5227M 61,94                                  DS|ENT(ESL)
B5228M       ------------------------------
B5229M       ----------l-------------------
D5230M 56,100                                 PDL|DES1|REP
B5231M       ------------------------------
B5232M       -------------l----------------
D5233M 58,108                                 SAUT²|REP
B5234M       l-----------------------------
B5235M       ------------------------------
B5236M 115,4                                  BS|ENT(L2)
B5237M       --------------l---------------
B5238M       ------------------------------
D5239M 117,50                                 INDEX|ENT(L2)
B5240M       ---------------------l----------
B5241M       ------------------------------
D5242M 73,62                                  PSP
B5243M       --------------------l----------
B5244M       ------------------------------
D5245M 71,30                                  PE|ENT(L2)
B5246M       ----l----------l-----------ll---
B5247M       ------------------------------
D5248M 44,38                                    SUITE
B5026M       ----------------------------l---
B5027M       ------------------------------
D5028M 62,25
B5029M       --------------l---------------
B5030M       ------------------------------
D5031M 54,0                                   AFFECT|REP
B5032M       ----l-------------------------
B5033M       ------------------------------
D5034M 62,60
B5035M       ---------------------------l----
B5036M       ------------------------------
D5037M 63,12
```

5269	)

```
B5269M      1---------111-1----111----------
B5270M      -----1----1---1----------------
B5271M      ----------1--------1------------
B5272M      ---------------------------------
D5273M 44,100                                  REP
B5274M      ------------1--------------------
B5275M      ---------------------------------
D5276M 59,116                                  CH(σ)
B5277M      --------------1-----------------
B5278M      ---------------------------------
D5279M 58,16                                   SAUT2|REP
B5280M      ---------------------------------
B5281M      ----------1----------------------
D5282M 56,100                                  PDL|DES1|REP
B5283       ---------------------------------
B5284M      --------------1-----------------
D5285M 58,108                                  SAUT²|REP
B5286M      ----------1----------------------
B5287M      ---------------------------------
D5288M 66,12                                   ELIMIN|REP
B5289π      1--------------------------------
B5290M      ---------------------------------
D5291M 44,64                                   NUMER|S
B5292M      --------------------1----------
B5293M      ---------------------------------
D5294M 71,30                                   PE|CH(σ)
B5295M      ---------------------------------
B5296M      -----1---------------------------
D5297M 56,23                                   CH(EP)
B5298M      --------------------1----------
B5299M      ---------------------------------
D5300M 73,62                                   DSP|ENT(σ)
```

5161	GOTO

```
B5161M      ---1--1-11-------11-------------
B5162M      ---------------------------------
B5163M      ---------1-------------------------
B5164M      ---------------------------------
D5165M 43,70                                   CH(GTS)
B5166M      ---1--1----------11-------------
B5167M      ---------------------------------
D5168M 43,66                                   ENT(GTS)
B5169M      --------1-------------------------
B5170M      ---------------------------------
D5171M 112,0                                   DCP|ENT(GTS)
```

4895	IDENTIFICATEURS	
B4895M	11111-1111--11111111--------1----	
B4896M	1----1------1111----------	
B4897M	1----------	
B4898M	----------	
D4899M	44,40	DECLA|ENT(L1)
B4900M	-11----------	
B4901M	----------	
D4902M	47,0	ST|PRE|ENT(σ)
B4903M	--------1----------	
B4904M	----------	
D4905M	112,0	DCP|ENT(σ)
B4906M	----1------11111----------1----	
B4907M	----------1----------	
D4908M	55,104	ST|ENT(σ)
B4909M	---1--1--1-------11----------	
B4910M	----------	
D4911M	55,120	ENT(σ)/CH(σ)
B4912M	----------	
B4913M	1----1------11-1----------	
D4914M	43,86	ENT(L)
B4915M	-------1----------	
B4916M	----------	
D4917M	42,6	SPEC|ENT(L1)
B4918M	----------1----------	
B4919M	----------	
D4920M	73,28	DET

5136	VALEURS BOOLEENNES	
B5136M	-11-1------111-1--1----------	
B5137M	----------	
B5138M	-11-1------111-1--1----------	
B5139M	----------	
D5140M	60,32	STARE|ENT(σ)

5122	NOMBRES PURS	
B5122M	-1111-1--1--111-1111--------1----	
B5123M	1----1------1111----------	
B5124M	-11-1------111-1--1--------1----	
B5125M	----------1----------	
D5126M	60,0	STORE|ENT(σ)
B5127M	----------	
B5128M	1----1------11-1----------	
D5129M	43,86	ENT(L)
B5130M	---1--1--1-------11----------	
B5131M	----------	
D5132M	55,120	ENT(σ)

5144	:

```
B5144M      1--1--1-1-1---1--111----------
B5145M      ------------------------------
B5146M      ----------1--------1----------
B5147M      ------------------------------
D5148M 44,100                                  REP
B5149M      --------------1-----------------
B5150M      ------------------------------
B5151M 58,16                                   SAUT2|REP
B5152M      ---1--1-1--------11------------
B5153M      ------------------------------
D5154M 61,28                                   ET|ENT(Ø)
B5155M      1-----------------------------
B5156M      ------------------------------
D5157M 115,0                                   BI|ENT(L2)
```

4924	:=

```
B4924M      ----------11---1---------------
B4925M      ------------------------------
B4926M      ----------1-------------------
B4927M      ------------------------------
D4928M 56,16                                   ST|CH(ASG)
B4929M      ----------1-------------------
B4930M      ------------------------------
D4931M 61,62                                   DDS|CH(ESL)
B4932M      --------------1---------------
B4933M      ------------------------------
D4934M 47,112                                  CH(ASG)
```

5197	]

```
B5197M      1---------1---11---1-----------
B5198M      --------------1---------------
B5199M      ------------------------------
B5200M      --------------1---------------
D5201M 60,100                                  ISW|S
B5202M      ----------1--------1-----------
B5203M      ------------------------------
D5204M 44,100                                  REP
B5205M      1-----------------------------
B5206M      ------------------------------
D5207M 115,4                                   BS|REMA
B5208M      --------------1---------------
B5212M      ------------------------------
D5213M 117,50                                  INDEX|CH(σ)
```

41,84	STEP
B41,84M	----------1-1-1-1----------11---
B41,85M	----------1----------
B41,86M	----------1---
B41,87M	----------
D41,88M	62,75
B41,89M	----------1----
B41,90M	----------
D41,91M	62,82
B41,92M	----------1-1----------
B41,93M	----------
D41,94M	44,100
B41,95M	----------1----------
B41,96M	----------
D41,97M	54,0
B41.98M	----------1----------
B41,99M	----------
D41,100M	58,16
B41,101M	----------
B41,102M	----------1----------
D41,103M	43,94

42,83	WHILE
B42,83M	----------1-1-1-1----------1---
B42,84M	----------
B42,85M	----------1---
B42,86M	----------
D42,87M	62,58
B42,88M	----------1----------
B42,89M	----------
D42,90M	58,16
B42,91M	----------1-1----------
B42,92M	----------
D42,93M	44,100
B42,94M	----------1----------
B42,95M	----------
D42;96M	54,0

42,108 REP

```
B42,108M  ----------------1---------111---
B42,109M  ----------1-----1----------------
B42,110M  ------------------------------1---
B42,111M  ---------------------------------
D42,112M 44,100
B42,113M  ------------------------------1---
B42,114M  ---------------------------------
D42,115M 62,46
B42,116M  ------------------------------1---
B42,117M  ---------------------------------
D42,118M 62,64
B42,119M  ----------------1----------------
B42,120M  ---------------------------------
D42,121M 58,72
B42,122M  ---------------------------------
B42,123M  ----------------1----------------
D42,124M 43,94
B42,125M  ---------------------------------
B42,126M  ----------1----------------------
D42,127M 56,100
```

REP

SAUT[1]|REP

SAUT|REP

PDL|DES1|REP

42,97 FOR

```
B42,97M   ---1--1-11-------11--------------
B42,98M   ---------------------------------
B42,99M   ---1--1----------11--------------
B42,100M  ---------------------------------
D42,101M 62,05
B42,102M  ---------1------------------------
B42,103M  ---------------------------------
D42,104M 62,16
B42,105M  --------1-------------------------
B42,106M  ---------------------------------
D42,107M 112,0
```

DCP|ENT(IF)

5249	(
B5249M	-11-1-----11111-1--1-1------1----
B5250M	1-----11-1----------
B5251M	---1------111-1--1--------1----
B5252M	----------
D5253M	56,49
B5254M	-11----------
B5255M	----------
D5256M	56,36
B5257M	---------1----------
B5258M	----------
D5259M	73,0
B5260M	----------1----------
B5261M	----------
D5262M	113,32
B5263M	----------
B5264M	1----------11-1----------
D5265M	56,28
B5266M	----------1----------
B5267M	----------
D5268M	61,60

ENT(P)

PRE|ENT(P)

D.IP|CH(σ_2)|ENT(L2)

PF|S

ENT(PD)

ST|DIP|CH(σ_2)|ENT(L2)

4884	TYPE
B4884M	---1----11----------
B4885M	----------
B4886M	--------11----------
B4887M	----------
D4888M	73,105
B4889M	---1----------
B4890M	----------
D4891M	47,60

SP|CH(LFP)|ENT(DEC)

OUBL|ENT(DE,TE)

5172	SWITCH
B5172M	---1----11----------
B5173M	----------
B5174M	---1----------
B5175M	----------
D5176M	47,60
B5177M	--------11----------
B5178M	----------
D5179M	73,105

OUBL|ENT(DE,SW)

SP|ENT(DEC)

5114	COMMENT
B5114M	---1----11----------
B5115M	----------
B5116M	---1----11----------
B5117M	----------
D5118M	43,1
B5119M	--------1----------
B5120M	----------
D5121M	113,53

ELIMINATION

5301	PROCEDURE
B5301M	1--1---111----------
B5302M	----------
B5303M	--------11----------
B5304M	----------
D5305M	73,105
B5306M	---1----------
B5307M	----------
D5308M	47,60
B5309M	1----------
B5310M	----------
D5311M	66,0
B5312M	-------1----------
B5313M	----------
D5314M	66,6

SP|ENT(DEC)

OUBL|ENT(DE,TE)

CH(DE,PR)

SPEC1/REP

42,63	UNTIL
B42,63M	----------1-1-1-1----------1----
B42,64M	----------1----------
B42,65M	----------
B42,66M	----------1----------
D42,67M 43,94	
B42,68M	----------1----
B42,69M	----------
D42,70M 62,122	
B42,71M	----------1----------
B42,72M	----------
D42,73M 58,16	
B42,74M	----------1----------
B42,75M	----------
D42,76M 62,101	
B42,77M	----------1----------
B42,78M	----------
D42,79M 44,100	
B42,80M	----------1----------
B42,81M	----------
D42,82M 54,0	

SAUT|REP

SAUT2|REP

REP

AFFECT|REP

41,104	DO
B41,104M	----1-1---1---1-1----------11---
B41,105M	----------
B41,106M	----------1---
B41,107M	----------
D41,108M 62,55	
B41,109M	------1----------
B41,110M	----------
D41,111M 63,74	
B41,112M	----1----------
B41,113M	----------
D41,114M 63,61	
B41,115M	----------1----------
B41,116M	----------
D41,117M 44,100	
B41,118M	----------1----------
B41,119M	----------
D41,120M 58,16	
B41,121M	----------1----------
B41,122M	----------
D41,123M 54,0	
B41,124M	----------1----
B41,125M	----------
D41,126M 63,12	

REP

SAUT2|REP

AFFECT|REP

5318	ARRAY
B5318M	1--1---111----------
B5319M	----------
B5320M	1----------
B5321M	----------
D5322M	66,0
B5323M	--------11----------
B5324M	----------
D5325M	73,105
B5326M	---1----------
B5327M	----------
D5328M	47,60
B5329M	------1----------
B5330M	----------
D5331M	66,6

CH(DE,TE,TA)

SP|CH(LPF)ENT(DEC)

OUBL|ENT(DE,TA)

SPEC1|REP

TABLE DES SOUS-PROGRAMMES DE TRAITEMENT DE LISTES ET DE GENERATION

A) OPERATIONS SUR LES LISTES

ST	Cherche le nom d'un identificateur dans $\mathcal{N}$ et place la codification correspondante dans la pile des opérandes
STO	Cherche un nombre (ou rentre un nombre) dans la liste C et place la codification correspondante dans la pile des opérandes
STA	Même action que STO mais pour les valeurs booléennes
DECLA	Rentre un couple (nom, codification) dans $\mathcal{N}$
NUMER	Donne une adresse fixe aux identificateurs d'une liste de déclarations
PF	Prépare la codification d'un paramètre formel
ELIMIN	Elimine le commentaire entre paramètres formels ou effectifs
SP	Prépare la codification d'une spécification donnée
SPEC	Spécification d'un paramètre formel se trouvant dans $\mathcal{N}$
STI	Prépare le calcul de la fonction d'adresse d'une variable indicée
PDL	Cherche uen étiquette (resp. rentre une étiquette) dans $\mathcal{E}$ et place la codification correspondante dans la pile des opérandes
DES	Prolonge la chaîne de reprise pour une étiquette
ET	Fournit "la valeur" d'une étiquette
OUBL	Ouverture niveau de bloc (de nomenclature) dans $\mathcal{N}$ et $\mathcal{E}$.
FEBL	"Efface" un niveau de nomenclature dans $\mathcal{N}$

B) SOUS-PROGRAMMES DE GENERATION

PRE	Test de la priorité de deux opérateurs binaires
EXU	Suite de PRE : génère une opération binaire avec $\alpha[a-2]$ et $\alpha[a-1]$ et remplace $\alpha[a-2]$ par la codification d'un résultat intermédiaire
TEST	Génère le saut conditionnel d'une proposition IF
SAUT1	Génère le saut incond. à la fin du calcul de l'expression qui suit le 'THEN' et substitue l'adresse du saut de TEST
SAUT2	Substitue l'adresse du saut incond. généré par SAUT1
SAUT1, SAUT2	Programmes correspondants dans le cas d'une instruction conditionnelle
AFFECT	Génération d'un transfert registre ⇒ tambour ou tambour ⇒ tambour (par l'intermédiaire du registre 0) : $\alpha[a] \Rightarrow \alpha[a-1]$
BI	Génère les instructions d'une expression qui est la borne inférieure dans une déclaration de tableau

BS	Calcul d'une borne supérieure et des paramètres de la fonction d'attribution d'adresse
INDICE	Calcul d'une expression en indice
ADR	Génération du calcul d'adresse d'une variable indicée
PE	Génération d'un sous-programme pour un paramètre effectif (3.3.2.2.)
DIP	Début instruction procédure : appel dans R10 de l'adresse du corps de procédure, gestion de la mémoire et appel du P.A.
IP	Activation des programmes interprétatifs, entrée et sortie corps de procédure
SAUT	Saut inconditionnel d'une instruction GO TO
DDS	"En-tête" d'une liste de S.P. d'une déclaration d'aiguillage
DS	S.P. pour un élément d'une liste d'aiguillage, l'adresse corresp. entre rentrée dans l'"en-tête"
FS	Sortie d'un calcul d'un élément d'aiguillage
ISW	Génération de l'appel du S.P. d'aiguillage après la compilation de l'exp. en indice d'un indicateur d'aiguillage.
OUBL	Génère le S.P. ouvert de la gestion automatique de la mémoire (zone adresses variables)
FEBL	Génère les sauts correspondants aux étiquettes locales au bloc
DCP	Début de corps de procédure : génère un saut, puis la codification de l'"en-tête" de procédure, appel OUBL
FCP	Fin de corps de procédure :rappel de l'adresse du début de corps dans R0 et réservation du nombre de mémoires correspondant au nombre de paramètres formels, appel FEBL.

1.3.2.5.

```
BEGIN''INTEGER''PROCEDURE' FACT(N)Δ 'VALUE' NΔ'INTEGER' NΔ
                    FACT:= 'IF' N=0 'THEN' 1 'ELSE' NxFACT(N-1)Δ

          FACT(3)

END'
```

*****

```
1          8,1     A15EM38,18
           8,2     AOEMO+
           8,3     TOEM4,127
           8,4     AOEV639
           8,5     TOM38,18
           8,6     RV8,53
           8,7     X
           8,8     X
           8,9     813695619
           8,10    1
           8,11    644
           8,12    4355
           8,13    4363
           8,14    X
           8,15    A15EM38,18
           8,16    AOEMO+
           8,17    TOEM5,4
           8,18    AOEV644
           8,19    TOM38,18
           8,20    RM37,20
           8,21    A2EM5,3
           8,22    S2EV1Y2
           8,23    A2EV1Z2
           8,24    A2EVON2
           8,25    RV8,28Z2
           8,26    A3EM4,111
           8,27    RV8,50
           8,28    A1OEM5,2
           8,29    AOEV644
           8,30    TOEM38,18
           8,31    RCV1,42
```

«J» ⇒ ⟨F_1⟩

F_1 ⇒ ⟨J⟩

repère zone adresses variables (variable)
adresse du s/progr. du paramètre effectif (variable)
codification et adresse du par. formel (fixe)
nombre des " (fixe)
dernière adresse occupée
emplacement résultats intermédiaires (fixe)
adresse de retour (variable)

«J» ⇒ F_2

F_2 ⇒ ⟨J⟩

⟶ appel du transporteur
⟶ début du corps de procédure

F_2 ⇒ ⟨J⟩

appel du P.A.

B I B L I O G R A P H I E

[1] Revised Report on the Algorithmic Language ALGOL 60
Peter Naur, éditeur.
International Federation for Information Processing 1962

[2] Y. Bar-Hillel, M. Perles, E. Shamir :
On formal properties of simple phrase structure grammars
Zeitschrift für Phonetik, Sprachwissenschaften und Komm. 1961.

[3] N. Chomsky : On certain formal properties of grammars Information and Contr. 2 (1959), 137 - 167.

[4] K. Culik : Formal Structure of ALGOL and Simplification of its description
Symbolic Languages in Data Processing,
Proceedings of the Symposium - Rome, 1962.
The Internation Computation Center, éditeur.

[5] R. J. Evey : The Theory and Application of Pushdown Store Machines
Rep. Nr NSF 10, the Comp. Lab. of Harvard University 1963

[6] S. Gorn : Detection of Generative Ambiguities in Contex free Mechanical Languages
Journal of the ACM 1963

[7] A.A. Grau : Recursive Processes and ALGOL - Translation
Comm. ACM 4 - 1961

[8] M. Nivat, L. Nolin : Sur un procédé de définition de la syntaxe d'ALGOL.
Publication de l'Institut Blaise Pascal 1963

[9] M. Paul : A General Processor for Certain Formal Languages
(même Ref. que [4]).

[10] M. Paul : Zur Struktur formaler Sprachen
Thèse 1962, Faculté des Sciences, Université de Mainz.

[11] K. Samelson, F.L. Bauer : Sequentielle Formelübersetzung
Elektr. Rechenanlagen 1 - 1959

[12] J. Jensen, P. Naur : An Implimentation of ALGOL 60 Procedures
T.I.B. 1/1/1961

VU,

Grenoble, le

Le Président de la Thèse

VU,

Grenoble, le

Le Doyen de la Faculté des Sciences

VU et permis d'imprimer,

Le Recteur de l'Académie de Grenoble