



**HAL**  
open science

## Précédence, analyse syntaxique et langages de programmation

Alain Colmerauer

► **To cite this version:**

Alain Colmerauer. Précédence, analyse syntaxique et langages de programmation. Autre [cs.OH]. Université Joseph-Fourier - Grenoble I, 1967. Français. NNT : . tel-00008436

**HAL Id: tel-00008436**

**<https://theses.hal.science/tel-00008436>**

Submitted on 10 Feb 2005

**HAL** is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

T H E S E S

présentées à la Faculté des Sciences  
de Grenoble pour obtenir le  
grade de Docteur ès Sciences Appliquées  
par

Alain COLMERAUER

Ingénieur IMAG, Licencié ès Sciences

première thèse

PRECEDENCE, ANALYSE SYNTAXIQUE  
ET LANGAGES DE PROGRAMMATION

deuxième thèse

PROPOSITION DONNEE PAR LA FACULTE

Thèses soutenues le 26 Septembre 1967,  
devant la commission d'examen

MM. VAUQUOIS

GASTINEL

Mme. BERTRANDIAS

MM. BOLLIET

GENUYS

Président

Examineurs



# FACULTE DES SCIENCES

## LISTE DES PROFESSEURS

DOYENS HONORAIRES :

M. MORET

M. WEIL

DOYEN :

M. BONNIER E.

PROFESSEURS TITULAIRES :

MM. NEEL Louis	Chaire de Physique Expérimentale
HEILMANN René	Chaire de Chimie
KRAVTCHEKNO Julien	Chaire de Mécanique Rationnelle
CHABAUTY Claude	Chaire de calcul différentiel et intégral
BENOIT Jean	Chaire de Radioélectricité
CHENE Marcel	Chaire de Chimie Papetière
WEIL Louis	Chaire de Thermodynamique
FELICI Noël	Chaire d'Electrostatique
KUNTZMANN Jean	Chaire de Mathématiques Appliquées
BARBIER Reynold	Chaire de Géologie Appliquée
SANTON Lucien	Chaire de Mécanique des Fluides
OZENDA Paul	Chaire de Botanique
FALLOT Maurice	Chaire de Physique Industrielle
KOSZUL Jean-Louis	Chaire de Mathématiques M.P.C.
GALVANI O.	Mathématiques
MOUSSA André	Chaire de Chimie Nucléaire
TRAYNARD Philippe	Chaire de Chimie Générale

SOUTIF Michel	Chaire de Physique Générale
CRAYA Antoine	Chaire d'Hydrodynamique
REULOS R.	Théorie des Champs
BESSON Jean	Chaire de Chimie
AYANT Yves	Physique Approfondie
GALLISSOT	Mathématiques
Melle LUTZ Elisabeth	Mathématiques
MM. BLAMBERT Maurice	Chaire de Mathématiques
BOUCHEZ Robert	Physique Nucléaire
LLIBOUTRY Louis	Géophysique
MICHEL Robert	Chaire de Minéralogie et Pétrographie
BONNIER Etienne	Chaire d'Electrochimie et d'Electrométallurgie
DESSAUX Georges	Chaire de Physiologie Animale
PILLET E.	Chaire de Physique Industrielle et Electrotechnique
VOCCOZ Jean	Chaire de Physique Nucléaire Théorique
DEBELMAS Jacques	Chaire de Géologie Générale
GERBER R.	Mathématiques
PAUTHENET R.	Electrotechnique
VAUQUOIS B.	Chaire de Calcul Electronique
BARJON R.	Physique Nucléaire
BARBIER Jean-Claude	Chaire de Physique
SILBER R.	Mécanique des Fluides
BUYLE-BODIN Maurice	Chaire d'Electronique
DREYFUS B.	Thermodynamique
KLEIN J.	Mathématiques
VAILLANT F.	Zoologie et Hydrobiologie
ARNOUD Paul	Chaire de Chimie M.P.C.
SENGEL P.	Chaire de Zoologie
BARNOUD F.	Chaire de Biosynthèse de la Cellulose
BRISSONNEAU P.	Physique
GAGNAIRE Didier	Chaire de Chimie Physique

Mme KOFLER L.	Botanique
MM. DEGRANGE Charles	Zoologie
PEBAY-PEROULA J.C.	Physique
RASSAT A.	Chaire de Chimie Systématique

PROFESSEURS SANS CHAIRE :

MM. GIDON P.	Géologie et Minéralogie
GIRAUD P.	Géologie
PERRET R.	Servomécanismes
Mme BARBIER M.J.	Electrochimie
Mme SOUTIF J.	Physique
MM. COHEN J.	Electrotechnique
DEPASSEL R.	Mécanique des Fluides
GASTINEL N.	Mathématiques Appliquées
ANGLES-d'AURIAC P.	Mécanique des Fluides
DUCROS P.	Minéralogie et Cristallographie
GLENAT R.	Chimie
LACAZE A.	Thermodynamique
BARRA J.	Mathématiques Appliquées
COUMES A.	Electronique
PERRIAUX J.	Géologie et Minéralogie
ROBERT A.	Chimie Papetière
BIAREZ J.P.	Mécanique Physique
BONNET G.	Electronique
CAUQUIS G.	Chimie Générale
BONNETAIN L.	Chimie Minérale
DEPOMMIER P.	Etude Nucléaire et Génie Atomique
HACQUES Gérard	Calcul Numérique
POLOUJADOFF M.	Electrotechnique

MAITRES DE CONFERENCES :

MM. DODU J.	Mécanique des Fluides
LANCIA Roland	Physique Automatique
Mme KAHANE J.	Physique
MM. DEPORTES C.	Chimie
Mme BOUCHE L.	Mathématiques
MM. SARROT-RAYNAUD J.	Géologie Propédeutique
Mme BONNIER M.J.	Chimie
MM. KAHANE A.	Physique Générale
DOLIQUE J.M.	Electronique
BRIERE G.	Physique M.P.C.
DESPRE P.	Chimie S.P.C.N.
LAJZEROWICZ J.	Physique M.P.C.
VALENTIN P.	Physique M.P.C.
BERTRANDIAS J.P.	Mathématiques Appliquées T.M.P.
LAURENT P.	Mathématiques Appliquées T.M.P.
CAUBET J.P.	Mathématiques Pures
PAYAN J.J.	Mathématiques
Mme BERTRANDIAS F.	Mathématiques Pures M.P.C.
MM. LONGEQUEUE J.P.	Physique
NIVAT M.	Mathématiques Appliquées
SOHM J.C.	Electrochimie
ZADWORNY F.	Electronique
DURAND F.	Chimie Physique
CARLIER G.	Biologie Végétale
AUBERT G.	Physique M.P.C.
DELPUECH J.J.	Chimie Organique
PFISTER J.C.	Physique C.P.E.M.
CHIBON P.	Biologie Animale
IDELMAN S.	Physiologie Animale
BLOCH D.	Electrotechnique
BRUGEL L.	I.U.T.
SIBILLE R.	I.U.T.

Que Monsieur le Professeur J. Kuntzmann, Directeur de l'Institut de Mathématiques Appliquées, reçoive mes remerciements les plus sincères pour m'avoir donné la possibilité de préparer et de présenter cette thèse.

Je remercie Monsieur le Professeur B. Vauquois, Directeur du Centre d'Etudes pour la Traduction automatique, de m'avoir fait l'honneur de présider le jury, et je remercie Monsieur Le Professeur N. Gastinel, Directeur de l'Institut de Programmation de Grenoble, d'avoir eu la bienveillance de s'intéresser à ce travail. Enfin je ne manquerais pas de remercier Madame Bertrandiaz, Maître de Conférence à l'Université de Grenoble, qui dans le cadre de la deuxième thèse, m'a donné l'occasion d'apprécier un domaine des mathématiques nouveau pour moi.

Je remercie tout spécialement Monsieur L. Bolliet, Maître de Conférence à l'Institut Universitaire de Technologie de Grenoble, de m'avoir accordé sa confiance pour réaliser ce travail au sein de son équipe. Je remercie aussi très vivement Monsieur F. Genuys, Conseiller Scientifique à la Compagnie IBM, d'avoir bien voulu m'aider et me guider dans mes recherches.

Enfin je remercie toutes les personnes du laboratoire, qui par leurs entretiens ou leurs travaux, ont contribué à la réalisation de cette thèse, en particulier Jean-Claude Boussard, dont les critiques et les conseils me furent très utiles durant la rédaction, Philippe Jorrand qui s'est toujours montré un ami attentif et perspicace, Laurent Trilling qui fut un chaleureux compagnon de travail et François Martin dont certains travaux me furent nécessaires.

Je ne voudrais pas non plus oublier dans mes remerciements tous ceux qui ont participé à la réalisation matérielle de cette thèse.



A ma Femme



TABLE DES MATIERES

	Page
INTRODUCTION .....	1
CHAPITRE 1 : Eléments de base .....	3
1.1 Quelques opérations sur les relations binaires .....	4
1.2 Grammaire "context-free" .....	5
1.3 Grammaire "context-free" parenthésée .....	6
1.4 Dérivation .....	9
1.5 Restrictions habituelles sur les grammaires "context-free" .....	13
1.6 Relations $\alpha$ , $\gamma$ , $\delta$ .....	15
1.7 Problème de l'analyse syntaxique .....	16
1.8 Automate à deux piles .....	18
CHAPITRE 2 : Précédence totale .....	22
2.1 Analyseur et relations de précédence totale : définition .....	23
2.2 Une classe simple de relations de précédence totale .....	25
2.3 Analyseur de précédence totale déterministe .....	28
2.4 Existence de relations de précédence totale incompatibles deux à deux .....	30
2.5 Famille optimale de triplets de relations de précédence totale ..	32
2.6 Grammaires et langages de précédence totale .....	35
CHAPITRE 3 : Précédence de gauche à droite .....	39
3.1 Analyseur et relations de précédence de gauche à droite : définition .....	40
3.2 Analyseur de précédence de gauche à droite déterministe .....	42
3.3. Une définition équivalente de relations de précédence de gauche à droite .....	44
3.4 Relation $\sigma$ .....	46
3.5 Plus petites relations de précédence de gauche à droite .....	49
3.6 Une variante plus générale de l'analyseur de précédence de gauche à droite .....	51
3.7 Cas particulier : relations de précédence terminale .....	57
3.8 Cas particulier : relations de précédence totale de gauche à droi- te .....	60
3.9 Langages de précédence totale, de précédence totale de gauche à droite et autres .....	63

	Page
CHAPITRE 4 : Programmation des différents algorithmes .....	68
4.1 Calcul de matrices booléennes représentant des relations de précédence .....	69
4.2 Généralités sur la programmation des différents analyseurs ....	72
4.3 Analyseur de précédence totale déterministe .....	73
4.4 Analyseur de précédence totale non déterministe .....	76
4.5 Analyseur de précédence de gauche à droite .....	80
4.6 Analyseur de précédence terminale .....	83
CHAPITRE 5 : Applications aux langages de programmation .....	85
5.1 Précédence totale et Algol 60 .....	86
5.2 Précédence de gauche à droite et recherche d'erreurs dans un programme Algol .....	97
5.3 Précédence de gauche à droite et Algol de N. Wirth et C.A.R. Hoare .....	109
5.4 Précédence terminale et compilateur de conversation Fortran ...	130
5.5 Précédence terminale et compilateur de conversation pour un sous-ensemble d'Algol .....	134
CONCLUSION .....	139
REFERENCES .....	141

## INTRODUCTION

Les grammaires "context-free" de N. Chomsky [Ch] semblent s'être imposées comme un modèle particulièrement commode pour décrire des langages, aussi bien en linguistique pour les langues naturelles qu'en compilation pour les langages de programmation. Dans ce dernier domaine, le langage Algol 60 [Na] en est l'exemple le plus typique.

Cependant l'utilisation de ces grammaires pour l'analyse automatique de textes se heurte à des difficultés d'ordre pratique : les algorithmes généraux d'analyse existants coûtent beaucoup de temps ou de place à l'ordinateur. Puisque la compilation nous intéresse surtout ici, nous avons étudié des méthodes d'analyse adaptées aux grammaires particulières que l'on y rencontre.

Les méthodes que nous préconisons utilisent des relations dites de "précédence" représentées par des matrices booléennes et calculées une fois pour toutes à partir de la grammaire considérée. Leur rôle est de permettre des regroupements facilitant l'analyse du texte. On peut considérer que la notion de précédence est une généralisation de celle de priorité entre les opérateurs d'une même expression algébrique.

Voici la démarche suivie au cours de ce travail. Nous avons tout d'abord défini un formalisme pratique pour manipuler nos relations binaires. Nous l'utilisons constamment, surtout pour le calcul de matrices booléennes représentant des relations de précédence. Nous nous intéressons aussi à quelques caractéristiques importantes des grammaires "context-free" et nous introduisons le problème de l'analyse syntaxique. Tout ceci fait l'objet du premier chapitre.

dans le deuxième chapitre nous abordons l'étude d'une première méthode

d'analyse très agréable par son efficacité et sa simplicité. Elle nous a permis entre autres de découvrir une sous-classe intéressante des langages "context-free" les langages "de précedence totale".

Les notions d'opérateurs et d'opérandes dans une grammaire apparaissent au troisième chapitre et permettent de compléter les travaux de R. Floyd [Fl.1] qui le premier, a introduit la notion de précedence.

Les quatrième et cinquième chapitres sont consacrés aux applications pratiques : tout d'abord la façon de programmer les différents algorithmes décrits, puis leur utilisation pour des langages de programmation déterminés.

CHAPITRE 1 : Eléments de base

### 1.1 Quelques opérations sur les relations binaires

Nous appellerons relation binaire sur un ensemble  $E$ , tout sous-ensemble  $\rho$  du produit cartésien  $E \times E$ . Par commodité nous noterons souvent  $a\rho b$  au lieu de  $(a,b) \in \rho$  et nous représenterons toujours une relation binaire par une lettre grecque minuscule ou un signe spécial. Le complément de  $\rho$  par rapport à  $E \times E$  sera noté  $\bar{\rho}$  ; on a donc

$$\bar{\rho} = E \times E - \rho$$

DEFINITION Etant donné deux relations binaires  $\rho$  et  $\sigma$  sur le même ensemble  $E$ , le produit  $\rho\sigma$  sera une nouvelle relation binaire sur  $E$  définie ainsi : pour tout  $a, b \in E$

$$a\rho\sigma b \equiv [\text{il existe } c \in E \text{ tel que } a\rho c \text{ et } c\sigma b]$$

Il est facile de vérifier que ce produit est associatif et qu'il est distributif, à droite et à gauche, par rapport à l'union. Du fait de l'associativité, nous pouvons donc définir sans ambiguïté l'élévation à une puissance entière non négative d'une relation binaire  $\rho$  sur  $E$ .

$$\rho^i = \rho\rho^{i-1} \text{ avec } a\rho^0 b \equiv [a=b] \quad \text{pour tout } a, b \in E$$

DEFINITION Etant donné une relation binaire  $\rho$  sur  $E$ , sa fermeture transitive  $\rho^+$  et sa fermeture transitive reflexive  $\rho^*$  seront deux nouvelles relations binaires sur  $E$  définies ainsi :

$$\rho^+ = \bigcup_{i=1}^{\infty} \rho^i \quad \rho^* = \rho^0 \cup \rho^+$$

Il découle immédiatement que  $\rho^+ = \rho\rho^* = \rho^*\rho$ . Dans le cas où  $E$  est un ensemble fini de  $n$  éléments, pour tout couple  $a, b \in E$  tel que  $a\rho^k b$  avec  $k \geq n$  il existe forcément  $\ell < n$  tel que  $a\rho^\ell b$ . De ce fait  $\bigcup_{i=1}^{\infty} \rho^i = \bigcup_{i=1}^{n-1} \rho^i$  et donc  $\rho^+ = \bigcup_{i=1}^{n-1} \rho^i$ .

De plus, comme nous le verrons au paragraphe 4.1, il est alors possible de transposer toutes les opérations sur des relations binaires de  $E$  en des opérations sur des matrices booléennes, de dimension  $n \times n$ , les représentant.

## 1.2 Grammaire "context-free"

Considérons un ensemble fini  $W$  appelé vocabulaire dont les éléments sont appelés symboles. Une chaîne  $x$  sur  $W$  sera une suite finie de symboles  $A_1, A_2, \dots, A_n \in W$ . Par commodité nous supprimerons les virgules séparant les éléments de la suite et nous noterons  $x = A_1 A_2 \dots A_n$ . La longueur de la chaîne  $x$  sera l'entier  $n$  et sera notée  $|x|$ . Par  $W^*$  nous désignerons l'ensemble de toutes les chaînes que l'on peut construire sur  $W$  y compris la chaîne vide de longueur nulle que nous noterons  $\Lambda$ . Etant donné deux chaînes  $x, y \in W^*$  avec  $x = A_1 A_2 \dots A_i$  et  $y = B_1 B_2 \dots B_j$  nous poserons  $xy = A_1 A_2 \dots A_i B_1 B_2 \dots B_j$ . Bien entendu  $x\Lambda = \Lambda x = x$  quelque soit  $x \in W^*$ .

DEFINITION Tout sous-ensemble de  $W^*$  sera appelé langage.

DEFINITION Une grammaire "context-free"  $G$  sera définie par

- un vocabulaire  $V_T$  dit terminal,
- un vocabulaire  $V_N$  dit non terminal tel que  $V_T \cap V_N = \emptyset$ ,
- un symbole distingué de  $V_N$  appelé axiome et noté  $S$ ,
- une relation binaire finie sur  $(V_T \cup V_N)^*$  notée  $\rightarrow$  dont les éléments seront appelés règles et telle que pour tout  $x, y \in (V_T \cup V_N)^*$ ,  $x \rightarrow y$  entraîne  $x \in V_N$ .

Cette définition est très voisine de celle donnée par Chomsky en [Ch]. Sur  $(V_T \cup V_N)^*$  nous définirons une deuxième relation binaire, notée  $\Rightarrow$ , qui sera une extension de la relation  $\rightarrow$  :

DEFINITION Pour tout couple de chaînes  $x, y \in (V_T \cup V_N)^*$  :

$x \Rightarrow y \equiv$  [il existe  $Z \in V_N$ ,  $u, v, z \in (V_T \cup V_N)^*$  tels que  $x = uZv$ ,  $y = uzv$ ,  $Z \rightarrow z$ ]

Il faut remarquer que les éléments  $Z, z, u, v$  ne sont pas toujours déterminés d'une façon unique ; voici un exemple :  $AB \Rightarrow AUB$  avec  $A \rightarrow AU$  et  $B \rightarrow UB$ .

DEFINITION Le langage défini par la grammaire  $G$  est le sous-ensemble  $L(G)$  de  $V_T^*$  ainsi défini :

$$L(G) = \{t \in V_T^* \mid S \Rightarrow^* t\}$$

Un tel langage est dit "context-free".

En convenant d'appeler phrase toute chaîne  $s \in (V_T \cup V_N)^*$  telle que  $S \Rightarrow^* s$  et de qualifier de terminale toute chaîne  $t \in V_T^*$ , ce langage  $L(G)$  est donc l'ensemble des phrases terminales de  $G$ .

Signalons une propriété bien connue de la relation  $\Rightarrow$ , propriété que nous utiliserons souvent par la suite :

LEMME 1.2.1 Si  $x_1 x_2 \Rightarrow^n y$  avec  $x_1, x_2, y \in (V_T \cup V_N)^*$  et  $n > 0$  alors il existe  $y_1, y_2 \in (V_T \cup V_N)^*$  et des entiers,  $i, j > 0$  tels que  $y = y_1 y_2$ ,  $x_1 \Rightarrow^i y_1$ ,  $x_2 \Rightarrow^j y_2$ ,  $i + j = n$ .

DEMONSTRATION Le théorème est vrai pour  $n=0$ . Supposons le vrai pour  $n$  et démontrons qu'il est vrai pour  $n+1$ . Si  $x_1 x_2 \Rightarrow^{n+1} y$  alors il existe  $z \in (V_T \cup V_N)^*$  tel que  $x_1 x_2 \Rightarrow^n z \Rightarrow y$ . Le théorème étant supposé vrai pour  $n$  il existe donc  $z_1, z_2$  et  $k, l$  tels que  $z = z_1 z_2$ ,  $x_1 \Rightarrow^k z_1$ ,  $x_2 \Rightarrow^l z_2$ ,  $k + l = n$ . Soit  $U \rightarrow u$  la règle qui permet de transformer  $z$  en  $y$ . L'élément  $U$  est soit contenu dans  $z_1$  soit dans  $z_2$ . Supposons par exemple qu'il soit contenu dans  $z_1$ . Il existe donc  $v, w$  tels que  $z_1 = v U w \Rightarrow v u w$ . En posant  $y_1 = v u w$ ,  $y_2 = z_2$ ,  $i = k + 1$ ,  $j = l$  on a donc  $y = y_1 y_2$ ,  $x_1 \Rightarrow^i y_1$ ,  $x_2 \Rightarrow^j y_2$ ,  $i + j = n + 1$ .

### 1.3 Grammaire "context-free" parenthésée

DEFINITION A toute grammaire "context-free"  $G$ , on peut associer une grammaire "context-free"  $G'$  dite grammaire parenthésée : soit  $V_T'$  et  $V_N'$  respectivement les vocabulaires terminal et non terminal de  $G'$ . A chaque symbole  $U \in V_N$  nous ferons correspondre deux nouveaux symboles notés par exemple  $[et]$  et nous désignerons par  $V_U$  l'ensemble des symboles de la forme  $[et]$  et par  $V_U^U$  l'ensemble des symboles de la forme  $]$ . Moyennant ces conventions  $G'$  sera définie ainsi :

- $V'_T = V_T UV [UV]$ ,
- $V'_N = V_N$
- l'axiome de  $G'$  est le même que celui de  $G$ ,
- à chaque règle  $U \rightarrow u$  de  $G$  correspond biunivoquement la règle  $U \rightarrow \begin{matrix} [u] \\ U \end{matrix}$  de  $G'$ .

Les grammaires parenthésées possèdent de nombreuses propriétés.

Tout d'abord il faut remarquer que pour tout couple de chaînes  $x, y$  telles que  $x \Rightarrow y$  dans  $G'$ , les chaînes  $u, v$  et la règle  $Z \rightarrow \begin{matrix} [z] \\ Z \end{matrix}$  telles que  $x = uZv$  et  $y = u \begin{matrix} [z] \\ Z \end{matrix} v$  sont déterminées d'une façon unique.

Une deuxième propriété fondamentale est la suivante :

LEMME 1.3.1 Si dans une grammaire parenthésée  $G'$  on a  $s \Rightarrow^n \begin{matrix} x[u]y \\ X \ Y \end{matrix}$  avec  $X, Y \in V_N$ ,  $s, u \in (V_T UV_N)^*$  et  $x, y \in (V_T UV_N UV [UV])^*$  alors il existe  $U \in V_N$  tel que  $U = X = Y$  et tel que  $s \Rightarrow^{n-1} xUy$ .

DEMONSTRATION Le théorème est vrai pour  $n=1$ . Supposons le vrai pour  $n$  et démontrons qu'il est vrai pour  $n+1$ . Si  $s \Rightarrow^{n+1} \begin{matrix} x[u]y \\ X \ Y \end{matrix}$  alors il existe  $z \in (V_T UV_N UV [UV])^*$  tel que  $s \Rightarrow^n z \Rightarrow \begin{matrix} x[u]y \\ X \ Y \end{matrix}$ . Trois cas se présentent alors :

- (1)  $z = xUy \Rightarrow \begin{matrix} x[u]y \\ X \ Y \end{matrix}$ . On a donc  $U = X = Y$  et  $s \Rightarrow^n xUy$ .
- (2)  $z = r \begin{matrix} [u]y \\ X \ Y \end{matrix} \Rightarrow \begin{matrix} x[u]y \\ X \ Y \end{matrix}$  avec  $r \in (V_T UV_N UV [UV])^*$  et  $r \Rightarrow x$ .

Le théorème étant vrai pour  $n$  il existe donc  $U \in V_N$  tel que  $U = X = Y$  et  $s \Rightarrow^{n-1} rUy$ . Du fait que  $r \Rightarrow x$  on a  $s \Rightarrow^n xUy$ .

- (3)  $z = x \begin{matrix} [u]t \\ X \ Y \end{matrix} \Rightarrow \begin{matrix} x[u]y \\ X \ Y \end{matrix}$  avec  $t \in (V_T UV_N UV [UV])^*$  et  $t \Rightarrow y$ .

Même démonstration que pour le cas (2).

En appliquant  $j$  fois ce lemme on déduit :

COROLLAIRE 1.3.2 Dans une grammaire parenthésée, si  $x \Rightarrow^i z$  et  $y \Rightarrow^j z$  avec  $x \in (V_T \cup V_N)^*$  et  $y, z \in (V_T \cup V_N \cup V[V])^*$  alors  $i \geq j$  et  $x \Rightarrow^{i-j} y$ .

En utilisant ce corollaire nous pouvons maintenant démontrer cette dernière propriété.

LEMME 1.3.3. Si dans une grammaire parenthésée on considère deux suites  $x_0, x_1, \dots, x_i \in (V_T \cup V_N \cup V[V])^*$  et  $y_0, y_1, \dots, y_j \in (V_T \cup V_N \cup V[V])^*$  telles que

- (a)  $x_0 \Rightarrow x_1 \Rightarrow \dots \Rightarrow x_i$                       (b)  $y_0 \Rightarrow y_1 \Rightarrow \dots \Rightarrow y_j$   
 (c)  $x_0, y_0 \in (V_T \cup V_N)^*$                       (d)  $x_i = y_j$

alors

- (1)  $i=j$  et  $x_0 = y_0$ ,  
 (2) si l'on désigne par  $\rho_1$  et  $\rho_2$  les sous-ensembles de règles intervenant respectivement dans les relations (a) et (b) alors  $\rho_1 = \rho_2$ .

DEMONSTRATION Du corollaire 1.3.2 on déduit que  $x_0 \Rightarrow^{i-j} y_0$ ,  $j \geq i$ ,  $y_0 \Rightarrow^{j-i} x_0$  d'où  $i=j$  et  $x_0 = y_0$ . Démontrons le deuxième point par récurrence sur  $i$ . Si  $i=0$  le théorème est vrai car  $\rho_1 = \rho_2 = \emptyset$ . Supposons le théorème vrai pour  $i$  et démontrons qu'il est vrai pour  $i+1$  : on a

$$x_0 \Rightarrow x_1 \Rightarrow \dots \Rightarrow x_{i+1}, y_0 \Rightarrow y_1 \Rightarrow \dots \Rightarrow y_{i+1} \text{ avec } x_0 = y_0 \text{ et } x_{i+1} = y_{i+1}$$

Si  $x_i = y_i$  le théorème est démontré. Si  $x_i \neq y_i$  il existe donc deux règles  $U \rightarrow \begin{bmatrix} u \\ U \end{bmatrix}$ ,  $V \rightarrow \begin{bmatrix} v \\ V \end{bmatrix}$  et des chaînes  $r_1, r_2, r_3$  telles que

$$x_{i+1} = y_{i+1} = r_1 \begin{bmatrix} u \\ U \end{bmatrix} r_2 \begin{bmatrix} v \\ V \end{bmatrix} r_3$$

$$x_i = r_1 U r_2 \begin{bmatrix} v \\ V \end{bmatrix} r_3$$

$$y_i = r_1 \begin{bmatrix} u \\ U \end{bmatrix} r_2 V r_3$$

(le fait d'intervertir  $x_i$  avec  $y_i$  ne changerait rien à la suite). Posons

$z_{i-1} = r_1 U r_2 V r_3$ . D'après le corollaire 1.3.2 il existe donc une suite de chaînes  $z_0, z_1, \dots, z_{i-1}$  telle que

$$x_0 = y_0 = z_0 \Rightarrow z_1 \Rightarrow \dots \Rightarrow z_{i-1}$$

Soit  $\rho$  le sous-ensemble de règles intervenant dans ces relations.

Soit d'autre part,  $\rho_1'$  et  $\rho_2'$  les sous-ensembles de règles intervenant respectivement dans les relations

$$x_0 \Rightarrow x_1 \Rightarrow \dots \Rightarrow x_i \text{ et } y_0 \Rightarrow y_1 \Rightarrow \dots \Rightarrow y_i$$

On a  $\rho_1 = \rho_1' U \{U \rightarrow u\}$  et  $\rho_2 = \rho_2' U \{V \rightarrow v\}$ . Le théorème étant supposé vrai pour  $i$ , on a  $\rho_1' = \rho U \{V \rightarrow v\}$  et  $\rho_2' = \rho U \{U \rightarrow u\}$ .

De ce fait  $\rho_1 = \rho_2 = \rho U \{U \rightarrow u\} U \{V \rightarrow v\}$ .

#### 1.4 Dérivation

Etant donné une grammaire  $G$  de vocabulaire  $V_T \cup V_N$  nous allons définir la notion de dérivation. Pour ceci nous aurons d'abord besoin de la notion de position dans une chaîne.

DEFINITION Considérons une chaîne  $x = A_1 A_2 \dots A_n$  avec  $A_1, A_2, \dots, A_n \in V_T \cup V_N$ . De la suite  $A_1, A_2, \dots, A_n$  extrayons la sous-suite  $x_N$  des éléments non terminaux. Nous dirons que l'élément non terminal  $A_p$  de  $x$  est en position  $k$  dans  $x$  si son rang est  $k$  dans la sous-suite  $x_N$ .

DEFINITION Une dérivation de longueur  $n$ ,  $n$  étant un entier positif ou nul, sera définie par :

- une suite  $x_0, x_1, \dots, x_n \in (V_T \cup V_N)^*$  telle que  $x_0 \Rightarrow x_1 \Rightarrow \dots \Rightarrow x_n$ ,
- dans le cas où  $n \neq 0$ , une suite d'entiers  $i_1, i_2, \dots, i_n$  telle que pour tout  $j \in \{1, 2, \dots, n\}$  la transformation de  $x_{j-1}$  en  $x_j$  se passe en appliquant la règle au symbole en position  $i_j$  de  $x_{j-1}$ .

Elle sera notée  $(x_0 ; i_1, x_1 ; i_2, x_2 ; \dots ; i_n, x_n)$  et nous dirons que  $x_0$  est son origine et  $x_n$  son extrémité.

En faisant intervenir la grammaire parenthésée  $G'$  associée à  $G$ , définissons la notion de structure.

DEFINITION Etant donnée une dérivation  $\Delta = (x_0 ; i_1, x_1 ; \dots ; i_n, x_n)$  dans  $G$ , on peut lui faire correspondre une et une seule dérivation associée dans la grammaire parenthésée  $G'$ , définie de la façon suivante :

- (1) elle est de la forme  $(y_0 ; i_1, y_1 ; \dots ; i_n, y_n)$ ,
- (2)  $x_0 = y_0$ ,
- (3) pour tout  $k \in (1, 2, \dots, n)$  la chaîne  $x_k$  est égale à la chaîne  $y_k$  dans laquelle on aurait remplacé tous les symboles appartenant à  $V \setminus \{V_k\}$  par la chaîne vide  $\Lambda$ .

La chaîne  $y_n$  est dite structure engendrée par la dérivation  $\Delta$ . Nous appellerons structure d'une phrase de  $G$ , toute structure engendrée par une dérivation d'origine  $S$  et ayant cette phrase pour extrémité.

En d'autres termes on peut dire que toute phrase  $s$  de  $G'$  qui ne diffère d'une phrase  $t$  de  $G$  que par des symboles appartenant à  $V \setminus \{V_k\}$  est une structure de  $t$ .

Introduisons une relation d'équivalence entre les dérivations de  $G$ .

DEFINITION Deux dérivations  $\Delta_1$  et  $\Delta_2$  de  $G$  seront dites équivalentes si et seulement si elles engendrent la même structure. On notera  $\Delta_1 \equiv \Delta_2$ .

Du lemme 1.3.3 il découle immédiatement :

LEMME 1.4.1 Deux dérivations équivalentes

- (1) ont même longueur, même origine, même extrémité,
- (2) font intervenir le même sous-ensemble de règles.

Pour démontrer une deuxième propriété essentielle des dérivations nous aurons besoin du lemme suivant qui ne concerne que les grammaires parenthésées.

LEMME 1.4.2 Pour toute application  $f$  qui à chaque chaîne  $u \in (V_T \cup V_N \cup V_{[UV]})^*$ , contenant au moins un symbole non terminal, fait correspondre la position  $f(u)$  d'un symbole non terminal de  $u$  et pour tout couple de chaînes  $x \in (V_T \cup V_N \cup V_{[UV]})^*$  et  $t \in (V_T \cup V_{[UV]})^*$ , tels que  $x \Rightarrow^n t$ , il existe une et une seule dérivation de la forme

$$(x = x_0; f(x_0), x_1; f(x_1), x_2; \dots; f(x_{n-1}), x_n = t)$$

DEMONSTRATION Remarquons tout d'abord que si une telle dérivation existe elle ne peut être qu'unique et ceci du fait que nous raisonnons dans une grammaire parenthésée. Démontrons son existence par récurrence sur  $n$ . Si  $n=0$  cette dérivation existe. Supposons qu'elle existe pour  $n$  et démontrons qu'il en existe une pour  $n+1$ . Si  $x \Rightarrow^{n+1} t$ , soit  $U \in V_N$  le symbole en position  $f(x)$  dans  $x$ .

Posons  $x = yUz$ . En appliquant deux fois le lemme 1.2.1, on déduit qu'il existe des chaînes terminales  $y', u', z'$  telles que  $t = y'u'z'$  et telles que  $y \Rightarrow^i y'$ ,  $U \Rightarrow^j u'$ ,  $z \Rightarrow^k z'$ ,  $i+j+k=n+1$ . Soit  $u$  la chaîne telle que  $U \Rightarrow u \Rightarrow^{j-1} u'$ . Nous avons donc la dérivation  $(x; f(x), yuz)$  et la relation  $yuz \Rightarrow^n t$ . Par supposition, il existe une dérivation de la forme  $(yuz = x_1; f(x_1), x_2; \dots; f(x_n), x_{n+1} = t)$ , d'où la dérivation  $(x; f(x), yuz = x_1; f(x_1), x_2; \dots; f(x_n), x_{n+1} = t)$ .

De ce lemme et de la définition de deux dérivations équivalentes on déduit une propriété concernant toutes les grammaires ;

COROLLAIRE 1.4.3 Pour toute application  $f$  qui à chaque chaîne  $u \in (V_T \cup V_N)^*$  contenant au moins un symbole non terminal fait correspondre la position  $f(u)$  d'un symbole non terminal de  $u$  et pour toute dérivation  $\Delta_1$  dont l'extrémité est terminale, il existe une et une seule dérivation  $\Delta_2$  de la forme

$$\Delta_2 = (y_0; f(x_0), x_1; \dots; f(x_{n-1}), x_n)$$

telle que  $\Delta_1 \equiv \Delta_2$ .

Nous pouvons maintenant donner deux définitions équivalentes de l'ambiguïté dans une grammaire "context-free".

DEFINITION Une grammaire "context-free"  $G$  sera dite ambiguë si et seulement si elle contient une phrase terminale ayant plusieurs structures.

L'ambiguïté réside donc dans l'existence de deux dérivations non équivalentes de même origine  $S$  et de même extrémité terminale. En utilisant le corollaire 1.4.1 dans lequel on aurait choisi l'application  $f$  de façon à ce que  $f(u)=1$  pour tout  $u$ , on en déduit cette définition équivalente de l'ambiguïté :

DEFINITION EQUIVALENTE Une grammaire "context-free"  $G$  est ambiguë si et seulement si il existe deux dérivations distinctes  $\Delta_1$  et  $\Delta_2$  de la forme :

$$\Delta_1 = (x_0; 1, x_1; 1, x_2; \dots; 1, x_i)$$

$$\Delta_2 = (y_0; 1, y_1; 1, y_2; \dots; 1, y_j)$$

avec  $x_0 = y_0 = S$ ,  $x_i = y_j$  et  $x_i, y_j \in V_T^*$ .

Cette deuxième définition est adoptée par plusieurs auteurs, entre autres par Ginsburg [Gi]. Nous utiliserons la première qui a l'avantage d'être symétrique.

Considérons par exemple la grammaire  $G_1$  et sa grammaire parenthésée  $G'_1$  :

$$\begin{array}{ll} G_1^* : S \rightarrow XX & G'_1 : S \rightarrow \begin{array}{c} [XX] \\ S \quad S \end{array} \\ X \rightarrow a & X \rightarrow \begin{array}{c} [a] \\ X \quad X \end{array} \\ X \rightarrow Xa & X \rightarrow \begin{array}{c} [Xa] \\ X \quad X \end{array} \end{array}$$

Aux dérivations

$$(1) (S; 1, XX; 1, XaX; 1, aaX; 1, aaa)$$

$$(2) (S; 1, XX; 1, XaX; 2, Xaa; 1, aaa)$$

$$(3) (S; 1, XX; 1, aX; 1, aXa; 1, aaa)$$

sont associées les dérivations

$$(S; 1, \begin{array}{c} [XX] \\ S \quad S \end{array}; 1, \begin{array}{c} [[Xa]X] \\ SX \quad X \quad S \end{array}; 1, \begin{array}{c} [[[a]a]X] \\ SXXaX \quad X \quad S \end{array}; 1, \begin{array}{c} [[[[a]a][a]] \\ SXX \quad X \quad XX \quad XS \end{array})$$

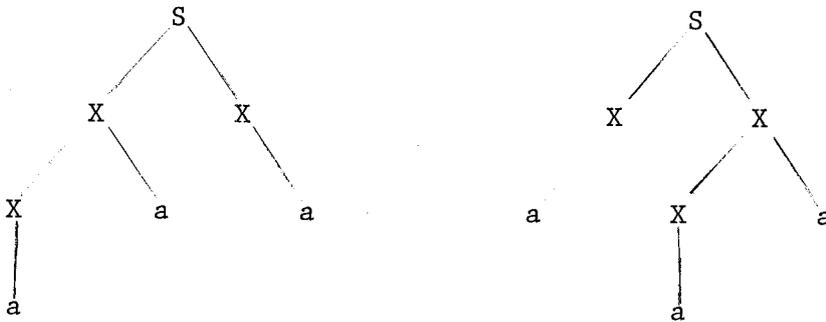
$$(S; 1, \begin{array}{c} [XX] \\ S \quad S \end{array}; 1, \begin{array}{c} [[Xa]X] \\ SX \quad X \quad S \end{array}; 2, \begin{array}{c} [[Xa][a]] \\ SX \quad XX \quad XS \end{array}; 1, \begin{array}{c} [[[[a]a][a]] \\ SXX \quad X \quad XX \quad XS \end{array})$$

$$(S; 1, \begin{array}{c} [XX] \\ S \quad S \end{array}; 1, \begin{array}{c} [[a]X] \\ SX \quad X \quad S \end{array}; 1, \begin{array}{c} [[a][Xa]] \\ SX \quad XX \quad XS \end{array}; 1, \begin{array}{c} [[[[a]][a[a]]] \\ SX \quad XXX \quad X \quad XS \end{array})$$

Les dérivations (1) et (2) sont donc équivalentes. Les dérivations (1) et (3) engendrent les structures

$$\begin{array}{c} [[ [a] a ] [a] ] \\ SXX \ X \ XX \ XS \end{array} \quad \text{et} \quad \begin{array}{c} [[ a ] [ [a] a ] ] \\ SX \ XXX \ X \ XS \end{array}$$

de la phrase terminale "aaa". La grammaire  $G_1$  est donc ambiguë. Les deux structures de "aaa" peuvent être représentées graphiquement ainsi :



### 1.5 Restrictions habituelles sur les grammaires "context-free"

L'étude que nous allons faire s'applique à toute grammaire "context-free" satisfaisant aux quatre restrictions habituelles :

- (1) - elle ne comporte pas de cycle, c'est-à-dire de symbole non terminal  $U$  tel que  $U \Rightarrow^+ U$
- (2) - elle ne comporte pas de règles dont le membre droit est la chaîne vide  $\Lambda$ ,
- (3) - pour tout symbole  $U \in V_T \cup V_N$  il existe toujours des chaînes  $x, y \in (V_T \cup V_N)^*$  telles que  $S \Rightarrow^* xUy$ ,
- (4) - pour tout symbole  $U \in V_T \cup V_N$  il existe toujours une chaîne  $t \in V_T^*$  telle que  $U \Rightarrow^* t$ .

Bien entendu on peut toujours se ramener à ce cas  $[G_1]$ . Il faut toutefois que le langage défini par la grammaire ne contienne pas la chaîne vide. Les restrictions (1) et (2) confèrent deux propriétés intéressantes à la relation  $\Rightarrow$  :

- la relation  $\Rightarrow^*$  devient une relation d'ordre partiel. En effet, par définition elle était déjà réflexive et transitive et des restrictions (1), (2) on déduit que  $x \Rightarrow^* y$  et  $y \Rightarrow^* x$  entraîne  $x=y$  pour tout  $x, y \in (V_T \cup V_N)^*$ , c'est-à-dire que la relation  $\Rightarrow^*$  est aussi antisymétrique.

- la deuxième propriété est exprimée dans le lemme qui suit :

LEMME 1.5.1 Si l'on désigne par  $g$  le nombre de règles de la grammaire considérée  $G$  alors, pour tout couple de chaînes  $x, y \in (V_T \cup V_N)^*$  et tout entier  $n > 0$  tels que  $x \Rightarrow^n y$ , on a la relation :

$$n \leq 2g|y| - g$$

DEMONSTRATION Si  $|y|=1$  alors, du fait de la restriction (2),  $|x|=1$  et, du fait de la restriction (1),  $n \leq g$ . Le théorème est donc vrai pour  $|y|=1$ ; supposons le vrai pour  $|y| < p$  et démontrons qu'il est vrai pour  $|y|=p$ . Si  $x \Rightarrow^n y$ , avec  $|y|=p$ , soit  $i$  le plus grand entier tel que  $0 \leq i \leq n$  et tel qu'il existe une chaîne  $u$  avec  $|u| > 1$  et

$$x \Rightarrow^{n-i} u \Rightarrow^i y$$

Du fait des restrictions (1), (2) on a toujours

$$n-i \leq g$$

Du fait que  $|u| > 1$  il existe  $u_1 \neq \Lambda$  et  $u_2 \neq \Lambda$  tel que  $u = u_1 u_2$ . D'après le lemme 1.21 il existe donc des entiers  $k$  et  $\ell$  et des chaînes  $y_1, y_2$  tels que  $y = y_1 y_2$ ,  $k + \ell = i$ ,  $u_1 \Rightarrow^k y_1$ ,  $u_2 \Rightarrow^\ell y_2$ . On a  $|y_1| < p$ ,  $|y_2| < p$  et donc par supposition

$$k \leq 2g|y_1| - g, \ell \leq 2g|y_2| - g \text{ d'où } i \leq 2g|y| - 2g$$

On en déduit que

$$n \leq 2g|y| - g$$

A partir de maintenant nous utiliserons le terme de grammaire pour désigner une grammaire "context-free" satisfaisant aux restrictions (1), (2), (3) et (4).

1.6 Relations  $\alpha, \gamma, \delta$ 

Pour toute grammaire  $G$  définissons trois relations binaires sur son vocabulaire  $V_T \cup V_N$ . Nous les noterons  $\alpha$  pour "adjacent",  $\gamma$  pour "gauche" et  $\delta$  pour "droite". Il est à noter que ces relations sont définies sur un ensemble fini.

DEFINITION Pour tout  $A, B \in V_T \cup V_N$  :

$A\alpha B \equiv$  [il existe  $U \in V_N$  et  $x, y \in (V_T \cup V_N)^*$  tels que  $U \rightarrow xAB y$ ]

$A\gamma B \equiv$  [il existe  $y \in (V_T \cup V_N)^*$  tel que  $A \rightarrow By$ ]

$A\delta B \equiv$  [il existe  $x \in (V_T \cup V_N)^*$  tel que  $B \rightarrow xA$ ]

Il faut noter la dissymétrie entre  $\gamma$  et  $\delta$ . Ces relations  $\alpha, \gamma, \delta$  ont pour principale propriété :

LEMME 1.6.1 Une condition nécessaire et suffisante pour que dans une grammaire  $G$  il existe une phrase dont la structure soit de la forme  $xAuvBy$  avec  $x, y \in (V_T \cup V_N \cup V_{\Gamma} \cup V_{\perp})^*$ ,  $A, B \in V_T \cup V_N$ ,  $u \in V_{\Gamma}^*$ ,  $|u|=i$ ,  $v \in V_{\perp}^*$ ,  $|v|=j$  est que  $A\delta^i \alpha \gamma^j B$ .

DEMONSTRATION La condition est suffisante. Si  $A\delta^i \alpha \gamma^j B$  alors il existe  $U, V \in V_T \cup V_N$  tels que  $A\delta^i U$ ,  $U\alpha V$ ,  $V\gamma^j B$ . Il existe donc une règle de  $G$  de la forme  $Z \rightarrow \dots UV \dots$  [\*]. D'après la restriction (3) sur les grammaires au paragraphe 1.5 on a  $S \Rightarrow^* \dots Z \dots$  dans  $G$  et donc aussi  $S \Rightarrow^* \dots Z \dots$  dans la grammaire parenthésée  $G'$ . On en conclut que  $S \Rightarrow^* \dots UV \dots$  dans  $G'$ . Du fait que  $A\delta^i U$  et  $V\gamma^j B$  on a  $U \Rightarrow^i \dots Au$ ,  $V \Rightarrow^j vB \dots$  dans  $G'$  avec  $u \in V_{\Gamma}^*$ ,  $|u|=i$  et  $v \in V_{\perp}^*$ ,  $|v|=j$ . De ce fait  $S \Rightarrow^* \dots AuvB \dots$  dans  $G'$ .

La condition est nécessaire. Soit  $n$  l'entier tel que  $S \Rightarrow^n xAuvBy$  dans la grammaire parenthésée  $G'$ . Si  $n=1$  le théorème est vrai car  $|u|=|v|=0$  et  $A\delta^0 \alpha \gamma^0 B$ . Supposons le théorème vrai pour  $n$  et démontrons qu'il est vrai  $n+1$ . Si  $S \Rightarrow^{n+1} xAuvBy$  dans  $G'$  il existe donc une chaîne  $s$  telle que  $S \Rightarrow^n s \Rightarrow xAuvBy$  dans  $G'$ . Soit  $Z \rightarrow [z]$  la règle utilisée pour transformer  $s$  en  $xAuvBy$ . En remarquant que  $z \neq \Lambda$  du fait de  $\overset{Z}{z}$  la

[\*] Pour représenter une chaîne commençant par un symbole  $X$  (respectivement se terminant par un symbole  $X$ ) on note  $X \dots$  (respectivement  $\dots X$ ).

restriction (2) sur les grammaires au paragraphe 1.5, la transformation de  $s$  en  $x\text{AuvBy}$  ne peut se faire que de quatre façons :

$$(1) s = x_1 Z x_2 \text{AuvBy} \Rightarrow x_1 \begin{bmatrix} z \\ Z \end{bmatrix} x_2 \text{AuvBy} = x \text{AuvBy} \text{ avec } x_1, x_2 \in (V_T \cup V_N \cup V_U \cup V_V)^*.$$

Le théorème étant supposé vrai pour  $n$  il est vrai pour  $s$  et donc  $A\delta^i \alpha \gamma^j B$ .

$$(2) s = x \text{AuvBy}_1 Z y_2 \Rightarrow x \text{AuvBy}_1 \begin{bmatrix} z \\ Z \end{bmatrix} y_2 = x \text{AuvBy} \text{ avec } y_1, y_2 \in (V_T \cup V_N \cup V_U \cup V_V)^*.$$

Même démonstration que dans le cas (1).

$$(3) s = x_1 Z x_2 v \text{By} \Rightarrow x_1 \begin{bmatrix} z \\ Z \end{bmatrix} x_2 v \text{By} = x \text{AuvBy} \text{ avec } x_1 \in (V_T \cup V_N \cup V_U \cup V_V)^* \text{ et } x_2 \in V_U^*.$$

Le théorème étant supposé vrai pour  $n$  il est vrai pour  $s$  et  $Z\delta^k \alpha \gamma^j B$  avec  $k = |x_2|$ . Du fait que  $u = \begin{bmatrix} z \\ Z \end{bmatrix}$  on a  $i = k + 1$  et du fait que  $A\delta Z$  on a  $A\delta^{k+1} \alpha \gamma^j B$  donc  $A\delta^i \alpha \gamma^j B$ .

$$(4) s = x \text{Auy}_1 Z y_2 \Rightarrow x \text{Auy}_1 \begin{bmatrix} z \\ Z \end{bmatrix} y_2 = x \text{AuvBy} \text{ avec } y_1 \in V_U^* \text{ et } y_2 \in (V_T \cup V_N \cup V_U \cup V_V)^*.$$

La démonstration est analogue à celle du cas (3).

Signalons que ce lemme permet d'énumérer tous les couples de symboles adjacents  $A, B$  pouvant apparaître dans une phrase d'une grammaire  $G$  : en effet, déterminer s'il existe des chaînes  $x, y$  telles que  $s \Rightarrow^* xAB y$  revient à savoir si  $A\delta^* \alpha \gamma^* B$ .

### 1.7 Problème de l'analyse syntaxique.

Etant donné une grammaire  $G$  de vocabulaire  $V_T \cup V_N$ , analyser une chaîne  $s \in (V_T \cup V_N)^*$  consistera à reconnaître si cette chaîne est une phrase de  $G$  et, dans l'affirmative, d'en donner chacune de ses structures. Par algorithme d'analyse syntaxique ou analyseur nous entendrons un processus permettant d'analyser toute chaîne terminale de la grammaire  $G$ .

En fait l'analyseur ne fournit pas chaque structure sous forme explicite mais sous forme d'une dérivation  $\Delta = (S = x_0; i_1, x_1; \dots; i_n, x_n = s)$  l'engendrant (revoir au paragraphe 1.4 la définition de la structure engendrée par une dérivation). Suivant

que la suite  $x_0, x_1, \dots, x_n$  est fournie dans le sens croissant ou décroissant des indices, on parle d'analyse descendante ou ascendante, l'axiome  $S=x_0$  étant considéré en quelque sorte comme un sommet. Trois remarques essentielles s'imposent au sujet des algorithmes d'analyse syntaxique les plus connus :

- Tout d'abord l'ordre dans lequel sont appliquées les règles pour construire la dérivation  $\Delta$  est imposé d'avance ; plus précisément, si l'algorithme est descendant, à chaque étape, la transformation de  $x_i$  en  $x_{i+1}$  est faite en appliquant la règle au symbole non terminal se trouvant placé le plus à gauche dans  $x_i$ , tandis que si l'algorithme est ascendant la transformation de  $x_i$  en  $x_{i-1}$  est faite de telle façon que pour réobtenir  $x_i$  à partir de  $x_{i-1}$  il faudrait appliquer la règle au symbole non terminal placé le plus à droite dans  $x_{i-1}$ .

- Si l'algorithme est général c'est-à-dire s'il s'applique à n'importe quel langage "context-free", alors il procède toujours par essais successifs. Ces essais sont soit faits en séquence (algorithme d'Irons [Ir.1][Ir.2]) soit en parallèle (algorithme de Cock [\*]). Les algorithmes qui ne sont pas généraux (algorithme de Floyd décrit en [F1.2] ou de Knuth [Kn] par exemple) peuvent facilement être rendus généraux en leur adjoignant un dispositif permettant de procéder par essais successifs. A ce sujet signalons les travaux de Floyd [F1.3] sur une façon systématique de programmer les algorithmes nécessitant des essais successifs.

- A chaque grammaire et à chacun de ces algorithmes peut être associé un automate "pushdown" classique c'est-à-dire un automate à une pile qui, d'une part, permet de reconnaître les chaînes du langage défini par cette grammaire et d'autre part, est ou n'est pas déterministe suivant que l'algorithme considéré peut ou ne peut pas se passer d'essais successifs pour analyser toutes les chaînes du langage. De ceci il découle que les classes de langages pour lesquelles ces algorithmes n'ont

---

[\*] Cet algorithme ainsi que d'autres est décrit en [BC].

pas besoin d'essais successifs sont incluses dans la classe des langages "déterministes", c'est-à-dire, l'ensemble des langages qui peuvent être acceptés par des automates "pushdown" déterministes. Cette inclusion n'est pas nécessairement stricte, c'est le cas de l'analyseur proposé par D.E. Knuth [Kn]. Pour le lecteur qui n'est pas familiarisé avec les automates "pushdown" et les langages déterministes nous renvoyons aux références [GL], [Gi] et [GG].

Signalons qu'une très bonne étude comparative des algorithmes d'analyse connus a été faite par T.V. Griffiths et S.R. Petrick [GP]. Chaque algorithme y est décrit au moyen d'un automate à deux piles plus simple que l'automate à une pile. Nous utiliserons un automate à deux piles très analogue, pour décrire nos deux principaux analyseurs qui seront de type ascendant. Deux particularités les concernant doivent être mentionnées.

- L'ordre d'application des règles ne correspond pas à celui mentionné plus haut et même, dans le cas du premier algorithme, cet ordre n'est pas imposé d'avance ; ce degré de liberté permet d'éviter certains essais.

- Pour décrire le premier algorithme il faut nécessairement disposer de deux piles.

Décrivons maintenant l'automate à deux piles que nous utiliserons.

### 1.8 Automate à deux piles

DEFINITION Un automate à deux piles sera défini par :

- un vocabulaire  $W$
- un premier sous-ensemble de  $W^* \times W^*$  dont chaque élément sera appelé configuration initiale,
- un deuxième sous-ensemble de  $W^* \times W^*$  dont chaque élément sera appelé configuration finale,
- une relation binaire finie sur  $W^* \times W^*$  notée  $\rightarrow$  dont les éléments seront appelés instructions.

Tout élément  $(x,y)$  de  $W^* \times W^*$  sera appelé configuration. Physiquement l'automate comprend deux piles et nous dirons qu'elles ont la configuration  $(x,y)$  si les chaînes  $x$  et  $y$  sont disposées ainsi dans ces deux piles :



Le fonctionnement de l'automate est le suivant :

- les deux piles ont au départ une configuration initiale.
- si à un moment donné la configuration des deux piles est :



$u_1, u_2, x_1, x_2$  étant des chaînes quelconques de  $W^*$ , l'instruction

$$(x_1, x_2) \rightarrow (y_1, y_2)$$

si elle existe, s'exécutera et transformera cette configuration en :



- Le caractère non déterministe de l'automate se traduit par le fait que pour une configuration donnée il peut y avoir plusieurs instructions susceptibles d'être exécutées. Dans ce cas on peut considérer que l'automate se scinde en plusieurs "sous-automates" qui fonctionnent alors en parallèle.

- L'automate ou le sous-automate s'arrête de fonctionner quand il n'existe plus d'instruction correspondant à la configuration de ses deux piles ou que cette configuration est une configuration finale.

Précisons le caractère déterministe d'un automate à deux piles :

DEFINITION Un automate à deux piles de vocabulaire  $W$  sera considéré comme non déterministe si et seulement si il comporte deux instructions distinctes

$$(x_1, y_1) \rightarrow (x'_1, y'_1) \text{ et } (x_2, y_2) \rightarrow (x'_2, y'_2)$$

pour lesquelles il existe des chaînes  $u, v \in V^*$  telles que

$$ux_1 = ux_2 \text{ et } y_1v = y_2v$$

Définissons la notion de cheminement :

DEFINITION Nous appellerons cheminement d'un automate toute suite  $C_0, C_1, \dots, C_n$  de configurations telle que :

- (1)  $C_0$  soit une configuration initiale
- (2)  $C_n$  soit une configuration finale ou une configuration à laquelle ne s'applique aucune instruction de l'automate,
- (3) si  $n \neq 0$  alors pour tout  $i \in \{1, 2, \dots, n\}$  le passage de la configuration  $C_{i-1}$  à la configuration  $C_i$  se fasse en exécutant une instruction de l'automate.

Considérons par exemple, l'automate à deux piles de vocabulaire  $\{a, b, \#\}$  défini ainsi :

#	x#
---	----

Configurations initiales

x étant une chaîne quelconque sur  $\{a, b\}$

Configuration finale

#	#
---	---

Instructions

$$(\#, a) \rightarrow (\#a, \Lambda)$$

$$(\#, b) \rightarrow (\#b, \Lambda)$$

$$(a, a) \rightarrow (aa, \Lambda)$$

$$(b, b) \rightarrow (bb, \Lambda)$$

$$(a, b) \rightarrow (\Lambda, \Lambda)$$

$$(b, a) \rightarrow (\Lambda, \Lambda)$$

Cet automate est déterministe et permet de reconnaître si la chaîne  $x$  contient ou ne contient pas autant de "a" que de "b". En effet, le cheminement ne se termine par la configuration finale que si l'on est dans la première éventualité. Voici deux exemples de cheminement :

(0) #	aababb#	(0) #	bababb#
(1) #a	ababb#	(1) #b	ababb#
(2) #aa	babb#	(2) #	babb#
(3) #a	abb#	(3) #b	abb#
(4) #aa	bb#	(4) #	bb#
(5) #a	b#	(5) #b	b#
(6) #	#	(6) #bb	#

CHAPITRE 2 : Pr cedence totale

## 2.1 Analyseur et relations de précedence totale : définition

Considérons une grammaire  $G$ , trois relations binaires quelconques  $\prec, \bar{=}, \succ$  définies sur son vocabulaire  $V_T \cup V_N$ , et l'automate à deux piles de vocabulaire  $V_T \cup V_N \cup \{\#, [, ]\}$ , avec  $(V_T \cup V_N) \cap \{\#, [, ]\} = \emptyset$ , défini ainsi :

#	x #	Configurations initiales
		x étant une chaîne quelconque de $(V_T \cup V_N)^*$
		Configuration finale
#	S #	S étant l'axiome de $G$
		Instructions

Pour tout  $A, B \in V_T \cup V_N \cup \{\#\}$  et toute règle  $U \rightarrow u$  de  $G$  on a :

- (1)  $(A, B) \rightarrow (A \bar{[} B, \Lambda)$  si et seulement si  $A \prec B$  ou  $A = \#$  et  $B \neq \#$ .
- (2)  $(A, B) \rightarrow (AB, \Lambda)$  si et seulement si  $A \bar{=} B$ .
- (3)  $(A, B) \rightarrow (A \bar{]} B, \Lambda)$  si et seulement si  $A \succ B$  ou  $A \neq \#$  et  $B = \#$ .
- (4)  $(\bar{[} u, \Lambda) \rightarrow (\Lambda, U)$

De la façon dont est défini cet automate il découle que s'il existe un cheminement se terminant par la configuration finale alors  $x$  est une phrase de la grammaire  $G$ . De plus, de ce cheminement on déduit directement une dérivation engendrant une structure de  $x$ . Nous considérons donc que cet automate est un analyseur pour la grammaire  $G$ , si, à chaque structure d'une phrase terminale  $x$  quelconque correspond au moins un cheminement se terminant par la configuration finale.

**DEFINITION** Si l'automate précédemment décrit est un analyseur pour la grammaire  $G$ , alors cet analyseur ainsi que les relations associées  $\prec, \bar{=}, \succ$  seront dits de précedence totale.

Bien entendu, pour une grammaire donnée  $G$  il existe toujours un analyseur de précedence totale ; il suffit, par exemple de prendre  $A \prec B$ ,  $A \bar{=} B$ ,  $A \succ B$  quelque soit  $A, B \in V_T \cup V_N$ . Donnons deux exemples de relations de précedence totale :

Soit le langage constitué uniquement de la chaîne "aba" défini par la grammaire :

$$G_2 : \begin{aligned} S &\rightarrow AB \\ A &\rightarrow ab \\ B &\rightarrow a \end{aligned}$$

Considérons les relations  $\leq, =, >$  définies par le tableau

	S	A	B	a	b
S	.	.	.	.	.
A	.	.	=	<	.
B	.	.	.	.	.
a	.	.	.	.	=
b	.	.	.	>	.

sur lequel figurent, à l'intersection de la  $i^{\text{ème}}$  ligne et de la  $j^{\text{ème}}$  colonne, la ou les relations éventuellement vérifiées entre le  $i^{\text{ème}}$  et le  $j^{\text{ème}}$  symbole. Si nous considérons l'automate associé à cette grammaire et à ces relations, alors le cheminement suivant correspond à la structure  $[[ab][a]]$  de la phrase terminale "aba" :

(0) #	aba#	(6) #	[A]a #
(1) #	[a ba#	(7) #	[A[a] #
(2) #	[ab a#	(8) #	[A B#
(3) #	[ab] a#	(9) #	[AB #
(4) #	Aa#	(10) #	[AB] #
(5) #	[A a#	(11) #	S#

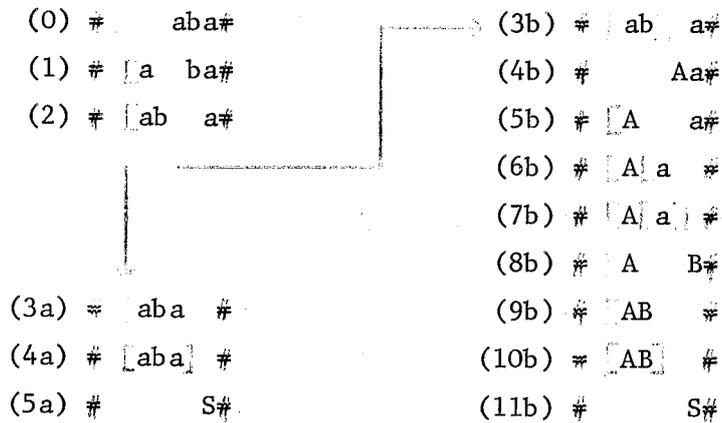
La chaîne "aba" n'ayant qu'une seule structure, les relations  $\leq, =, >$  du tableau précédent sont donc de précedence totale pour la grammaire  $G_2$ . Considérons maintenant le même langage défini par la grammaire ambiguë :

$$G_3 : \begin{aligned} S &\rightarrow aba \\ S &\rightarrow AB \\ A &\rightarrow ab \\ B &\rightarrow a \end{aligned}$$

Les relations  $<, =, >$  du tableau qui suit sont de précédence totale pour  $G_2$ .

	S	A	B	a	b
S	.	.	.	.	.
A	.	.	=	<	.
B	.	.	.	.	.
a	.	.	.	.	=
b	.	.	.	=	>

En effet, aux deux structures  $[aba]$  et  $[[ab]a]$  de la chaîne "aba" correspondent les deux cheminements :



Il faut remarquer que dans ces deux exemples il n'existe pas de cheminement "parasite" c'est-à-dire de cheminement ne se terminant pas par la configuration finale. On ne pourra pas toujours se ramener à ce cas. De plus, alors que l'automate correspondant au premier exemple est déterministe, celui du deuxième exemple ne l'est pas du fait des deux instructions  $(b,a) \rightarrow (ba, \Lambda)$  et  $(b,a) \rightarrow ([b], a)$  provenant des relations  $b=a$  et  $b>a$ .

## 2.2 Une classe simple de relations de précédence totale

Pour une grammaire donnée, il existe en général plusieurs analyseurs de précédence totale et à chacun sont associées d'une façon biunivoque des relations de précédence totale.

En utilisant le théorème qui suit il est facile d'énumérer tous les analyseurs de précedence totale capables d'analyser non seulement les chaînes terminales d'une grammaire donnée, mais aussi toutes les autres chaînes construites sur son vocabulaire  $V_T \cup V_N$ .

THEOREME 2.2.1 Une condition nécessaire et suffisante pour que trois relations binaires  $\leq, \equiv, \succ$  sur le vocabulaire  $V_T \cup V_N$  d'une grammaire  $G$  soient de précedence totale et que l'analyseur associé à ces relations puisse analyser toutes les chaînes construites sur  $V_T \cup V_N$ , est que chacune des inclusions suivantes soit vérifiée :

$$\alpha \leq \equiv, \alpha \gamma^+ \leq \leq, \delta^+ \alpha \leq \succ, \delta^+ \alpha \gamma^+ \leq \succ \quad (\leq \cup \succ)$$

(Les relations  $\alpha, \gamma, \delta$  sont celles qui sont définies au paragraphe 1.6.)

DEMONSTRATION Démontrons que la condition est nécessaire. Supposons que les relations  $\leq, \equiv, \succ$  sont de précedence totale et considérons l'analyseur qui leur est associé ainsi que deux symboles quelconques  $A, B \in V_T \cup V_N$ . D'après le lemme 1.6.1, si  $A \delta^i \alpha \gamma^j B$  alors il existe une phrase  $s$  de structure  $x A u v y$  avec  $x, y \in (V_T \cup V_N \cup \{ \epsilon \} )^*$ ,  $u \in V_T^*$ ,  $|u|=i$ ,  $v \in V_N^*$ ,  $|v|=j$ .

- Si  $A \alpha B$  alors  $A \delta^i \alpha \gamma^j B$  avec  $i=0$  et  $j=0$  et de ce fait  $u=v=\Lambda$ . L'analyseur devra donc réduire  $A$  et  $B$  en même temps dans la phrase  $s$  et donc  $A \equiv B$ . On en conclut que  $\alpha \leq \equiv$ .
- Si  $A \alpha \gamma^+ B$  alors  $A \delta^i \alpha \gamma^j B$  avec  $i=0$  et  $j>0$  et de ce fait  $u=\Lambda$  et  $v \neq \Lambda$ . L'analyseur devra donc réduire  $B$  avant  $A$  dans la phrase  $s$  et donc  $A \succ B$ .  
On en conclut que  $\alpha \gamma^+ \leq \succ$ .
- Si  $A \delta^+ \alpha B$  alors  $A \delta^i \alpha \gamma^j B$  avec  $i>0$  et  $j=0$  et de ce fait  $u \neq \Lambda$  et  $v=\Lambda$ .  
L'analyseur devra donc réduire  $A$  avant  $B$  dans la phrase  $s$  et donc  $A \succ B$ .  
On en conclut que  $\delta^+ \alpha \leq \succ$ .
- Si  $A \delta^+ \alpha \gamma^+ B$  alors  $A \delta^i \alpha \gamma^j B$  avec  $i>0$  et  $j>0$  et de ce fait  $u \neq \Lambda$  et  $v \neq \Lambda$ .  
L'analyseur devra donc réduire soit  $B$  avant  $A$ , soit  $A$  avant  $B$  et donc  $A \succ B$  ou  $A \succ B$ .  
On en conclut que  $\delta^+ \alpha \gamma^+ \leq \succ \quad (\leq \cup \succ)$ .

Démontrons que la condition est suffisante. Considérons une phrase quelconque  $x \in S$  de  $G$  et une de ses structures  $s$ . On peut toujours poser :

$$x = X_1 X_2 \dots X_n \text{ et } s = v_1 X_1 u_1 v_1 X_2 u_2 v_2 \dots v_n X_n u_n$$

avec  $n > 0$  et pour  $i = 1, 2, \dots, n$

$$X_i \in V_T \cup V_N, u_i \in V_j^*, v_i \in V_l^*$$

et de plus :

$$v_1 \neq \Lambda \text{ et } u_n \neq \Lambda$$

Pour démontrer que la condition est suffisante, compte tenu de la définition d'un analyseur de précédence totale (paragraphe 2.1) et du lemme 1.3.1., il suffit de montrer qu'il existe deux entiers  $p$  et  $q$  qui satisfont aux quatre conditions :

- (a)  $1 < p < q < n$
- (b)  $u_q \neq \Lambda$  et si  $q \neq n$  alors  $X_q \succ X_{q+1}$
- (c) si  $p \neq q$  alors pour  $i = p+1, p+2, \dots, q$ ,  $u_{i-1} v_i = \Lambda$  et  $X_{i-1} \bar{=} X_i$
- (d)  $v_p \neq \Lambda$  et si  $p \neq 1$  alors  $X_{p-1} \prec X_p$  et pour  $i = 2, 3, \dots, p$ ,  $X_{i-1} \prec X_i$  ou  $X_{i-1} \bar{=} X_i$

- Soit  $q$  le plus petit entier tel que  $1 < q < n$  et tel que la condition (b) soit satisfaite. Cet entier existe car pour  $q = n$  la condition (b) est vérifiée.
- Soit  $p$  le plus petit entier tel que  $1 < p < q$  et tel que la condition (c) soit satisfaite. Cet entier existe car pour  $p = q$  la condition (c) est vérifiée.

Les entiers  $p$  et  $q$  satisfont aux conditions (a), (b), (c) ; il ne reste plus qu'à démontrer qu'ils satisfont aussi à la condition (d). Si  $p = 1$  la condition (d) est satisfaite sinon considérons un entier quelconque  $i$  tel que  $1 < i < p$ . Quatre cas se présentent et dans chaque cas nous nous servirons de la condition nécessaire du lemme 1.6.1 :

- (1)  $u_{i-1} = \Lambda, v_i = \Lambda$ . Donc  $X_{i-1} \alpha X_i$  et du fait que  $\alpha \in \bar{C}$ ,  $X_{i-1} \neq X_i$ . Le cas où  $i=p$  est exclu car il contredit la définition de  $p$ .
- (2)  $u_{i-1} = \Lambda, v_i \neq \Lambda$ . Donc  $X_{i-1} \alpha \gamma^+ X_i$  et du fait que  $\alpha \gamma^+ \in \bar{C}$ ,  $X_{i-1} \leq X_i$ .
- (3)  $u_{i-1} \neq \Lambda, v_i = \Lambda$ . Donc  $X_{i-1} \delta^+ \alpha X_i$  et du fait que  $\delta^+ \alpha \in \bar{C}$ ,  $X_{i-1} \geq X_i$ . Ce cas est exclu car il contredit la définition de  $q$ .
- (4)  $u_{i-1} \neq \Lambda, v_i \neq \Lambda$ . Donc  $X_{i-1} \delta^+ \alpha \gamma^+ X_i$  et du fait que  $\delta^+ \alpha \gamma^+ \in \bar{C}$ ,  $X_{i-1} \leq X_i$  où  $X_{i-1} \geq X_i$ . Le cas où  $X_{i-1} \geq X_i$  est exclu car il contredit la définition de  $q$ .

La condition  $d$  est donc satisfaite.

### 2.3 Analyseur de précedence totale deterministe

Le cas où l'analyseur de précedence totale est deterministe est particulièrement intéressant du fait de la simplicité de l'algorithme d'analyse qui lui correspond. Un autre point important est la "rapidité" d'un tel analyseur. Attachons un temps d'exécution à chaque instruction de l'automate que constitue l'analyseur. On a alors la propriété suivante :

**THEOREME 2.3.1** L'analyse d'une chaîne terminale  $t$  par un analyseur de précedence totale deterministe se fait en un temps au plus proportionnel à la longueur  $|t|$  de cette chaîne. Plus exactement, en désignant par  $N$  le nombre d'instructions exécutées par l'automate et par  $g$  le nombre de règles de la grammaire considérée, on a  $N \leq (6g+1)|t|$ .

**DEMONSTRATION** Reportons nous à la description d'un analyseur de précedence totale au paragraphe 2.1. Soit  $n$  le nombre d'instructions du type (4) exécutées par l'automate pour analyser  $t$  et soit  $p$  le nombre de symboles différents de  $\#$  contenus dans la pile de droite à sa dernière configuration. Le nombre d'instructions du type (3) exécutées est au plus égal à  $n+1$  et du fait que le nombre de symboles qui sont passés de la pile droite dans la pile gauche est  $|t|+n-p$ , le nombre d'instructions du type (1) ou (2) exécutées est  $|t|+n-p$ . Au total on a donc  $N \leq 3n+|t|+1-p$ , d'où  $N \leq 3n+|t|+1$ . D'après le lemme 1.5.1, on a  $n \leq 2g|t|-g$  et donc  $N \leq (6g+1)|t|$ . En majorant le temps d'exécution de chaque instruction par celui de l'instruction la plus longue à s'exécuter, le premier point du théorème se trouve démontré.

Mentionnons un sous-produit intéressant de cette démonstration : si  $t$  est une phrase, alors, pour l'analyser, l'automate exécutera exactement  $3n + |t| - 1$  instructions,  $n$  étant l'entier tel que  $S \Rightarrow^n t$ .

Comment reconnaître si un analyseur de précedence totale est déterministe ?

En se reportant à la définition d'un automate à deux piles déterministe (paragraphe 1.8) et à la description d'un analyseur de précedence totale (paragraphe 2.1) on conclut immédiatement :

**THEOREME 2.3.2** Une condition nécessaire et suffisante pour qu'un analyseur de précedence totale défini dans une grammaire  $G$  soit déterministe est que :

- (1) toutes les règles de  $G$  aient des membres droits distincts, c'est-à-dire que  $X \rightarrow u$  et  $Y \rightarrow u$  entraîne  $X=Y$ ,
- (2) les relations de précedence totale  $\prec, \bar{\prec}, \succ$  associées à cet analyseur soient incompatibles deux à deux, c'est-à-dire que  $(\prec \cap \bar{\prec}) \cup (\succ \cap \bar{\prec}) \cup (\prec \cap \succ) = \emptyset$ .

Nous allons étudier sous quelles conditions il est possible de définir des relations de précedence totale incompatibles deux à deux dans une grammaire donnée.

## 2.4 Existence de relations de précédence totale incompatibles deux à deux

Le théorème qui suit permet de décider si dans une grammaire donnée il existe des relations de précédence totale incompatibles deux à deux.

THEOREME 2.4.1 Une condition nécessaire et suffisante pour qu'il existe des relations de précédence incompatibles deux à deux pour une grammaire  $G$ , dans laquelle ont été définies les relations  $\alpha$ ,  $\gamma$ ,  $\delta$ , est que

$$(\delta^+ \alpha \cap \alpha) \cup (\alpha \gamma^+ \cap \alpha) \cup (\delta^+ \alpha \gamma^+ \cap \alpha) \cup (\delta^+ \alpha \cap \alpha \gamma^+) = \emptyset.$$

DEMONSTRATION Démontrons par l'absurde que la condition est nécessaire. Supposons que dans la grammaire  $G$  il existe les relations  $\leq, =, >$  incompatibles deux à deux et que  $(\delta^+ \alpha \cap \alpha) \cup (\alpha \gamma^+ \cap \alpha) \cup (\delta^+ \alpha \gamma^+ \cap \alpha) \cup (\delta^+ \alpha \cap \alpha \gamma^+) \neq \emptyset$ .

De ce fait il existe deux entiers  $i, j$  et deux symboles  $A, B \in V_T \cup V_N$  tels que l'une des deux éventualités suivantes se présente :

- (1)  $A \alpha B$  et  $A \delta^i \alpha \gamma^j B$  avec  $i > 0, i+j > 0, j > 0$
- (2)  $A \delta^i \alpha B$  et  $A \alpha \gamma^j B$  avec  $i > 0, j > 0$

Eventualité (1)

D'après le lemme 1.6.1, il existe donc deux phrases  $s_1$  et  $s_2$  de structures respectives

$$\dots AB \dots \text{ et } \dots Au_1 v_1 B \dots \text{ avec } u_1 \in V_J^*, |u_1|=i, v_1 \in V_L^*, |v_1|=j.$$

Du fait de la restriction (4) du paragraphe 1.5 il existe donc deux phrases terminales  $t_1$  et  $t_2$  de structures respectives

$$\dots au_2 v_2 b \dots \text{ et } \dots au_2 u_1 v_1 v_2 b \dots \text{ avec } a, b \in V_T, u_2 \in V_J^*, v_2 \in V_L^*, \\ u_2 = \dots A \text{ ou } u_2 = \Lambda \text{ et } A = a, v_2 = B \dots \text{ ou } v_2 = \Lambda \text{ et } B = b.$$

Au cours de l'analyse de  $t_1$  l'analyseur traitera une phrase de structure  $\dots AB \dots$ . Les relations  $\leq, =, >$  étant incompatibles deux à deux, l'analyseur sera donc amené, au cours de l'analyse de  $t_2$ , à traiter une phrase de structure  $\dots Au_1 v_1 B \dots$ . Il devra donc, d'une part réduire  $A$  et  $B$  en même temps et d'autre part réduire  $A$  avant  $B$  ou  $B$  avant  $A$ , car  $u_1 v_1 \neq \Lambda$  du fait que  $i+j > 0$ . Par conséquent  $A \neq B$  et  $A < B$  ou  $A = B$  et  $A > B$  ce qui contredit le fait que les relations  $\leq, =, >$  sont incompatibles deux à deux.

Eventualité (2)

D'après le lemme 1.6.1 il existe donc deux phrases  $s_1$  et  $s_2$  de structures respectives

$$\dots Au_1 B \dots \text{ et } \dots Av_1 B \dots \text{ avec } u_1 \in V_{\lceil}^*, |u_1|=i, v_1 \in V_{\lfloor}^*, |v_1|=j.$$

Du fait de la restriction (4) du paragraphe 1.5 il existe donc deux phrases terminales  $t_1$  et  $t_2$  de structures respectives

$$\dots au_2 u_1 v_2 b \dots \text{ et } \dots av_2 v_1 v_2 b \dots \text{ avec } a, b \in V_T, u_2 \in V^*, v_2 \in V^*, \\ u_2 = \dots \rfloor_A \text{ ou } u_2 = \Lambda \text{ et } A = a, v_2 = \lfloor_B \dots \text{ ou } v_2 = \Lambda \text{ et } B = b.$$

Les relations  $\leq, =, \geq$  étant incompatibles deux à deux, au cours des analyses respectives de  $t_1$  et  $t_2$ , l'analyseur sera amené à traiter des phrases ayant respectivement pour structures  $\dots Au_1 B \dots$  et  $\dots Av_1 B \dots$ . Du fait que  $u_1 \neq \Lambda$  et  $v_1 \neq \Lambda$  car  $i > 0$  et  $j > 0$ , il devra donc d'une part réduire A avant B et d'autre part réduire B avant A. Par conséquent  $A \geq B$  et  $B \leq A$  ce qui contredit le fait que les relations  $\leq, =, \geq$  sont incompatibles deux à deux.

Démontrons que la condition est suffisante. Sur le vocabulaire  $V_T \cup V_N$  de G considérons les relations binaires  $\leq, =, \geq$  ainsi définies :

$$\begin{aligned} = &= \alpha \\ \leq &= \alpha \gamma^+ \\ \geq &= \delta^+ \alpha \cup (\delta^+ \alpha \gamma^+ \cap \overline{\alpha \gamma^+}) \end{aligned}$$

Du fait que  $\alpha \subset =$ ,  $\alpha \gamma^+ \subset \leq$ ,  $\delta^+ \alpha \subset \geq$ ,  $\delta^+ \alpha \gamma^+ \subset (\leq \cup \geq)$ , ces relations sont de précedence totale (théorème 2.2.1). On a

$$(\delta^+ \alpha \cap \alpha) \cup (\alpha \gamma^+ \cap \alpha) \cup (\delta^+ \alpha \gamma^+ \cap \alpha) \cup (\delta^+ \alpha \cap \alpha \gamma^+) = (\leq \cap =) \cup (\geq \cap =) \cup (\leq \cap \geq)$$

Si la condition du théorème est vérifiée on en conclut que ces relations de précedence totale sont incompatibles deux à deux.

## 2.5 Famille optimale de triplets de relations de précedence totale

DEFINITION Considérons une application qui à toute grammaire  $G$  fait correspondre un triplet de relations de précedence totale  $(\langle, \bar{=}, \rangle)_G$ . La famille de triplets ainsi définie sera dite optimale si et seulement si, quelle que soit la grammaire  $G$ , on est toujours dans l'un des deux cas :

- (1) les relations  $\langle, \bar{=}, \rangle$  sont incompatibles deux à deux,
- (2) les relations  $\langle, \bar{=}, \rangle$  ne sont pas incompatibles deux à deux, mais on ne peut trouver dans  $G$  d'autres relations de précedence totale qui le soient.

THEOREME 2.5.1 Les familles de triplets  $(\langle, \bar{=}, \rangle)_G$  de relations définies comme suit sont des familles optimales de relations de précedence totale :

$$\begin{aligned} \bar{=} &= \alpha \\ \langle &= \alpha\gamma^+ \cup \rho_1 \\ \rangle &= \delta^+\alpha \cup \rho_2 \end{aligned}$$

$\rho_1$  et  $\rho_2$  étant des relations binaires sur le vocabulaire  $V_T \cup V_N$  de  $G$  satisfaisant aux deux conditions :

$$\begin{aligned} \rho_1 \cap \rho_2 &= \overline{\delta^+\alpha\gamma^+} \cap \overline{\delta^+\alpha} \cap \overline{\alpha\gamma^+} \\ \rho_1 \cup \rho_2 &= \emptyset \end{aligned}$$

De plus l'analyseur associé aux relations  $\langle, \bar{=}, \rangle$  peut analyser toutes les chaînes construites sur le vocabulaire  $V_T \cup V_N$  de la grammaire  $G$ .

DEMONSTRATION Du fait que  $\alpha \in \bar{=}$ ,  $\alpha\gamma^+ \in \langle$ ,  $\delta^+\alpha \in \rangle$ ,  $\delta^+\alpha\gamma^+ \in (\langle \rangle)$ , les relations  $\langle, \bar{=}, \rangle$  sont de précedence totale et l'analyseur associé est capable d'analyser toutes les chaînes sur  $V_T \cup V_N$  (théorème 2.2.1). Du fait que

$$(\bar{=} \langle) \cup (\bar{=} \rangle) \cup (\langle \rangle) = (\alpha\gamma^+ / \alpha) (\delta^+\alpha / \alpha) (\delta^+\alpha\gamma^+ / \alpha) (\delta^+\alpha / \alpha\gamma^+)$$

la famille de triplets  $(\langle, \bar{=}, \rangle)_G$  est optimale (théorème 2.4.1).

Il est intéressant de remarquer que les relations  $\leq, \geq$  sont définies ici d'une façon symétrique. Cependant pour éviter certains reculs à l'analyseur associé au triplet  $(\leq, \overline{=}, \geq)_G$  on a intérêt à prendre  $\rho_1 = \emptyset$ . On obtient alors :

$\overline{=} = \alpha$
$\leq = \alpha\gamma^+$
$\geq = \delta^+\alpha U(\delta^+\alpha\gamma^+ \cap \overline{\alpha\gamma^+})$

Considérons par exemple le langage dont les chaînes sont de la forme :

$$a^i b^j a^j \cup a^i b^i a^j c \quad \text{avec } i > 0 \text{ et } j > 0,$$

on peut le définir par la grammaire :

$G_4 : S \rightarrow X$	$S \rightarrow VY$
$X \rightarrow aU$	$Y \rightarrow ac$
$X \rightarrow aX$	$Y \rightarrow aY$
$U \rightarrow bA$	$V \rightarrow aB$
$U \rightarrow bUA$	$V \rightarrow aVB$
$A \rightarrow a$	$B \rightarrow b$

et en utilisant les formules ci-dessus on obtient le tableau [\*] :

	S	X	Y	U	V	A	B	a	b	c
S	.	.	.	.	.	.	.	.	.	.
X	.	.	.	.	.	.	.	.	.	.
Y	.	.	.	.	.	.	.	.	.	.
U	.	.	.	.	.	$\overline{=}$	.	$\leq$	.	.
V	.	.	$\overline{=}$	.	.	.	$\overline{=}$	$\leq$	$\leq$	.
A	.	.	.	.	.	$\geq$	.	$\geq$	.	.
B	.	.	$\geq$	.	.	.	$\geq$	$\geq$	$\geq$	.
a	.	$\overline{=}$	$\overline{=}$	$\overline{=}$	$\overline{=}$	$\geq$	$\overline{=}$	$\leq$	$\leq$	$\overline{=}$
b	.	.	$\geq$	$\overline{=}$	.	$\overline{=}$	$\geq$	$\leq$	$\leq$	.
c	.	.	.	.	.	.	.	.	.	.

[\*] En remplaçant par le signe  $\leq$ , un ou plusieurs signes  $\geq$  encerclés de ce tableau, on obtient toutes les relations de précédence que l'on peut obtenir en faisant varier  $\rho_1$  et  $\rho_2$  dans les formules du théorème précédent.

Remarquons que toutes les règles de la grammaire  $G_4$  ont des membres droits distincts et que les relations du tableau sont incompatibles deux à deux. De ce fait l'analyseur associé à ces relations est donc déterministe. Voici les principales étapes de l'analyse des chaînes "aaabbaa" et "aabbaaac".

#	aaabbaa#	#	aabbaaac#
#	[a][a][a][b][b][a][a]	#	[a][a][b][b][a][a][ac]
#	[a][a][a][b][b][a] A#	#	[a][a][b][b][a][aY]
#	[a][a][a][b]bA]	A#	[a][a][b][b][aX]
#	[a][a][a][b]A]	#	[a][a][b][b]
#	[a][a][a]	#	[a][a][b]
#	[a][a]X]	#	[a][aB]
#	[aX]	#	[avB]
#	[X]	#	[vY]
#	S#	#	S#

Signalons, comme nous le verrons dans la démonstration du théorème 2.6.3 que le langage défini par la grammaire  $G_4$  n'est pas reconnaissable par un automate à une pile classique déterministe.

## 2.6 Grammaires et langages de précedence totale

DEFINITION Nous appellerons grammaire de précedence totale toute grammaire pour laquelle il existe un analyseur de précedence totale déterministe.

Des théorèmes 2.3.2 et 2.4.1 on déduit immédiatement une définition équivalente.

DEFINITION EQUIVALENTE : Une grammaire de précedence totale  $G$  est une grammaire qui satisfait aux deux conditions :

- (1) Toutes ses règles ont des membres droits distincts,
- (2)  $(\delta^+ \alpha \cap \alpha) \cup (\alpha \gamma^+ \cap \alpha) \cup (\delta^+ \alpha \gamma^+ \cap \alpha) \cup (\delta^+ \alpha \cap \alpha \gamma^+) = \emptyset$ .

Donnons une propriété essentielle des grammaires de précedence totale.

THEOREME 2.6.1 Une grammaire de précedence totale ne peut être ambiguë.

DEMONSTRATION Considérons une grammaire de précedence totale  $G$ . Il existe donc un analyseur de précedence totale déterministe pour  $G$ . Si  $G$  est ambiguë il existe, par définition, une phrase terminale  $t$  de  $G$  ayant plusieurs structures et à chacune de celles-ci correspond au moins un cheminement de l'analyseur pour analyser  $t$ . Cet analyseur ne peut donc être déterministe et il y a contradiction.

Si nous reconsidérons par exemple la grammaire  $G_1$  de la fin du paragraphe 1.3

$$G_1 : \begin{array}{l} S \rightarrow XX \\ X \rightarrow a \\ X \rightarrow Xa \end{array}$$

qui était ambiguë, nous pouvons facilement vérifier qu'effectivement elle n'est pas de précedence totale car, du fait que  $X\delta X$  et  $X\alpha a$ , on a  $X\delta\alpha a$  et, du fait que  $X\alpha a$ , on a  $\delta\alpha\alpha \neq \emptyset$ .

Nous savons que le problème de déterminer si une grammaire est ou n'est pas ambiguë, ne peut être résolu en général, cependant le théorème 2.6.1 peut contribuer à sa solution dans certains cas particuliers.

Intéressons nous maintenant aux langages que l'on peut définir au moyen d'une grammaire de précedence totale.

DEFINITION Tout langage "context-free" qui peut être défini au moyen d'une grammaire de précedence totale sera dit de précedence totale.

En remarquant la symétrie de la définition équivalente d'une grammaire de précedence totale on déduit cette propriété :

THEOREME 2.6.2 Si L est un langage de précedence totale défini sur le vocabulaire  $V_T$ , son réfléchi, qui est le langage  $\hat{L}$  défini par

$$\hat{L} = \{A_1 A_2 \dots A_n \in V_T^* \mid A_1, A_2, \dots, A_n \in V_T \text{ et } A_n A_{n-1} \dots A_1 \in L\}$$

est aussi de précedence totale.

Il est intéressant de comparer l'ensemble des langages de précedence totale à un autre sous-ensemble bien connu de langages "context-free", celui des langages déterministes. Rappelons que ces langages sont ceux que l'on sait reconnaître au moyen d'un automate à une pile (pushdown) déterministe. Nous avons obtenu le résultat suivant :

THEOREME 2.6.3 L'ensemble des langages de précedence totale n'est pas inclus dans l'ensemble des langages déterministes et de leurs réfléchis.

DEMONSTRATION Pour cette démonstration nous utiliserons les résultats suivants démontrés par Ginsburg et Greibach en [GG] :

- (1) Si D désigne un langage déterministe et R un langage régulier, les trois langages

$$D \cap R,$$

$$DR = \{xy \mid x \in D \text{ et } y \in R\},$$

$$D/R = \{x \mid \text{il existe } y \in R \text{ avec } xy \in D\}$$

sont déterministes.

- (2) Le langage  $L_5 = \{a^i b^j a^j \cup a^i b^i a^j \mid i > 0, j > 0\}$  n'est pas déterministe. (C'est d'ailleurs un langage intrinsèquement ambigu c'est-à-dire qui ne peut être défini par une grammaire non ambiguë).

Considérons maintenant le langage

$$L_4 = \{a^i b^j a^j \cup a^i b^i a^j c \mid i > 0, j > 0\}$$

défini par la grammaire de précedence totale  $G_4$  du paragraphe 2.5 et le langage

$$L_6 = \{fd^i e^j d^j \cup d^i e^i d^j \mid i > 0, j > 0\}$$

que l'on peut définir par une grammaire de précedence totale  $G_6$  symétrique de la grammaire  $G_4$ . Le langage  $L_4 \cup L_6$  est de précedence totale. En effet on peut toujours s'arranger pour que les vocabulaires de  $G_4$  et de  $G_6$  soient disjoints et de ce fait il n'y aura pas d'interférence entre les relations de précedence totale de  $G_4$  et celles de  $G_6$  lorsque l'on construira la grammaire définissant le langage  $L_4 \cup L_6$ . Du fait que

$$L_5 = ((L_4 \cup L_6) c^* \cap a^* b^* a^* c) / c$$

le langage  $L_4 \cup L_6$  ne peut être déterministe et du fait de la symétrie de sa définition son réfléchi ne peut l'être non plus.

Nous n'avons malheureusement pas pu trouver une démonstration rigoureuse permettant d'affirmer que l'ensemble des langages de précedence totale n'incluait pas le sous-ensemble de langages déterministes dont les réfléchis sont aussi déterministes.

Cependant il semble bien que le langage déterministe :

$$\{a^i b^i \cup c a^i b^{2i} c \mid i > 0\}$$

dont le réfléchi est aussi déterministe n'est pas de précedence totale.

En terminant ce chapitre, signalons un langage de précedence totale très simple qui ne semble pas être déterministe. Il s'agit du langage dont chaque chaîne est de longueur impaire et est formée d'une suite de "a" et de "b" telle que le symbole "a" placé le plus à droite se trouve au milieu de celle-ci. Ce langage peut être défini par la grammaire de précedence totale :

$S \rightarrow a$   
 $S \rightarrow aSB$   
 $S \rightarrow bSB$   
 $B \rightarrow b$

de laquelle on déduit le tableau suivant en utilisant les formules données au paragraphe 2.5.

	S	B	a	b
S	.	$\bar{=}$	.	$\bar{<}$
B	.	$\bar{>}$	.	$\bar{>}$
a	$\bar{=}$	$\bar{>}$	$\bar{<}$	$\bar{<}$
b	$\bar{=}$	$\bar{>}$	$\bar{<}$	$\bar{<}$

Pour voir de quelle façon l'analyseur procède dans ce cas fournissons lui par exemple la chaîne "ababb" en configuration initiale. On obtient :

(0) #	ababb#	(10) #	[a]b[a] BB#
(1) #	[a babb#	(11) #	[a]b SBB#
(2) #	[a]b abb#	(12) #	[a]bS BB#
(3) #	[a]b[a bb#	(13) #	[a]bSB B#
(4) #	[a]b[a]b b#	(14) #	[a]bSB] B#
(5) #	[a]b[a]b]b #	(15) #	[a SB#
(6) #	[a]b[a]b]b] #	(16) #	[aS B#
(7) #	[a]b[a]b B#	(17) #	[aS]B #
(8) #	[a]b[a]b] B#	(18) #	[aS]B] #
(9) #	[a]b[a BB#	(19) #	S#

CHAPITRE 3 : Précédence de gauche à droite

### 3.1 Analyseurs et relations de précédence de gauche à droite : définition

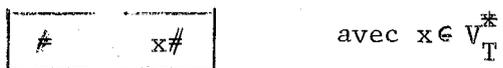
DEFINITION Etant donné une grammaire  $G$  de vocabulaire  $V_T \cup V_N$ , nous appelons opérateurs et opérandes les éléments respectifs de deux vocabulaires disjoints  $V_R$  et  $V_E$  tels que  $V_R \cup V_E = V_T \cup V_N$  et choisis de telle façon que :

- (1) tout symbole terminal soit un opérateur,
- (2) il n'existe pas de règle de  $G$  de la forme  $U \rightarrow xXY$  avec  $x, y \in (V_T \cup V_N)^*$  et  $X, Y$  opérandes,
- (3) il n'existe pas de règles de  $G$  de la forme  $U \rightarrow V$  avec  $V$  opérande.

La condition (3) pourrait aussi s'écrire : tout membre droit de règle contient au moins un opérateur. Bien entendu il est toujours possible de choisir  $V_R$  et  $V_E$  de façon à ce qu'ils satisfassent aux trois restrictions ; il suffit de prendre par exemple  $V_R = V_T \cup V_N$  et  $V_E = \emptyset$ .

Supposons qu'un tel choix ait été fait et considérons maintenant trois relations binaires  $\leq, =, \geq$ , sur  $V_T \cup V_N$  telles que  $\leq U = U \geq \subset V_R \times V_R$ , et l'automate à deux piles de vocabulaire  $V_T \cup V_N \cup \{#, [, ]\}$ , avec  $(V_T \cup V_N) \cap \{#, [, ]\} = \emptyset$ , défini ainsi :

Configuration initiale des deux piles



Configuration finale des deux piles



Instructions

Pour tout  $A, B \in V_R \cup \{#\}$ , tout  $E \in V_E \cup \{\Lambda\}$  et toute règle  $U \rightarrow u$  de  $G$  on a :

- (1)  $(A, EB) \rightarrow (A[EB, \wedge)$  si et seulement si  $A < B$  ou  $A = \#$  et  $B \neq \#$
- (2)  $(A, EB) \rightarrow (AEB, \wedge)$  si et seulement si  $A = B$
- (3)  $(A, EB) \rightarrow (AE], B)$  si et seulement si  $A > B$  ou  $A \neq \#$  et  $B = \#$
- (4)  $([u], B) \rightarrow (\wedge, UB)$  si et seulement si  $U \in V_E$  ou  $U \in V_R$  et  $B \in V_T \cup \{\#\}$

DEFINITION Si l'automate précédemment décrit est un analyseur pour la grammaire  $G$ , alors cet analyseur ainsi que les relations associées  $\langle, \bar{=}, \rangle$  seront dits de précedence de gauche à droite.

Comme au paragraphe 2.1 nous considérons que cet automate est un analyseur pour la grammaire  $G$  si à chaque structure d'une phrase terminale  $x$  qui lui est fournie en configuration initiale correspond au moins un cheminement se terminant sur la configuration finale. Il faut remarquer que la forme particulière des instructions du type (4) impose d'appliquer les règles dans un ordre bien précis : entre autre la pile droite ne peut jamais contenir plus de deux symboles non terminaux.

Donnons un exemple d'analyseur de précedence de gauche à droite : soit la grammaire

$$\begin{array}{ll}
 G_7 : & S \rightarrow a \\
 & S \rightarrow b \\
 & S \rightarrow (SRS) \\
 & R \rightarrow + \\
 & R \rightarrow - \\
 & V_T = \{a, b, +, -\} \\
 & V_N = \{S, R\}
 \end{array}$$

Choisissons les ensembles d'opérateurs et d'opérandes ainsi

$$V_R = \{a, b, +, -, R\} \quad V_E = \{S\}$$

Comme on pourrait le vérifier par la suite les relations  $\langle, \bar{=}, \rangle$  définies par le tableau qui suit sont de précedence de gauche à droite.

	a				
	b	(	)	R	+
ab	.	.	>	>	<
(	<	<	.	=	.
)	.	.	>	>	.
R	<	<	=	.	.
+ -	>	>	.	.	<

Pour analyser la phrase terminale "(a-(b+a))" l'analyseur de précédence de gauche à droite associé à ces relations passe successivement par les configurations :

(0) #	(a-(b+a)) #	(13) #	[(SR[(b Ra)) #
(1) #	[( a-(b+a)) #	(14) #	[(SR[(b Ra)) #
(2) #	[( [a -(b+a)) #	(15) #	[(SR[( SRa)) #
(3) #	[( [a[- (b+a)) #	(16) #	[(SR[(SR a)) #
(4) #	[( [a[-] (b+a)) #	(17) #	[(SR[(SR[a )) #
(5) #	[( [a R(b+a)) #	(18) #	[(SR[(SR[a )) #
(6) #	[( [a R(b+a)) #	(19) #	[(SR[(SR S)) #
(7) #	[( SR(b+a)) #	(20) #	[(SR[(SRS) ) #
(8) #	[( SR (b+a)) #	(21) #	[(SR[(SRS)] ) #
(9) #	[( SR[( b+a)) #	(22) #	[(SR S) #
(10) #	[( SR[(b +a)) #	(23) #	[(SRS) #
(11) #	[( SR[( [b [+ a)) #	(24) #	[(SRS)] #
(12) #	[( SR[( [b [+ a)) #	(25) #	S #

### 3.2 Analyseur de précédence de gauche à droite déterministe

Sur l'exemple précédent on peut remarquer que l'analyseur est déterministe. D'une façon générale en revoyant la définition d'un automate à deux piles déterministe (paragraphe 1.8) et en considérant l'automate décrit au début du paragraphe 3.1 on déduit :

THEOREME 3.2.1 Une condition nécessaire et suffisante pour qu'un analyseur de précedence de gauche à droite défini dans une grammaire  $G$  soit déterministe est que :

- (1) toutes les règles de  $G$  aient des membres droits distincts,
- (2) les relations de précedence de gauche à droite  $<, =, >$  associées à cet analyseur soient incompatibles deux à deux.

Ce résultat est le même que celui concernant les analyseurs de précedence totale déterministes (théorème 2.3.2). En utilisant une démonstration très proche de celle du théorème 2.3.1 qui concerne la rapidité d'un analyseur de précedence totale déterministe, on obtient aussi :

THEOREME 3.2.2 L'analyse d'une chaîne terminale  $t$  par un analyseur de précedence de gauche à droite se fait en un temps au plus proportionnel à la longueur  $|t|$  de cette chaîne. Plus exactement, en désignant par  $N$  le nombre d'instructions exécutées par l'automate et par  $g$  le nombre de règles de la grammaire considérée, on a  $N \leq (6g+1) |t|$ .

Voici maintenant une propriété qui ne concerne que ce type d'analyseur :

THEOREME 3.2.3 Tout langage reconnaissable par un analyseur de précedence de gauche à droite déterministe est aussi reconnaissable par un automate classique à une pile déterministe.

DEMONSTRATION En se reportant à la définition d'un analyseur de précedence de gauche à droite au début du paragraphe 3.1, on peut constater que, du fait des instructions de type (4), la pile droite de l'automate ne contient jamais plus de deux symboles non terminaux. De ce fait il est possible de définir un automate équivalent qui n'empilerait jamais de symboles dans la pile droite ; celle-ci se comporterait donc comme un ruban de lecture. En adjoignant maintenant un nombre fini d'états à ce nouvel automate on peut alors le transformer en un automate qui ne puisse lire qu'un seul symbole à la fois sur le sommet de la pile gauche. Ce dernier automate est donc un automate classique à une pile.

### 3.3 Une définition équivalente des relations de précedence de gauche à droite

Considérons une grammaire  $G$  de vocabulaire  $V_T \cup V_N$  dans laquelle on a choisi des ensembles d'opérateurs et d'opérandes  $V_R$  et  $V_E$  conformément à la définition du début du paragraphe 3.1. Il est possible de donner une définition équivalente de relations de précedence de gauche à droite qui ne fait plus intervenir d'automate. Pour ceci nous aurons besoin de définir une position privilégiée  $h(x)$  de chaque chaîne  $x$ .

**DEFINITION** Pour toute chaîne  $x \in (V_T \cup V_N)^*$  qui contient au moins un symbole non terminal,  $h(x)$  désignera la position du symbole qui, parmi les symboles non terminaux de  $x$  non immédiatement précédés d'un opérande, se trouve placé le plus à droite.

Il faut remarquer que la position  $h(x)$  est toujours définie à partir du moment où  $x$  contient au moins un symbole non terminal. A partir de  $h$  il nous est alors possible de définir la relation  $\Rightarrow$  qui sera une restriction de la relation  $\Rightarrow$  :

**DEFINITION** Pour tout couple de chaîne  $x, y \in (V_T \cup V_N)^*$  :

$x \Rightarrow y \equiv [(x; h(x), y)$  est une dérivation dans la grammaire  $\bar{G}$ ]

Au prochain paragraphe nous verrons quelques propriétés de la relation  $\Rightarrow$ , cependant dès maintenant nous disposons de suffisamment d'éléments pour donner une définition équivalente des relations de précedence de gauche à droite.

**DEFINITION EQUIVALENTE** Trois relations binaires  $\leq, \bar{=}, \geq$  sur  $V_T \cup V_N$  telles que  $\leq \cup \bar{=} \cup \geq \subset V_R \times V_R$  seront dites de précedence de gauche à droite si et seulement si pour tout  $A, B \in V_R$ ,  $U \in V_N$ ,  $E \in V_E \cup \{\Lambda\}$ ,  $x, y, u \in (V_T \cup V_N)^*$  on a :

- (1)  $U \rightarrow xAEBy$  entraîne  $A \bar{=} B$ ,

- (2)  $S \xrightarrow{*} xAUy \xrightarrow{*} xAEBuy$  entraîne  $A \leq B$   
 (la transformation de  $xAUy$  en  $xAEBuy$  est supposée faite en appliquant la règle  $U \rightarrow EBu$  à l'élément  $U$ ).
- (3)  $S \xrightarrow{*} xUBy \xrightarrow{*} xuAEBy$  entraîne  $A \geq B$   
 (la transformation de  $xUBy$  en  $xAEBuy$  est supposée faite en appliquant la règle  $U \rightarrow uAE$  à l'élément  $U$ ).

DEMONSTRATION En faisant intervenir les restrictions habituelles sur les grammaires du paragraphe 1.5 et les propriétés des dérivations énoncées au lemme 1.4.1 et au corollaire 1.4.3, on déduit immédiatement que la condition énoncée dans cette définition équivalente peut aussi s'exprimer ainsi :

Quelle que soit la phrase terminale  $t$  et quelle que soit sa structure  $s$ , il existe une dérivation engendrant  $s$  qui est de la forme :

$$(S=r_0; h(r_1), t_1, \dots; h(r_{n-1}), r_n=t)$$

avec pour tout  $i \in \{1, 2, \dots, n\}$ , tout  $U \in V_N$  en position  $h(r_{i-1})$  dans  $r_{i-1}$ , tout  $A, B \in V_R$ , tout  $E \in V_E \cup \{\Lambda\}$  et tout  $x, y, u, v \in (V_T \cup V_N)^*$  :

- (1)  $r_{i-1} = xUy$ ,  $U \rightarrow uAEbv$ ,  $r_i = xuAEbvy$  entraîne  $A = B$ ,
- (2)  $r_{i-1} = xAUy$ ,  $U \rightarrow EBv$ ,  $r_i = xAEBvy$  entraîne  $A \leq B$ ,
- (3)  $r_{i-1} = xUBy$ ,  $U \rightarrow uAE$ ,  $r_i = xuAEBy$  entraîne  $A \geq B$ .

Considérons maintenant trois relations binaires quelconques  $\leq, =, \geq$  sur  $V_T \cup V_N$  et associons leur l'automate du paragraphe 3.1 dans lequel on aurait permuté les configurations initiales avec les configurations finales et le membre droit de chaque instruction avec son membre gauche. On a donc l'automate :

Configuration initiale : 

#	S#
---	----

Configurations finales : 

#	t#
---	----

 avec  $t \in V_t^*$

Instructions : pour tout  $A, B \in V_R \cup \{\#\}$ ,  $E \in V_E \cup \{\Lambda\}$  et toute règle  $U \rightarrow u$  :

- (4)  $(\Lambda, UB) \rightarrow ([u], B)$  si et seulement si  $U \in V_E$  ou  $V \in V_R$  et  $B \in V_T \cup \{\#\}$
- (3)  $(AE \ ] , B) \rightarrow (A, EB)$  si et seulement si  $A > B$  ou  $A \neq \#$  et  $B = \#$
- (2)  $(AEB, \Lambda) \rightarrow (A, EB)$  si et seulement si  $A = B$
- (1)  $(A[EB, \Lambda) \rightarrow (A, EB)$  si et seulement si  $A < B$  ou  $A = \#$  et  $B \neq \#$

De cet automate on déduit que la condition énoncée précédemment est une condition nécessaire et suffisante pour que les relations  $<, =, >$  soient de précédence de gauche à droite.

### 3.4 Relation $\sigma$

Sur  $V_T \cup V_N$  définissons la relation binaire  $\sigma$  ainsi :

DEFINITION Pour tout couple  $A, B \in V_T \cup V_N$  :

$$A \sigma B \equiv [\text{il existe } x, y \in (V_T \cup V_N)^* \text{ tels que } S \xrightarrow{*} xAB y]$$

Nous allons essayer d'évaluer  $\sigma$  en fonction des relations  $\alpha, \gamma, \delta$  du paragraphe 1.6. Pour ceci nous aurons besoin de deux lemmes.

LEMME 3.4.1 Si  $A, B \in V_T \cup V_N$  et  $A \sigma B$  alors  $A \in V_R$  ou  $B \in V_R$

DEMONSTRATION Si  $A \sigma B$  alors il existe un entier  $n$  et deux chaînes  $x, y \in (V_T \cup V_N)^*$  tels que  $S \xrightarrow{n} xAB y$ . Raisonnons par récurrence sur  $n$ . Si  $n=1$  le théorème est vrai car il ne peut y avoir deux opérandes adjacents dans une règle de la grammaire (début du paragraphe 3.1). Supposons le théorème vrai pour  $n$  et démontrons qu'il est vrai pour  $n+1$ . Si  $S \xrightarrow{n+1} xAB y$  alors il existe  $s \in (V_T \cup V_N)^*$  tel que  $S \xrightarrow{n} s \xrightarrow{U} xAB y$ . Soit  $U \rightarrow u$  la règle utilisée pour transformer  $s$  en  $xAB y$ . Dans la phrase  $s$  il ne peut y avoir d'opérande adjacent au symbole  $U$  et ceci du fait que  $s$  ne contient pas deux opérandes adjacents par supposition et du fait de la définition de la relation  $\xrightarrow{n}$ . La chaîne  $u$  ne contient pas deux opérandes adjacents. Lors de la transformation de  $s$  en  $xAB y$  il ne peut donc se former un couple d'opérandes adjacents.

LEMME 3.4.2 Si  $S \vdash^* xABy$  et  $AB \vdash u$  avec  $A, B \in V_T \cup V_N$  et  $u, x, y \in (V_T \cup V_N)^*$  alors il existe  $x', y' \in (V_T \cup V_N)^*$  tels que  $S \vdash^* x'AB'y' \vdash x'uy'$ , l'élément auquel est appliquée la règle pour transformer  $x'AB'y'$  en  $x'uy'$  étant soit A soit B.

DEMONSTRATION D'après la restriction (4) du paragraphe 1.5 il existe  $t \in V_T^*$  tel que  $xABy \vdash^* t$  et d'après le corollaire 1.4.3  $xABy \vdash^* t$ . Il existe donc  $s_0, s_1, \dots, s_n \in (V_T \cup V_N)^*$  tels que :

$$xABy = s_0 \vdash s_1 \vdash \dots \vdash s_n = t$$

Soit  $i$  le plus grand entier tel que  $i \leq n$  et tel que dans la chaîne  $s_i$  on n'ait appliqué aucune règle à l'élément A et à l'élément B. On a donc

$$S \vdash^* s_i = x'AB'y' \vdash x'vy' = s_{i+1} \text{ avec } x', y', v \in (V_T \cup V_N)^*$$

L'élément auquel est appliqué la règle pour transformer  $x'AB'y'$  en  $x'vy'$  étant soit A soit B. En se reportant à la définition de la relation  $\vdash$  on conclut que si  $AB \vdash u$  alors

$$S \vdash^* x'AB'y' \vdash xuy$$

et que l'élément auquel est appliqué la règle pour transformer  $x'AB'y'$  en  $xuy$  est le même que celui pour transformer  $x'AB'y'$  en  $xvy$ .

En plus des deux lemmes précédents nous aurons besoin de quatre relations binaires sur  $V_T \cup V_N$  notées  $\tau$  pour "terminal",  $\nu$  pour "non terminal",  $\rho$  pour "opérateur",  $\xi$  pour "opérande".

DEFINITION Pour tout couple  $A, B \in V_T \cup V_N$  :

$$[A\tau B \equiv A=B \text{ et } A, B \in V_T]$$

$$[A\nu B \equiv A=B \text{ et } A, B \in V_N]$$

$$[A\rho B \equiv A=B \text{ et } A, B \in V_R]$$

$$[A\xi B \equiv A=B \text{ et } A, B \in V_E]$$

Moyennant ces éléments on a :

$$\text{THEOREME 3.4.3} \quad \sigma = \delta^* \rho(\delta \varepsilon)^* \alpha \gamma^* \tau \cup \rho(\delta \varepsilon)^* \alpha \gamma^* \cup \varepsilon(\delta \varepsilon)^* \alpha$$

DEMONSTRATION Démontrons tout d'abord les cinq points suivants :

- (1)  $\alpha \subset \sigma$
- (2)  $\delta \varepsilon \sigma \subset \sigma$
- (3)  $\delta \sigma \tau \subset \sigma$
- (4)  $\rho \sigma \gamma \subset \sigma$
- (5)  $\sigma$  ne contient pas d'autres éléments que ceux énumérés en (1), (2), (3) et (4).

Point (1) : Considérons deux symboles  $A, B \in V_T \cup V_N$  tels que  $A \alpha B$ . Il existe donc une règle de la forme  $Y \rightarrow uABv$  avec  $Y \in V_N$  et  $u, v \in (V_T \cup V_N)^*$ . D'après les restrictions (3) et (4) du paragraphe 1.5 il existe  $t \in V_T^*$  tel que  $S \Rightarrow^* t$  et tel que la règle  $Y \rightarrow uABv$  soit utilisée pour construire  $t$  à partir de  $S$ . D'après le corollaire 1.4.3 et le lemme 1.4.1 on en conclut qu'il existe  $x, y \in (V_T \cup V_N)^*$  avec  $S \Rightarrow^* xYy \Rightarrow^* xuABvy \Rightarrow^* t$ .

Points (2), (3) et (4) : Ils découlent immédiatement des lemmes 3.4.1 et 3.4.2.

Points (5) : Il découle immédiatement de la définition de la relation  $\Rightarrow$  et du lemme 3.4.1.

Des points (1), (2), (3), (4), (5) et du fait que  $\gamma \tau = \tau \gamma = \rho = \emptyset$  on en déduit que :

$$\sigma = \bigcup_{i=0}^{n-1} \bigcup_{j=0}^{n-1} \bigcup_{k=0}^{n-1} \delta^k \rho^j (\delta \varepsilon)^i \alpha \gamma^i \tau^j \sigma^k$$

En sommant par rapport à  $k$  :

$$\sigma = \bigcup_{i=0}^{n-1} \bigcup_{j=0}^{n-1} (\delta^+ \rho^j (\delta \varepsilon)^i \alpha \gamma^i \tau^j \cup \rho^j (\delta \varepsilon)^i \alpha \gamma^i \tau^j)$$

En sommant par rapport à  $j$  :

$$o = \bigcup_{i=0}^{n-i} (\delta^+ \rho(\delta) \alpha_\gamma^+ \tau \cup \delta^+ \alpha \tau \cup \rho(\delta) \alpha_\gamma^+ \cup (\delta) \alpha^i)$$

En sommant par rapport à  $i$  :

$$\sigma = \delta^+ \rho(\delta \varepsilon) \alpha_\gamma^+ \tau \cup \delta^+ \alpha \tau \cup \rho(\delta \varepsilon) \alpha_\gamma^+ \cup (\delta \varepsilon) \alpha \cup \alpha$$

Remarquons que :

$$\delta^+ = \delta^+ \rho(\delta \varepsilon) \cup (\delta \varepsilon)^+ \quad \text{d'où} \quad \delta^+ \alpha \tau = \delta^+ \rho(\delta \varepsilon) \alpha \tau \cup (\delta \varepsilon)^+ \alpha \tau$$

On a donc :

$$\sigma = \delta^+ \rho(\delta \varepsilon) \alpha_\gamma^* \tau \cup \rho(\delta \varepsilon) \alpha_\gamma^+ \cup (\delta \varepsilon) \alpha.$$

Du fait que  $(\delta \varepsilon) \alpha = \rho(\delta \varepsilon) \alpha \cup \varepsilon(\delta \varepsilon) \alpha$  :

$$\sigma = \delta^+ \rho(\delta \varepsilon) \alpha_\gamma^* \tau \cup \rho(\delta \varepsilon) \alpha_\gamma^* \cup \varepsilon(\delta \varepsilon) \alpha$$

Et du fait que  $\rho(\delta \varepsilon) \alpha_\gamma^* = \rho(\delta \varepsilon) \alpha_\gamma^* \tau \cup \rho(\delta \varepsilon) \alpha_\gamma^*$

$$\sigma = \delta^+ \rho(\delta \varepsilon) \alpha_\gamma^* \tau \cup \rho(\delta \varepsilon) \alpha_\gamma^* \cup \varepsilon(\delta \varepsilon) \alpha$$

### 3.5 Plus petites relations de précédence de gauche à droite

En nous servant des résultats précédents il nous est maintenant possible d'exprimer des relations de précédence de gauche à droite en fonction de relations plus simples. En plus des relations  $\alpha, \gamma, \delta$  et  $\tau, \nu, \rho, \xi$  nous aurons besoin de trois autres relations binaires sur  $V_T \cup V_N$  notées  $\underline{\alpha}, \underline{\gamma}, \underline{\delta}$  par analogie avec  $\alpha, \gamma, \delta$ .

DEFINITION Pour tout couple  $A, B \in V_T \cup V_N$  :

$A \underline{\alpha} B \equiv [A, B \in V_R \text{ et il existe } U \in V_N, E \in V_R \cup \{\Lambda\}, x, y \in (V_T \cup V_N)^* \text{ tels que } U \rightarrow xAEBy]$

$A \underline{\gamma} B \equiv [B \in V_R \text{ et il existe } E \in V_E \cup \{\Lambda\}, y \in (V_T \cup V_N)^* \text{ tels que } A \rightarrow EBy]$

$A \underline{\delta} B \equiv [A \in V_R \text{ et il existe } E \in V_E \cup \{\Lambda\}, x \in (V_T \cup V_N)^* \text{ tels que } B \rightarrow xAE]$

Nous avons alors la propriété suivante :

LEMME 3.5.1 Une condition nécessaire et suffisante pour que trois relations binaires  $\underline{\alpha}, \underline{\gamma}, \underline{\delta}$  sur  $V_T \cup V_N$  soient de précedence de gauche à droite est que

- (1)  $\underline{\alpha} \subset \underline{\gamma}$
- (2)  $\rho(\delta \underline{\alpha})^* \underline{\alpha} \gamma^* \underline{\gamma} \subset \underline{\alpha}$
- (3)  $(\delta \underline{\alpha} (\delta \underline{\alpha})^* \underline{\alpha} \cup \underline{\delta} \delta^* \rho(\delta \underline{\alpha})^* \underline{\alpha} \gamma^* \tau) \subset \underline{\delta}$

DEMONSTRATION Remarquons que les points (2) et (3) de ce lemme sont équivalents aux points (2 bis) et (3 bis) qui suivent et ceci du fait du théorème 3.4.3

$$(2\text{bis}) \quad \rho \sigma \underline{\gamma} \subset \underline{\alpha}$$

$$(3\text{bis}) \quad (\delta \underline{\alpha} \sigma \cup \underline{\delta} \sigma \tau) \subset \underline{\delta}$$

Pour démontrer le théorème il suffit donc de se reporter à la définition équivalente du paragraphe 3.3 et d'utiliser le lemme 3.4.2.

Introduisons maintenant le concept de plus petites relations de précedence de gauche à droite.

DEFINITION Etant donné une grammaire dans laquelle on a choisi des opérateurs et des opérands, nous dirons que les relations de précedence de gauche à droite  $\underline{\alpha}, \underline{\gamma}, \underline{\delta}$  sont les plus petites si et seulement pour toutes autres relations de précedence de gauche à droite  $\underline{\alpha}', \underline{\gamma}', \underline{\delta}'$  dans la même grammaire avec les mêmes

opérandes on a toujours  $\prec \subset \prec'$ ,  $\equiv \subset \equiv'$ ,  $\succ \subset \succ'$

Du lemme précédent il découle

COROLLAIRE 3.5.2 Quel que soit la grammaire et quel que soit le choix des opérateurs et des opérandes, il existe toujours de plus petites relations de précedence de gauche à droite  $\prec, \equiv, \succ$  qui sont :

$$\begin{aligned} \equiv &= \underline{\alpha} \\ \prec &= \rho(\delta\varepsilon)^* \underline{\alpha} \gamma^* \underline{\gamma} \\ \succ &= \underline{\delta} \varepsilon (\delta\varepsilon)^* \underline{\alpha} \cup \underline{\delta} \delta^* \rho(\delta\varepsilon)^* \underline{\alpha} \gamma^* \underline{\tau} \end{aligned}$$

Rappelons que les relations  $\underline{\alpha}, \underline{\gamma}, \underline{\delta}$ , sont définies au début de ce paragraphe, les relations  $\varepsilon, \rho, \tau$  au paragraphe 3.3 et les relations  $\alpha, \gamma, \delta$ , au paragraphe 4.6.

### 3.6 Une variante plus générale de l'analyseur de précedence de gauche à droite

Nous avons vu, au début du paragraphe 3.1, que le choix d'opérateurs et d'opérandes dans une grammaire  $G$  devait entre autres satisfaire à la condition (3) qui interdit d'avoir dans  $G$  des règles de la forme  $U \rightarrow V$  avec  $V$  opérande. Cette condition restreint beaucoup les choix possibles. Nous nous proposons donc de la remplacer par une condition (3 bis) moins forte :

DEFINITION ETENDUE Nous appellerons opérateurs et opérandes les éléments respectifs de deux vocabulaires disjoints  $V_R$  et  $V_E$  tels que  $V_R \cup V_E = V_T \cup V_N$  et choisis de telle façon que :

- (1) tout symbole terminal soit un opérateur,
- (2) il n'existe pas de règle de la forme  $U \rightarrow xXYy$  avec  $x, y \in (V_T \cup V_N)^*$  et  $X, Y$  opérandes,
- (3bis) il n'existe pas de règle de la forme  $U \rightarrow V$  avec  $U$  opérateur et  $V$  opérande.

Considérons une grammaire  $G$  de vocabulaire  $V_T \cup V_N$  dans laquelle on a choisi des ensembles  $V_R$  et  $V_E$  conformément à la définition étendue. A cette grammaire faisons correspondre la grammaire  $G'$  de vocabulaire opérande et opérateur  $V'_R, V'_E$  ainsi définie :

- $V'_R = V_R$
- $V'_E = \{E \in V_E \mid \text{il existe } R \in V_R \text{ et } x, y \in (V_T \cup V_N)^* \text{ tel que } E \rightarrow xRy \text{ dans } G\}$
- toute règle de  $G'$  est de la forme  $X \rightarrow E'_0 R_1 E'_1 \dots R_n E'_n$  avec  $n > 0$ ,  $E'_i \in V'_E \cup \{\Lambda\}$  et  $R_i \in V'_R$  et on a

$$X \rightarrow E'_0 R_1 E'_1 \dots R_n E'_n \text{ dans } G' \equiv \left[ \begin{array}{l} \text{il existe } E_0, E_1, \dots, E_n \in V_E \cup \{\Lambda\} \\ \text{avec } n > 0, X \rightarrow E_0 R_1 E_1 \dots R_n E_n \\ \text{dans } G \text{ et pour tout } i \in \{0, 1, \dots, n\} \\ E_i \Rightarrow^* E'_i. \end{array} \right]$$

- si l'axiome  $S$  de  $G$  est un opérateur alors  $G'$  aura le même axiome, sinon la grammaire  $G'$  aura pour axiomes l'ensemble des  $S_1, S_2, \dots, S_t \in V'_E$  tels que  $S \Rightarrow^* S_i$  pour tout  $i \in \{1, 0, \dots, t\}$ . Il va de soi que tous les résultats antérieurs se généralisent à des grammaires à plusieurs axiomes.

On peut alors remarquer trois choses :

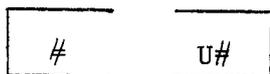
- La grammaire  $G$  et la grammaire  $G'$  définissent le même langage.
- Les vocabulaires  $V'_R$  et  $V'_E$  de  $G'$  sont choisis d'une façon conforme à la définition non étendue d'opérandes et d'opérateurs (paragraphe 3.1).
- Si l'on désigne respectivement par  $\langle, \bar{=}, \rangle$  et  $\langle', \bar{=}', \rangle'$  les relations du corollaire 3.5.2 associées aux grammaires  $G$  et  $G'$ , pour tout couple d'opérateurs  $A, B$  on a  $A \langle B \equiv A \langle' B$ ,  $A \bar{=} B \equiv A \bar{=} B$ ,  $A \rangle B \equiv A \rangle' B$ .

Aux relations  $\langle, \bar{=}, \rangle$  définies dans la grammaire  $G$  au corollaire 3.5.2 on peut donc associer l'analyseur de précedence de gauche à droite dans  $G'$  qui est :

## Configurations initiales

avec  $x \in V_T^*$ 

## Configurations finales

avec si  $S \in V_R$ ,  $U=S$  et, si  $S \in V_E$ ,  $U \in V_E$  et  $S \Rightarrow^* U$   
Instructions [\*]

Pour tout  $A, B \in V_R \cup \{\#\}$ , tout  $E \in V_E \cup \{\Lambda\}$ , toute règle de  $G$  de la forme  $U \rightarrow E_0 R_1 E_1 \dots R_n E_n$  avec  $n > 0$ ,  $E_0, E_1, \dots, E_n \in V_E \cup \{\Lambda\}$ ,  $R_1, R_2, \dots, R_n \in V_R$  et tout  $E_0, E_1, \dots, E_n \in V_E \cup \{\Lambda\}$  tels que  $E_i \Rightarrow^* E'_i$  ( $0 \leq i \leq n$ ) on a :

- (1)  $(A, EB) \rightarrow (A[EB, \Lambda])$  si et seulement si  $A < B$  ou  $A = \#$  et  $B \neq \#$
- (2)  $(A, EB) \rightarrow (AEB, \Lambda)$  si et seulement si  $A = B$
- (3)  $(A, EB) \rightarrow (AE], B)$  si et seulement si  $A > B$  ou  $A = =$  et  $B = =$
- (4)  $(E'_0 R_1 E'_1 \dots R_n E'_n], B) \rightarrow (\Lambda, UB)$  si et seulement si  $U \in V_E$  ou  $U \in V_R$  et  $B \in V_T \cup \{\#\}$

Cet analyseur constitue donc une variante plus générale de l'analyseur de précedence de gauche à droite du fait que le choix des opérateurs et des opérandes n'a besoin que d'être conforme à la définition étendue d'opérateurs et d'opérandes. Comme nous l'avons déjà mentionné les formules des relations  $<, =, >$  sont les mêmes que celles concernant l'analyseur de précedence de gauche à droite classique, c'est-à-dire que :

$$\begin{aligned}
 = &= \underline{\alpha} \\
 < &= \rho(\delta\varepsilon)^* \gamma \underline{\gamma} \\
 > &= \underline{\delta\varepsilon}(\delta\varepsilon)^* \underline{\alpha} \cup \underline{\delta\delta}^* \rho(\delta\varepsilon)^* \underline{\alpha} \gamma^* \tau
 \end{aligned}$$

---

[\*] En fait il faudrait raisonner sur  $V_E'$  et non sur  $V_E$  mais ceci ne fait qu'introduire des instructions parasites qui ne gênent en rien le fonctionnement de l'automate.

Considérons par exemple la grammaire suivante qui définit des expressions arithmétiques :

$G_8$  :       $A \rightarrow PB$   
           $A \rightarrow APB$        $V_T = \{a, +, -, *, /, (, )\}$   
           $A \rightarrow B$   
           $B \rightarrow BMC$        $V_N = \{A, B, C, P, M\}$   
           $B \rightarrow C$   
           $C \rightarrow (A)$       axiome : A (exceptionnellement nous ne l'avons pas noté S)  
           $C \rightarrow a$   
           $P \rightarrow +$   
           $P \rightarrow -$   
           $M \rightarrow *$   
           $M \rightarrow /$

Conformément à la définition étendue on peut choisir les opérateurs et les opérands ainsi :

$$V_R = \{a, +, -, *, /, (, ), P, M\} \quad V_E = \{A, B, C\}$$

En utilisant les formules précédentes on obtient

	a	$\begin{matrix} + \\ - \end{matrix}$	$\begin{matrix} * \\ / \end{matrix}$	( )	P	M
a	.	<	<	.	>	>
+ -	>	.	.	>	.	.
* /	>	.	.	>	.	.
(	<	.	.	<	=	<
)	.	<	<	.	>	>
P	<	.	.	<	>	<
M	<	.	.	<	>	>

Le tableau suivant représente la relation  $\Rightarrow^*$  entre deux opérandes :

	A	B	C
A	1	1	1
B	.	1	1
C	.	.	1

L'analyse de la chaîne "a+a\*(-a)" se fera donc ainsi :

```

#      a+a*(-a)#
# [a      +a*(-a)#
# [a [+      a*(-a)#
# [a [+      a*(-a)#
# [a      Pa*(-a)#
# [a      Pa*(-a)#
#      CPa*(-a)#
# [CP      a*(-a)#
# [CP [a      *(-a)#
# [CP [a [*      (-a)#
# [CP [a [*      (-a)#
# [CP [a      M(-a)#
# [CP [a      M(-a)#
# [CP      CM(-a)#
# [CP [CM      (-a)#
# [CP [CM [(      -a)#
# [CP [CM [( [-      a)#
# [CP [CM [( [-      a)#
# [CP [CM [(      Pa)#
# [CP [CM [( [P      a)#
# [CP [CM [( [P [a      )#
# [CP [CM [( [P [a      )#
# [CP [CM [( [P      C)#
# [CP [CM [( [PC      )#
# [CP [CM [(      A)#
# [CP [CM [(A      #
# [CP [CM [(A      #
# [CP [CM      C#
# [CP [CM      #
# [CP      B#
# [CPB      #
#      A#

```

### 3.7 Cas particulier : Relations de précedence terminale

Soit  $G$  une grammaire de vocabulaire  $V_T \cup V_N$  qui ne comporte aucune règle de la forme  $U \rightarrow xXYy$  avec  $x, y \in (V_T \cup V_N)^*$  et  $X, Y \in V_N$ . Conformément à la définition étendue d'opérateurs et d'opérandes du paragraphe précédent on peut prendre  $V_R = V_T$  et  $V_E = V_N$  c'est-à-dire que l'on peut considérer que les symboles terminaux sont des opérateurs et les symboles non terminaux des opérandes. En se reportant aux formules du corollaire 3.5.2 et en tenant compte du fait que  $\alpha = \tau \alpha \cup \alpha \tau$ ,  $\rho = \tau$ ,  $\varepsilon = \vee$  on obtient alors :

$$\begin{aligned} \bar{=} &= \underline{\alpha} \\ \leq &= \alpha \gamma \underline{*} \underline{\gamma} \\ \geq &= \underline{\delta} \delta \underline{*} \underline{\alpha} \end{aligned}$$

Ces relations correspondent à celles définies par R. Floyd en [F1.1]. Du fait de la symétrie entre les relations  $\leq$  et  $\geq$  et du fait qu'elles ne concernent que les symboles terminaux nous les appellerons relations de précedence terminale. On peut leur associer l'analyseur du paragraphe précédent qui se présente alors sous cette forme très simple :

Configurations initiales

$$\boxed{\#} \quad \boxed{x\#} \quad \text{avec } x \in V_T$$

Configurations finales

$$\boxed{\#} \quad \boxed{U\#} \quad \text{avec } U \in V_N \text{ et } S \Rightarrow^* U$$

Instructions

Pour tout  $A, B \in V_T \cup \{\#\}$ , tout  $E \in V_N \cup \{\Lambda\}$ , toute règle de  $G$  de la forme

$U \xrightarrow{E} R_1 E_1 \dots R_n E_n$  avec  $n > 0$ ,  $E_0, E_1, \dots, E_n \in V_N \cup \{\Lambda\}$ ,  $R_1, R_2, \dots, R_n \in V_T$  et tout

$E'_0, E'_1, \dots, E'_n \in V_N \cup \{\Lambda\}$  tels que  $E_i \Rightarrow^* E'_i$  ( $0 \leq i \leq n$ ) on a :

- (1)  $(A, EB) \rightarrow (A \lfloor EB, \Lambda)$  si et seulement si  $A < B$  ou  $A = \#$  et  $B \neq \#$
- (2)  $(A, EB) \rightarrow (AEB, \Lambda)$  si et seulement si  $A = B$
- (3)  $(A, EB) \rightarrow (AE \rfloor, B)$  si et seulement si  $A > B$  ou  $A \neq \#$  et  $B = \#$
- (4)  $(E'_0 R_1 E'_1 \dots R_n E'_n, \Lambda) \rightarrow (\Lambda, U)$

Considérons la grammaire  $G_8$  de l'exemple précédent, elle peut s'écrire sous une forme équivalente :

$$\begin{array}{ll}
 A \rightarrow +B & \\
 A \rightarrow -B & V_T = \{a, +, -, *, /, (, )\} \\
 A \rightarrow A+B & \\
 A \rightarrow A-B & \\
 A \rightarrow B & V_N = \{A, B, C\} \\
 B \rightarrow B \cdot C & \\
 B \rightarrow B/C & \\
 B \rightarrow C & \text{axiome : } A \\
 C \rightarrow (A) & \\
 C \rightarrow a &
 \end{array}$$

Du fait que dans aucune règle il n'apparaît deux symboles non terminaux adjacents on peut définir des relations de précedence terminale. En utilisant les formules précédentes on obtient :

	a	+	*/	(	)
a	.	>	>	.	>
+-	<	>	<	<	>
*/	<	>	>	<	>
(	<	<	<	<	=
)	.	>	>	.	>

La relation  $\Rightarrow^*$  entre deux symboles non terminaux se résume au tableau

	A	B	C
A	1	1	1
B	.	1	1
C	.	.	1

L'analyse de la chaîne "a+a\*(-a)" se fera ainsi :

```
#      a+a*(-a)#
#[a    +a*(-a)#
#[a]   +a*(-a)#
#      C+a*(-a)#
#[C+   a*(-a)#
#[C+[a  *(-a)#
#[C+[a] *(-a)#
#[C+   C*(-a)#
#[C+[C* (-a)#
#[C+[C*([ -a)#
#[C+[C*([ [- a)#
#[C+[C*([ [-[a )#
#[C+[C*([ [-[a ]#
#[C+[C*([ [- C)#
#[C+[C*([ [-C ]#
#[C+[C*([ A)#
#[C+[C*([A) #
#[C+[C*([A)] #
#[C+[C*C  C #
#[C+[C*C] #
#[C+    B #
#[C+B]  #
#      A #
```

### 3.8 Cas particulier, relations de précedence totale de gauche à droite

Dans une grammaire  $G$  de vocabulaire  $V_T \cup V_N$  on peut toujours considérer que tous les symboles sont des opérateurs c'est-à-dire que  $V_R = V_T \cup V_N$  et  $V_E = \emptyset$ . L'automate se présente alors sous la forme :

Configurations initiales

$\#$        $x\#$        $x \in V_T^*$

Configuration finale

$\#$        $S\#$

Instructions

Pour tout  $A, B \in V_T \cup V_N \cup \{\#\}$  et toute règle  $U \rightarrow u$  de  $G$  on a :

- (1)  $(A, B) \rightarrow (A \lfloor B, \wedge)$       si et seulement si  $A \prec B$  ou  $A = \#$  et  $B \neq \#$
- (2)  $(A, B) \rightarrow (AB, \wedge)$       si et seulement si  $A = B$
- (3)  $(A, B) \rightarrow (A \rfloor B)$       si et seulement si  $A \succ B$  ou  $A \neq \#$  et  $B = \#$
- (4)  $(\lfloor u \rfloor, B) \rightarrow (\wedge, UB)$       si et seulement si  $B \in V_T \cup \{\#\}$

DEFINITION Si l'automate précédemment décrit est un analyseur pour la grammaire  $G$ , alors cet analyseur ainsi que les relations  $\prec, \bar{=}, \succ$  qui lui sont associées seront dits de précedence totale de gauche à droite.

En se reportant au corollaire 3.5.2 et en remarquant que dans ce cas  $\underline{\alpha} = \alpha$ ,  $\underline{\gamma} = \gamma$ ,  $\underline{\delta} = \delta$ ,  $\rho = T \cup \gamma$ ,  $\varepsilon = \emptyset$  on déduit le théorème :

THEOREME 3.8.1 Dans toute grammaire les relations  $\prec, \bar{=}, \succ$  définies comme suit sont les plus petites relations de précedence totale de gauche à droite.

$\bar{=}$	$=$	$\alpha$
$\prec$	$=$	$\alpha \gamma^+$
$\succ$	$=$	$\delta^+ \alpha \gamma^* \tau$

Ces relations correspondent à celles données par C. Pair en [Pa.1]. N. Wirth et H. Weber [WW] ont défini des relations  $\prec, \equiv, \succ$  très voisines de celles de C. Pair. Seule la relation  $\succ$  diffère et peut être exprimée ainsi :

$$\succ = \delta^+ \alpha \gamma^*$$

On a la propriété

LEMME 3.8.2  $\alpha \gamma^* \cap \delta^+ \alpha \gamma^* \neq \emptyset$  entraîne  $\alpha \gamma^* \cap \delta^* \tau \neq \emptyset$

DEMONSTRATION Si  $\alpha \gamma^* \cap \delta^+ \alpha \gamma^* \neq \emptyset$  alors il existe  $A, B \in V_T \cup V_N$  tels que  $A \alpha \gamma^* B$  et  $A \delta^+ \alpha \gamma^* B$ . Du fait de la restriction (4) (paragraphe 1.4) dans une grammaire il existe donc  $b \in V_T$  tel que  $B \gamma^* b$ . Donc  $A \alpha \gamma^* b$  et  $A \delta^+ \alpha \gamma^* b$  d'où  $\alpha \gamma^* \cap \delta^* \tau \neq \emptyset$ .

Du théorème 3.8.1 et du lemme 3.8.2 on déduit que les relations de N. Wirth et H. Weber sont des relations de précédence totale de gauche à droite telles que leur non incompatibilité deux à deux entraîne l'inexistence de relations de précédence totale de gauche à droite incompatibles deux à deux dans la grammaire considérée. De ce fait on a :

COROLLAIRE 3.8.3 Une condition nécessaire et suffisante pour que dans une grammaire il existe des relations de précédence totale de gauche à droite incompatibles deux à deux est que :

$$(\delta^+ \alpha \cap \alpha) \cup (\alpha \gamma^+ \cap \alpha) \cup (\delta^+ \alpha \gamma^+ \cap \alpha) \cup (\delta^+ \alpha \cap \alpha \gamma^+) \cup (\delta^+ \alpha \gamma^+ \cap \alpha \gamma^+) = \emptyset$$

Cette condition peut s'écrire aussi :

$$(\alpha \gamma^+ \cap \alpha) \cup (\delta^+ \alpha \gamma^* \cap \alpha \gamma^*) = \emptyset$$

La condition de ce corollaire est plus forte que la condition d'existence de relations de précédence totale incompatibles deux à deux donnée au théorème 2.4.1. Il faut donc s'attendre à trouver des grammaires pour lesquelles il existe des relations de précédence totale incompatibles deux à deux mais pour lesquelles il n'existe pas de relations de précédence totale de gauche à droite incompatible

deux à deux. La grammaire  $G_6$  du paragraphe 2.6 en est un exemple. Reportons nous aux relations  $\prec, \equiv, \succ$  définies à la fin du paragraphe 2.5.

**COROLLAIRE 3.8.4** Pour toute grammaire dans laquelle il existe des relations de précédence totale de gauche à droite incompatibles deux à deux, les relations de précédence totale  $\prec, \equiv, \succ$  ainsi définies :

$$\equiv = \alpha$$

$$\prec = \alpha\gamma^+$$

$$\succ = \delta^+\alpha \cup (\delta^+\alpha \cap \alpha\gamma^+)$$

sont de précédence totale de gauche à droite.

**DEMONSTRATION** Si dans  $G$  il existe des relations de précédence totale incompatibles deux à deux alors  $\delta^+\alpha\gamma^+ \cap \alpha\gamma^+ = \emptyset$  d'après le corollaire 3.8.3. De ce fait  $\delta^+\alpha\gamma^+ \cap \alpha\gamma^+ = \delta^+\alpha\gamma^+$  et donc  $\succ = \delta^+\alpha\gamma^*$ . D'après le théorème 3.8.1 on en conclut que les relations  $\prec, \equiv, \succ$  sont de précédence totale de gauche à droite.

### 3.9 Langages de précedence totale, de précedence totale de gauche à droite et autres.

Au paragraphe 2.6 nous avons comparé l'ensemble des langages de précedence totale à celui des langages déterministes. Nous allons compléter les résultats obtenus en faisant intervenir d'autres classes de langages.

**DEFINITION** Nous appellerons grammaire de précedence totale de gauche à droite toute grammaire pour laquelle il existe un analyseur de précedence totale de gauche à droite déterministe.

Du théorème 3.2.1 et du corollaire 3.8.3 on déduit immédiatement cette définition équivalente :

**DEFINITION EQUIVALENTE** Une grammaire de précedence totale de gauche à droite est une grammaire qui satisfait aux deux conditions :

(1) Toutes les règles ont des membres droits distincts

$$(2) (\alpha\gamma^+\cap\alpha)\cup(\delta^+\alpha\gamma^*\cap\alpha\gamma^*) = \emptyset$$

Rappelons que la condition (2) peut aussi s'écrire :

$$(\delta^+\alpha\cap\alpha)\cup(\alpha\gamma^+\cap\alpha)\cup(\delta^+\alpha\gamma^+\cap\alpha)\cup(\delta^+\alpha\cap\alpha\gamma^+)\cup(\delta^+\alpha\gamma^+\cap\alpha\gamma^+) = \emptyset$$

et que de ce fait les grammaires de précedence totale de gauche à droite sont bien un cas particulier des grammaires de précedence totale. Passons maintenant aux langages définis par de telles grammaires :

**DEFINITION** Tout langage "context free" qui peut être défini au moyen d'une grammaire de précedence totale de gauche à droite sera dit de précedence totale de gauche à droite.

Comparons la classe de ces langages avec celles des langages réguliers (états finis).

**THEOREME 3.9.1** L'ensemble des langages réguliers est strictement inclus dans le sous-ensemble des langages de précedence totale de gauche à droite dont les réfléchis sont aussi de précedence totale de gauche à droite.

**DEMONSTRATION** A tout langage régulier  $R$  défini sur un vocabulaire  $V_T$  on sait associer un automate d'état fini déterministe capable de l'analyser. Soit  $Q$  l'ensemble de ses états. Considérons la grammaire  $G$  de vocabulaire  $V_T \cup V_N$  ainsi définie :

-  $V_N = QU\{S\}$  (on s'arrange de façon à ce que  $S \notin Q$ )

- A chaque instruction de l'automate de la forme

$aE \rightarrow F$  avec  $a \in V_T$  et  $E, F \in Q$

on fait correspondre la règle de  $G$

$F \rightarrow a$  si  $E$  est l'état initial

$F \rightarrow Ea$  si  $E$  n'est pas l'état initial

- A chaque état final  $E$  de l'automate, on fait correspondre la règle de  $G$

$S \rightarrow E$

Le langage défini par la grammaire  $G$  est  $R$ . De plus du fait que l'automate était déterministe, toutes les règles de  $G$  ont des membres droits distincts. Si l'on regarde la forme des règles de  $G$  on s'aperçoit que  $\alpha\gamma^+ = \emptyset$ ,  $\delta^+\alpha\gamma^* = \tau\delta^+\alpha\gamma^*$ ,  $\alpha\gamma^* = \nu\alpha\gamma^*$ . De ce fait  $(\alpha\gamma^+ \cap \alpha) \cup (\delta^+\alpha\gamma^* \cap \alpha\gamma^*) = \emptyset$  et  $R$  est un langage de précedence totale de gauche à droite. Tout langage régulier  $R$  est donc de précedence totale de gauche à droite et du fait que le réfléchi d'un langage régulier est toujours régulier,  $R$  est aussi le réfléchi d'un langage de précedence totale de gauche à droite. Il ne reste plus qu'à démontrer que l'inclusion est stricte. Pour ceci considérons la grammaire

$S \rightarrow ab$

$S \rightarrow aSb$

Le langage ainsi défini ainsi que son réfléchi n'est pas régulier mais est de précedence totale de gauche à droite.

Comparons maintenant l'ensemble des langages de précedence totale de gauche à droite avec l'ensemble des langages déterministes.

**THEOREME 3.9.2** L'ensemble des langages de précedence totale de gauche à droite est inclus dans l'ensemble des langages déterministes mais n'est pas inclus dans l'ensemble de leurs réfléchis.

**DEMONSTRATION** La première partie du théorème se démontre en particulierisant le théorème 3.2.3 aux analyseurs de précedence totale de gauche à droite. Pour démontrer la deuxième partie il suffit de trouver un exemple de réfléchi de langage de précedence totale de gauche à droite qui ne soit pas déterministe. Considérons le langage

$$L_4 = \{a^i b^j a^j \cup a^i b^i a^j c \mid i > 0, j > 0\}$$

défini par la grammaire  $G_4$  du paragraphe 2.5. Il est facile de vérifier que  $\tilde{L}_4$  est de précedence totale de gauche à droite. Pour ceci il suffit de définir  $\tilde{L}_4$  par une grammaire symétrique de  $G_4$ . En se reportant à la démonstration de théorème 2.6.3 et en remarquant que

$$\{a^i b^j a^j \cup a^i b^i a^j \mid i > 0, j > 0\} = (L_4 c^* \cap a^* b^* a^* c) / c$$

on conclut que  $L_4$  n'est pas déterministe.

Pour terminer ce chapitre récapitulons les imbrications des différentes classes de langages. Chaque classe sera désignée par une lettre.

- R : ensemble des langages réguliers (états finis)
- D : ensemble des langages déterministes
- $\tilde{D}$  : ensemble des réfléchis des langages déterministes
- $P_t$  : ensemble des langages de précedence totale

$P_{\text{tgd}}$  : ensemble des langages de précédence totale de gauche à droite  
 $\tilde{P}_{\text{tgd}}$  : ensemble des réfléchis des langages de précédence totale de gauche à droite.

D'après le paragraphe 2.6 :

$$P_t \not\subset D \cup \tilde{D} \quad \text{et} \quad D \cap \tilde{D} \not\subset P_t$$

Les langages de précédence totale de gauche à droite sont un cas particulier des langages de précédences totale et du fait que tout réfléchi d'un langage de précédence totale est encore un langage de précédence totale (théorème 2.6.2) :

$$P_{\text{tgd}} \cup \tilde{P}_{\text{tgd}} \subset P_t$$

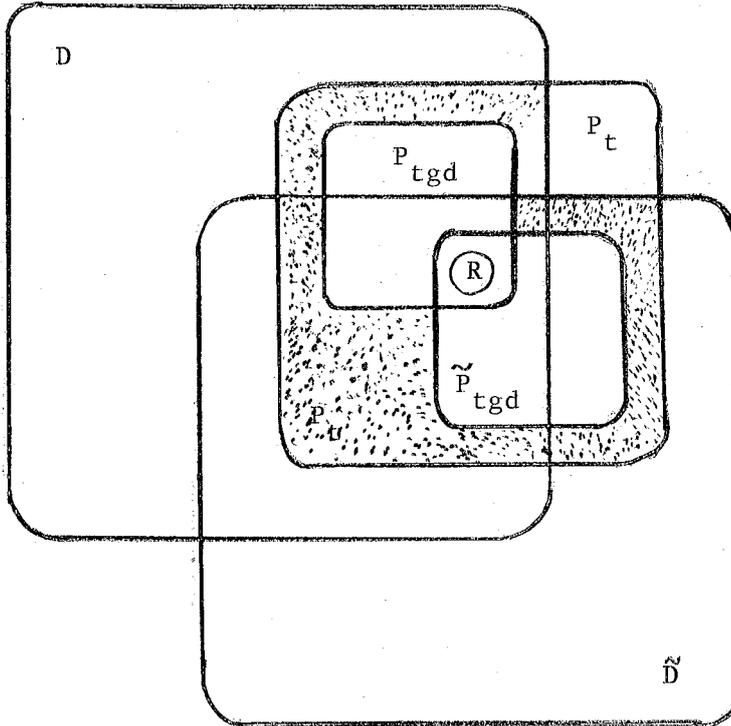
Nous venons de voir (théorème 3.9.2) que :

$$P_{\text{tgd}} \subset D \quad \text{et} \quad P_{\text{tgd}} \not\subset \tilde{D} \quad \text{d'où} \quad \tilde{P}_{\text{tgd}} \subset \tilde{D} \quad \text{et} \quad \tilde{P}_{\text{tgd}} \not\subset D$$

Du fait du théorème 3.9.1 :

$$R \subset P_{\text{tgd}} \cap \tilde{P}_{\text{tgd}} \quad \text{et} \quad R \neq P_{\text{tgd}} \cap \tilde{P}_{\text{tgd}}$$

Nous pouvons donc tracer le schéma :



Nous ignorons si la zone marquée de points existe ou n'existe pas. A remarquer que si cette zone était vide, l'intersection d'un langage de précedence totale avec un langage déterministe serait toujours un langage de précedence totale de gauche à droite.



#### CHAPITRE 4 : Programmation des différents algorithmes

#### 4.1 Calcul de matrices booléennes représentant des relations de précedence

Au cours des deux chapitres précédents nous avons vu que les relations de précédences pouvaient s'exprimer en fonction de relations simples. Nous avons entre autres obtenu les résultats suivants :

Relations de précedence totale (paragraphe 2.5)

$$\begin{aligned} \cdot &= \alpha \\ \cdot & \\ \cdot < &= \alpha \gamma^+ \\ \cdot > &= \delta^+ \alpha \cup (\delta^+ \alpha \gamma^+ \cap \overline{\alpha \gamma^+}) \end{aligned}$$

Relations de précedence de gauche à droite (paragraphe 3.5)

$$\begin{aligned} \cdot &= \underline{\alpha} \\ \cdot < &= \rho(\delta \varepsilon)^* \alpha \gamma^* \underline{\gamma} \\ \cdot > &= \underline{\delta} \cup (\delta)^* \cup \underline{\delta} \delta^* \rho(\delta \varepsilon)^* \alpha \gamma^* \tau \end{aligned}$$

Relations de précedence terminale (paragraphe 3.7)

$$\begin{aligned} \cdot &= \underline{\alpha} \\ \cdot < &= \alpha \gamma^* \underline{\gamma} \\ \cdot > &= \underline{\delta} \delta^* \alpha \end{aligned}$$

Relations de précedence totale de gauche à droite (paragraphe 3.8)

$$\begin{aligned} \cdot &= \alpha \\ \cdot < &= \alpha \gamma^+ \\ \cdot > &= \delta^+ \alpha \gamma^* \tau \end{aligned}$$

Rappelons que les relations  $\alpha, \gamma, \delta$  ont été définies au paragraphe 1.6, les relations  $\varepsilon, \rho, \tau$  au paragraphe 3.4 et les relations  $\underline{\alpha}, \underline{\gamma}, \underline{\delta}$  au paragraphe 3.5.

Soit  $n$  le nombre d'éléments du vocabulaire  $V_T \cup V_N$  de la grammaire considérée. Posons

$$V_T \cup V_N = \{e_1, e_2, \dots, e_n\}$$

et représentons chaque relation binaire  $\rho$  sur  $V_T \cup V_N$  par une matrice booléenne  $R = \{r_{ij}\}$  de dimension  $n \times n$  ainsi définie :

$$r_{ij} = \text{vrai} \text{ si et seulement si } e_i \rho e_j$$

Les opérations définies sur les relations binaires induisent alors des opérations sur des matrices booléennes, opération que nous noterons de la même façon.

Soient  $A = \{a_{ij}\}, B = \{b_{ij}\}, C = \{c_{ij}\}$  trois matrices booléennes de dimensions  $n \times n$  représentant des relations binaires.

- Si  $C = A \cup B$  alors  $c_{ij} = a_{ij} \vee b_{ij}$ .

- Si  $C = A \cap B$  alors  $c_{ij} = a_{ij} \wedge b_{ij}$ .

- Si  $C = \bar{A}$  alors  $c_{ij} = \neg a_{ij}$ .

- Si  $C = AB$  alors  $c_{ij} = \bigvee_{k=1}^n a_{ik} \wedge b_{kj}$  et de ce fait le produit  $AB$  peut-

être calculé par l'algorithme :

```

pour i:=1 pas 1 jusquà n faire
  pour j:=1 pas 1 jusquà n faire
    pour k:=1 pas 1 jusquà n faire
      début s:=faux ;
      pour k:=1 pas 1 jusquà n faire
        s:=s∨A[i,k]∧B[k,j] ;
      C[i,j] :=s fin

```

- Du fait que  $A^+ = \bigcup_{i=1}^{n-1} A^i$  la fermeture transitive  $A^+$  de A peut-être calculée en utilisant l'algorithme simple et efficace de Warshall  $[w_a]$  : l'exécution de

```

pour k:=1 pas 1 jusqu'a n faire
pour i:=1 pas 1 jusqu'a n faire
pour j:=1 pas 1 jusqu'a n faire
A [i,j] := A [i,j] ∨ A [i,k] ∧ A [k,j]

```

transforme A en sa fermeture transitive. Pour obtenir la fermeture transitive réflexive  $\hat{A}^*$  il suffit de compléter  $A^+$  en mettant des vrai sur toute sa première diagonale.

En fait nous avons utilisé une variante de ces deux algorithmes. La procédure PROFER qui suit permet de calculer un produit ou une fermeture transitive :

```

procédure PROFER(A,B,C) ; booleen tableau A,B,C ;
pour k:=1 pas 1 jusqu'a n faire
pour i:=1 pas 1 jusqu'a n faire
si A [i,k] alors
pour j:=1 pas 1 jusqu'a n faire
C [i,j] := C [i,j] ∨ B [k,j]

```

si après avoir remis C à zéro (on met des faux partout dans C) on appelle PROFER(A,B,C) on obtient  $C=AB$ , à condition que C soit différent de A et de B. Si on appelle PROFER(A,A,A) alors on transforme A en sa fermeture transitive.

Si chaque ligne de matrice booléenne est représentée par une suite de "bits" découpée en plusieurs mémoires on peut alors utiliser le fait que dans la plupart des calculateurs digitaux il existe une instruction permettant de réaliser l'union logique de deux mémoires d'un seul coup. L'exécution de la dernière

boucle pour de la procédure PROFER peut alors être considérablement accélérée. D'autre part le fait de représenter chaque élément booléen par un seul "bit" permet de n'utiliser qu'un nombre réduit de mémoires pour représenter une matrice booléenne.

Moyennant tous ces éléments, la façon de programmer un calcul quelconque de matrices booléennes représentant des relations de précédence se déduit immédiatement des formules rappelées en tête de ce paragraphe.

#### 4.2 Généralités sur la programmation des différents analyseurs

La programmation de chaque analyseur de précédence est inspirée directement de sa description correspondante par un automate à deux piles. Signalons cependant deux différences :

- On n'intercale pas de symboles  $[, ]$  entre les éléments de la pile gauche. Les éléments passent de la pile droite à la gauche jusqu'à rencontre d'une précédence du type  $\succ$ . Après avoir reculé dans la pile gauche jusqu'à rencontre d'une précédence du type  $\prec$ , la sous-chaîne ainsi isolée est alors réduite, c'est-à-dire remplacée par un seul symbole non terminal. Ce processus est alors répété autant de fois qu'il le faut.

- Au départ la pile droite ne contient pas la totalité de la chaîne à analyser. Cette dernière est introduite dans la pile en cours d'analyse, élément par élément, au fur et à mesure des besoins.

Voici les conventions générales adoptées:

- Les symboles du vocabulaire de la grammaire considérée sont numérotés de 1 à NBSYMBOLS. Dans le cas où il est nécessaire de faire la distinction

entre opérateurs et opérands, les opérateurs sont numérotés de 1 à NBOPERATEURS et les opérands de NBOPERATEURS+1 à NBSYMBOLES. AXIOME a pour valeur le numéro de l'axiome.

- La chaîne à analyser est supposée être encadrée de deux symboles spéciaux # dont le numéro, différent de celui de tous les autres symboles, est égal à STOP. Cette chaîne est lue au moyen d'une entier procédure PROCHAINSYMBOLE dont la valeur est égale au numéro du prochain symbole lu.

- Les règles sont placées à la suite les unes des autres dans l'entier tableau R [1:RMAX], en remplaçant chaque symbole par son numéro, et chaque signe  $\rightarrow$  par un entier égal à la longueur de la chaîne qui suit. Le nombre 0 marque la fin de l'ensemble de toutes les règles. Par exemple la grammaire  $S \rightarrow ab, S \rightarrow aSb$  serait représentée par le tableau R qui suit

3	2	1	2	3	3	1	3	2	0	...
↑										↓
1										RMAX

en numérotant a par 1, b par 2 et S par 3.

- Les relations  $<$ ,  $=$ ,  $>$  et éventuellement la relation  $\Rightarrow^*$  entre deux opérands, sont respectivement représentées par les booléen tableau MI, ME, MS et SU.

- Les deux piles sont groupées en un seul entier tableau P [1:PMAX], chaque pile se trouvant à une extrémité de ce tableau.

#### 4.3 Analyseur de précedence totale déterministe

L'analyseur de précedence totale déterministe ne s'applique qu'aux grammaires de précedence totale (paragraphe 2.6) ou de précedence totale de

gauche à droite (paragraphe 3.9). Du fait que dans ces grammaires les relations  $\langle, =, \rangle$  sont incompatibles deux à deux il n'est pas nécessaire d'utiliser les trois relations ; on se contentera par exemple des relations  $=$  et  $\rangle$ . Cependant afin de détecter au plus tôt si la chaîne à analyser n'est pas une phrase, il peut être intéressant de compléter la relation  $\rangle$  en considérant qu'elle est vérifiée partout où aucune des relations  $\langle, =$  ne l'est. L'algorithme d'analyse étant très simple, le programme correspondant qui suit se passe de commentaires.

DEBUT ENTIER TABLEAU P[1:PMAX],R[1:RMAX] ;  
BOOLEEN TABLEAU ME,MS[1:NBSYMBOLS,1:NBSYMBOLS] ;  
ENTIER A,AXIOME,B,J,K,M,STOP,U,V ;

INITIALISATION:

A:=STOP:=PROCHAINSYMBOLE ; B:=PROCHAINSYMBOLE ; J:=0 ; K:=PMAX+1;

AVANCEMENT:

ECRIRE( " AVANCEMENT " ,B) ; J:=J+1 ; P[J]:=A ; A:=B ;

SI K > PMAX ALORS B:=PROCHAINSYMBOLE SINON

DEBUT B:=PIKJ ; K:=K+1 FIN ;

TEST D AVANCEMENT:

SI A=STOP ALORS ALLERA AVANCEMENT ;

SI B=STOP ALORS ALLERA DEBUT DE REcul ;

SI NON MSIA,BI ALORS ALLERA AVANCEMENT ;

DEBUT DE REcul:

J:=J+1 ; P[J]:=A ; K:=K-1 ; P[K]:=B ; M:=J ;

REcul:

ECRIRE( " REcul " ,A) ; B:=A ; M:=M-1 ; A:=P[M] ;

TEST DE REcul:

SI A=STOP ALORS ALLERA RECHERCHE DE REGLES ;

SI MEIA,BI ALORS ALLERA REcul ;

RECHERCHE DE REGLES:

POUR U:=2,U+RIU+2 TANTQUE RIU-1] ≠ 0 FAIRE

SI RIU=J-M ALORS

DEBUT POUR V:=1 PAS 1 JUSQUA J-M FAIRE

SI P[M+V] ≠ RIU+V ALORS ALLERA RIEN ;

B:=RIU-1] ; ALLERA REDUCTION ; RIEN: FIN ;

ECRIRE( " ERREUR " ) ; ALLERA FIN D ANALYSE ;

REDUCTION:

ECRIRE( " REDUCTION " ,B) ; J:=M-1 ;

SI B ≠ AXIOME OU A ≠ STOP OU PIK] ≠ STOP

ALORS ALLERA TEST D AVANCEMENT ;

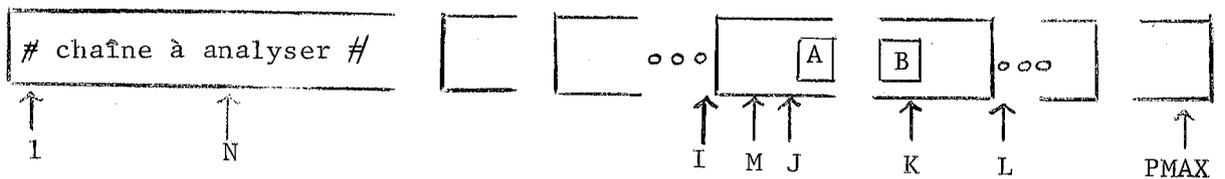
ECRIRE( " CHAINE CORRECTE " ) ;

FIN D ANALYSE:

FIN

#### 4.4 Analyseur de précedence totale non déterministe

L'analyseur de précedence totale non déterministe s'applique à n'importe quelle grammaire. Il se présente sous forme d'une procédure récursive ANALYSE qui se rappelle chaque fois que plusieurs éventualités se présentent : plusieurs relations de précedence vérifiées entre deux symboles ou plusieurs règles ayant le même membre droit. A chaque appel de cette procédure qui correspond donc à une bifurcation, il est nécessaire de retenir l'état dans lequel on se trouve car, si l'on arrive à une impasse, il faut revenir à la dernière bifurcation pour essayer une autre voie. De ce fait on est amené d'une part à conserver toute la chaîne à analyser en mémoire et d'autre part à recopier le contenu des deux piles à chaque appel de la procédure ANALYSE. Cette recopie se fait d'ailleurs à l'intérieur même du tableau P dont voici la configuration :

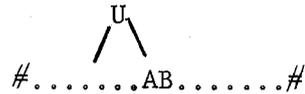


- A et B ont respectivement pour valeurs les sommets des deux piles.
- Les indices I,J et K,L permettent de délimiter la pile gauche et la pile droite.
- L'indice M repère jusqu'où on a reculé dans la pile gauche.
- L'indice N repère à quelle endroit on est dans la chaîne.

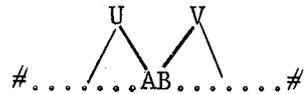
Dans le cas où il existe deux symboles A,B, pour lesquelles on a à la fois  $A < B$  et  $A > B$  il se peut que la chaîne en cours d'analyse

#.....AB.....#

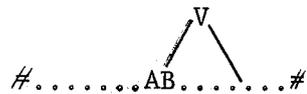
soit d'une part réduite ainsi



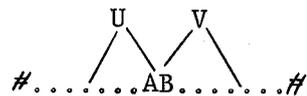
puis ainsi



et d'autre part réduite ainsi



puis ainsi



Dans ce cas l'analyseur cheminerait par deux voies différentes correspondant à une même structure. Nous avons donc prévu certains tests et certaines actions pour éviter ceci. Cependant cette légère modification de l'analyseur implique que les relations de précédence qui lui sont fournies soient de précédence totale de gauche à droite ou soit telles qu'elles permettent d'analyser des chaînes comportant aussi des symboles non terminaux ce qui est notamment le cas pour les relations  $\dot{<} = \alpha \delta^+$ ,  $\dot{=} = \alpha$ ,  $\dot{>} = \gamma^+ \alpha \cup (\gamma^+ \alpha \delta^+ \cap \overline{\alpha \delta^+})$ .

Une autre conséquence de cette modification est la suivante : si on considère que pour tout couple de symboles A,B on a toujours  $A \dot{<} B$ ,  $A \dot{=} B$ ,  $B \dot{>} A$ , alors cet analyseur se comporte comme l'analyseur par "substitution directe" le plus classique.

DEBUT ENTIER TABLEAU P[1:PMAX],R[1:RMAX] ;  
BOOLEEN TABLEAU MI,ME,MS[1:NBSYMBOLS,1:NBSYMBOLS] ;  
 ENTIER AXIOME,STOP,V ;

PROCEDURE ANALYSE(NO,A,B,I,J,K,L,M,N) ;  
 VALEUR NO,A,B,I,J,K,L,M,N ; ENTIER NO,A,B,I,J,K,L,M,N ;  
DEBUT ENTIER U ; BOOLEEN BON ;  
 AIGUILLAGE CAS:=DEBUT DE REcul CAS 1,  
 RECHERCHE DE REGLES CAS 2,REDUCTION CAS 3,CAS 4 ;  
 SI NO=0 ALORS ALLERA INITIALISATION CAS 0 ;

RECOPIE:

ECRIRE( " BIFURCATION " ) ; U:=J-1 ;  
POUR V:=1 PAS 1 JUSQUA U FAIRE P[J+V]:=P[I+V] ;  
 I:=J ; J:=J+U ; M:=M+U ; U:=L-K ;  
POUR V:=1 PAS 1 JUSQUA U FAIRE P[K-V]:=P[L-V] ;  
 L:=K ; K:=K-U ; ALLERA CAS[NO] ;

INITIALISATION CAS 0:

STOP:=P[1]:=PROCHAINSYMBOLE ;  
POUR U:=2,U+1 TANTQUE P[U-1] ≠ STOP FAIRE  
DEBUT P[U]:=PROCHAINSYMBOLE ; I:=U FIN ;  
 A:=P[1] ; B:=P[2] ; J:=1 ; K:=L:=PMAX ; N:=3 ;

AVANCEMENT:

ECRIRE( " AVANCEMENT " ,B) ; J:=J+1 ; P[J]:=A ; A:=B ;  
 SI K=L ALORS DEBUT B:=P[N] ; N:=N+1 FIN  
 SINON DEBUT B:=P[K] ; K:=K+1 FIN ;

TEST D AVANCEMENT:

SI A=STOP ALORS ALLERA AVANCEMENT ;  
 SI B=STOP ALORS ALLERA DEBUT DE REcul CAS 1 ;  
 SI MI[A,B] OU ME[A,B] ALORS  
DEBUT SI MS[A,B] ALORS ANALYSE(1,A,B,I,J,K,L,M,N) ;  
 ALLERA AVANCEMENT FIN ;  
 SI NON MS[A,B] ALORS ALLERA IMPASSE ;

DEBUT DE REcul CAS 1:

J:=J+1 ; P[J]:=A ; K:=K-1 ; P[K]:=B ; M:=J ;

REcul:

ECRIRE( " REcul " ,A) ; B:=A ; M:=M-1 ; A:=P[M] ;

TEST DE REcul:

SI A=STOP ALORS ALLERA RECHERCHE DE REGLES CAS 2 ;  
 SI ME[A,B] ALORS  
DEBUT SI MI[A,B] ALORS ANALYSE(2,A,B,I,J,K,L,M,N) ;  
 ALLERA REcul FIN ;

RECHERCHE DE REGLES CAS 2:

BON:= FAUX ;

NO:= SI A=STOP ALORS 3 SINON SI MS[A,B] ALORS 4 SINON 3 ;

POUR U:=2,U+R[U]+2 TANTQUE R[U-1] ≠ 0 FAIRE

SI R[U]=J-M ALORS

DEBUT POUR V:=1 PAS 1 JUSQUA J-M FAIRE

SI P[M+V] ≠ R[U+V] ALORS ALLERA RIEN ;

SI NO=4 ALORS DEBUT SI NON MI[A,R[U-1]] ET NON  
 ME[A,R[U-1]] ALORS ALLERA RIEN FIN ;

SI BON ALORS ANALYSE(NO,A,B,I,J,K,L,M,N) SINON BON:= VRAI ;  
 B:=R[U-1] ; RIEN: FIN ;

```
SI NON BON ALORS ALLERA IMPASSE ;  
REDUCTION CAS 3: CAS 4:  
  ECRIRE( " REDUCTION " ,B) ; J:=M-1 ;  
  SI B ≠ AXIOME OU A ≠ STOP OU P[K] ≠ STOP ALORS  
  ALLERA SI NO=4 ALORS AVANCEMENT SINON TEST D AVANCEMENT ;  
  ECRIRE( " CHAINE CORRECTE " ) ;  
IMPASSE:  
  ECRIRE( " IMPASSE " ) FIN DE LA PROCEDURE ;  
ANALYSE(0,0,0,0,0,0,0,0,0) FIN
```

4.5 Analyseur de précedence de gauche à droite

L'analyseur de précedence de gauche à droite que nous présentons ici correspond à la variante plus générale décrite au paragraphe 3.6. Du point de vue de déterminisme il est un peu plus général qu'un analyseur déterministe, tout en ne procédant pas par essais successifs. Nous imposons que les relations  $\langle, =, \rangle$  soient incompatibles deux à deux, mais nous n'imposons pas que toutes les règles aient des membres droits distincts. Cette dernière restriction est remplacée par une condition plus pratique à satisfaire.

(1) Les règles ayant un opérande comme membre gauche n'ont pas besoin d'avoir des membres droits distincts.

(2) Si deux règles ont le même opérateur R comme membre gauche et sont de la forme

$$\begin{array}{l} R-E_0 R_1 E_1 \dots R_n E_n \\ R-E'_0 R_1 E'_1 \dots R_n E'_n \end{array}$$

avec  $R_1, R_2, \dots, R_n \in V_R$  et  $E_0, E_1, \dots, E_n \in V_E \cup \{\Lambda\}$ , alors il n'existe pas

$E''_0, E''_1, \dots, E''_n \in V_E \cup \{\Lambda\}$  avec  $E_i \Rightarrow^* E''_i$  et  $E'_i \Rightarrow^* E''_i$  pour tout  $i \in \{0, 1, \dots, n\}$ .

Il faut remarquer que cet analyseur accepte certaines ambiguïtés mineures dans la grammaire ce qu'un analyseur rigoureusement déterministe ne pourrait faire.

DEBUT ENTIER TABLEAU P[1:PMAX],R[1:RMAX] ;  
BOOLEEN TABLEAU ME,MS[1:NBOPERATEURS,1:NBOPERATEURS],  
SU[NBOPERATEURS+1:NBSYMBOLES,NBOPERATEURS+1:NBSYMBOLES] ;  
ENTIER A,AXIOME,B,C,I,J,K,LR,N,STOP,U,V,W ; BOOLEEN FINI ;

INITIALISATION:

I:=J:=1 ; K:=PMAX+1 ; FINI:= FAUX ;  
A:=P[J]:=STOP:=PROCHAINSYMBOLE ; B:=PROCHAINSYMBOLE ;

AVANCEMENT:

ECRIRE( " AVANCEMENT " ,B) ; I:=J:=J+1 ; A:=P[J]:=B ;  
SI K > PMAX ALORS B:=PROCHAINSYMBOLE SINON  
DEBUT B:=P[K] ; K:=K+1 FIN ;

TEST D AVANCEMENT:

SI A=STOP ALORS ALLERA AVANCEMENT ;  
SI B=STOP ALORS ALLERA DEBUT DE REcul ;  
SI NON MS[A,B] ALORS ALLERA AVANCEMENT ;

DEBUT DE REcul:

LR:= SI I=J ALORS 0 SINON 1 ; K:=K-1 ; P[K]:=B ;

REcul:

ECRIRE( " REcul " ) ; C:=A ; I:=I-1 ; A:=P[I] ; LR:=LR+1 ;  
SI A > NBOPERATEURS ALORS  
DEBUT LR:=LR+1 ; E1: I:=I-1 ; A:=P[I] ;  
SI A > NBOPERATEURS ALORS ALLERA E1 FIN ;

TEST DE REcul:

SI A=STOP ALORS ALLERA RECHERCHE DE REGLES ;  
SI ME[A,C] ALORS ALLERA REcul ;

RECHERCHE DE REGLES:

N:=0 ; POUR U:=2,U+R[U]+2 TANTQUE R[U-1] ≠ 0 FAIRE  
SI LR=R[U] ALORS  
DEBUT V:=J ; POUR W:=U+LR PAS -1 JUSQUA U+1 FAIRE  
DEBUT SI R[W] ≤ NBOPERATEURS ALORS  
ALLERA SI R[W]=P[V] ALORS E4 SINON E6 ;  
SI P[V] ≤ NBOPERATEURS ALORS ALLERA E6 ;  
E2: SI SU[R[W],P[V]] ALORS ALLERA E3 ;  
V:=V-1 ;  
ALLERA SI P[V] > NBOPERATEURS ALORS E2 SINON E6 ;  
E3: V:=V-1 ;  
ALLERA SI P[V] > NBOPERATEURS ALORS E3 SINON E5 ;  
E4: V:=V-1 ; E5: FIN ;  
N:=N+1 ; P[K-N]:=R[U-1] ; E6: FIN ;

REDUCTION:

SI N=0 ALORS ALLERA ERREUR ;  
SI P[K-1] ≤ NBOPERATEURS ALORS  
DEBUT ECRIRE( " R-REDUCTION " ,P[K-1]) ;  
SI A=STOP ET B=STOP ET P[K-1]=AXIOME ALORS ALLERA BON ;  
J:=1 ; B:=P[K-1] ; ALLERA TEST D AVANCEMENT FIN ;  
SI A=STOP ET A=B ET AXIOME > NBOPERATEURS ALORS FINI:= VRAI ;  
POUR U:=1 PAS 1 JUSQUA N FAIRE  
DEBUT ECRIRE( " E-REDUCTION " ,P[K-U]) ; P[I+U]:=P[K-U] ;  
SI FINI ALORS  
DEBUT SI SU[AXIOME,P[K-U]] ALORS ALLERA BON FIN  
FIN ;

```
J:=I+N ; K:=K+1 ;  
SI NON FINI ALORS ALLERA TEST D AVANCEMENT ;  
ERREUR:  
  ECRIRE( " ERREUR " ) ; ALLERA FIN D ANALYSE ;  
BON:  
  ECRIRE( " CHAINE CORRECTE " ) ;  
FIN D ANALYSE:  
FIN
```

#### 4.6 Analyseur de précedence terminale

L'analyseur de précedence terminale que nous décrivons est un cas particulier de l'analyseur décrit au paragraphe précédent. Il s'applique aux grammaires dont les règles ne contiennent pas deux symboles non terminaux adjacents. Le vocabulaire terminal est choisi comme ensemble d'opérateurs et de ce fait la seule autre restriction qui subsiste est l'existence de relations de précedence terminales incompatibles deux à deux. NBOPERATEURS est donc aussi égal au nombre de symboles terminaux de la grammaire. Comme dans le cas précédent cet analyseur accepte aussi des ambiguïtés mineures dans la grammaire.

Il faut remarquer que du fait que les symboles non terminaux n'interviennent jamais dans les relations de précedence ils ne jouent qu'un rôle secondaire dans l'analyse. Il est donc possible de décrire un analyseur qui n'aurait pas besoin des règles de la grammaire. Cet analyseur trouverait la structure de toute phrase mais malheureusement ne reconnaîtrait pas toujours si celle-ci est correcte.

De plus si l'on représentait la structure obtenue par un arbre syntaxique, on ne connaîtrait pas le symbole non terminal qui se trouve à chaque noeud. C'est sous cette forme que Floyd [F1.1] avait présenté l'analyseur de précedence terminale.

```

DEBUT ENTIER TABLEAU P[1:PMAX],R[1:RMAX];
BOOLEEN TABLEAU ME,MS[1:NBOPERATEURS,1:NBOPERATEURS],
SU[NBOPERATEURS+1:NBSYMBOLES,NBOPERATEURS+1:NBSYMBOLES];
ENTIER A,AXIOME,B,C,I,J,LR,N,STOP,U,V,W; BOOLEEN FINI;

INITIALISATION:
I:=J:=1; FINI:= FAUX;
A:=P[J]:=STOP:=PROCHAINSYMBOLE; B:=PROCHAINSYMBOLE;
AVANCEMENT:
Ecrire(" AVANCEMENT " ,B);
I:=J:=J+1; A:=P[J]:=B; B:=PROCHAINSYMBOLE;
TEST D'AVANCEMENT:
SI A=STOP ALORS ALLERA AVANCEMENT;
SI B=STOP ALORS ALLERA DEBUT DE REcul;
SI NON MS[A,B] ALORS ALLERA AVANCEMENT;
DEBUT DE REcul:
LR:= SI I=J ALORS 0 SINON 1;
REcul:
Ecrire(" REcul " ); C:=A; I:=I-1; A:=P[I]; LR:=LR+1;
SI A > NBOPERATEURS ALORS
DEBUT LR:=LR+1; E1: I:=I-1; A:=P[I];
SI A > NBOPERATEURS ALORS ALLERA E1 FIN;
TEST DE REcul:
SI A=STOP ALORS ALLERA RECHERCHE DE REGLES;
SI ME[A,C] ALORS ALLERA REcul;
RECHERCHE DE REGLES:
N:=0; POUR U:=2,U+R[U]+2 TANTQUE R[U-1] ≠ 0 FAIRE
SI LR=R[U] ALORS
DEBUT V:=J; POUR W:=U+LR PAS -1 JUSQUA U+1 FAIRE
DEBUT SI R[W] < NBOPERATEURS ALORS
ALLERA SI R[W]=P[V] ALORS E4 SINON E6;
SI P[V] < NBOPERATEURS ALORS ALLERA E6;
E2: SI SU[R[W],P[V]] ALORS ALLERA E3;
V:=V-1;
ALLERA SI P[V] > NBOPERATEURS ALORS E2 SINON E6;
E3: V:=V-1;
ALLERA SI P[V] > NBOPERATEURS ALORS E3 SINON E5;
E4: V:=V-1; E5: FIN;
N:=N+1; P[J+N]:=R[U-1]; E6: FIN;
REDUCTION:
SI N=0 ALORS ALLERA ERREUR;
SI A=STOP ET A=B ET AXIOME > NBOPERATEURS ALORS FINI:= VRAI;
POUR U:=1 PAS 1 JUSQUA N FAIRE
DEBUT Ecrire(" REDUCTION " ,P[J+U]); P[I+U]:=P[J+U];
SI FINI ALORS
DEBUT SI SU[AXIOME,P[J+U]] ALORS ALLERA BON FIN
FIN;
J:=I+N;
SI NON FINI ALORS ALLERA TEST D'AVANCEMENT;
ERREUR:
Ecrire(" ERREUR " ); ALLERA FIN D'ANALYSE;
BON:
Ecrire(" CHAINE CORRECTE " );
FIN D'ANALYSE:
FIN

```



## CHAPITRE 5 : Applications aux langages de programmation

### 5.1 Précédence totale et Algol 60

Le langage de programmation Algol a été défini au moyen d'une grammaire "context-free" (ou sous forme normale de Backus). Cette grammaire est malheureusement ambiguë du fait qu'elle ne permet pas d'exprimer le lien entre l'apparition d'un identificateur et sa déclaration, ce qui entraîne parfois plusieurs interprétations possibles. Afin de se ramener à une grammaire non ambiguë, nous avons été amené à introduire quatre nouveaux symboles terminaux qui sont :

IDENTIFICATEUR-ARITHMETIQUE  
 IDENTIFICATEUR-BOOLEEN  
 IDENTIFICATEUR-DE-DESIGNATION  
 IDENTIFICATEUR-DE-PROCEDURE-INSTRUCTION

Après cette modification et quelques autres n'altérant en rien le langage Algol, nous avons obtenu une grammaire de précédence totale [\*] dont nous donnons la liste des règles et les relations de précédence totale aux pages suivantes. La seule restriction introduite est l'interdiction des étiquettes purement numériques, ce que d'ailleurs la plupart des compilateurs Algol interdisent aussi.

Pour l'écriture des règles de grammaire nous avons adopté les conventions suivantes :

- Chaque symbole terminal (symbole de base) ou non terminal (métavariable) est représenté par une suite quelconque de caractères ne comportant aucun espace libre ; les espaces servent de séparateurs.

- Lorsqu'une règle a un membre gauche identique à celui de la règle précédente, celui-ci est supprimé et le signe  $\rightarrow$  est remplacé par le mot OU (ou OR).

---

[\*] Du fait que dans cette grammaire les symboles terminaux 'CROCHET-GAUCHE' et 1 sont à la fois dans la relation  $\alpha_V^+$  et  $\delta^+ \alpha_V^+$  celle-ci n'est pas de précédence totale de gauche à droite.

- Le signe  $\rightarrow$  est remplacé par  $::=$ .

Dans le tableau représentant des relations de précedence les signes  $\leftarrow, \bar{=}, \succ$  sont respectivement remplacés par 1,2,4. Si le tableau est trop grand pour tenir sur une seule page, il est découpé en plusieurs bandes verticales.



## REGLES (SUITE)

```

INSTRUCTION-CONDITIONNELLE ::= INSTRUCTION-SI
OU INSTRUCTION-SI 'SINCN' INSTRUCTION-QUELCONQUE
OU INSTRUCTION-SI 'SINEN'
OU PROPOSITION-SI INSTRUCTION-POUR
OU IDENTIFICATEUR : INSTRUCTION-CONDITIONNELLE
INSTRUCTION-SI ::= PROPOSITION-SI INSTRUCTION-INCONDITIONNELLE
OU PROPOSITION-SI
PROPOSITION-SI ::= 'SI' EXPRESSION-BOOLEENNE 'ALORS'
INSTRUCTION-POUR ::= IDENTIFICATEUR : INSTRUCTION-POUR
OU PROPOSITION-POUR
PROPOSITION-POUR ::= 'POUR' PARTIE-GAUCHE-ARITHMETIQUE := FIN-DE-PROPOSITION-POUR
FIN-DE-PROPOSITION-POUR ::= EXPRESSION-ARITHMETIQUE 'FAIRE'
OU EXPRESSION-ARITHMETIQUE 'PAS' EXPRESSION-ARITHMETIQUE 'JUSQUA'
EXPRESSION-ARITHMETIQUE 'FAIRE'
OU 'TANTQUE' EXPRESSION-BOOLEENNE 'FAIRE'
OU EXPRESSION-ARITHMETIQUE , FIN-DE-PROPOSITION-POUR
OU EXPRESSION-ARITHMETIQUE 'PAS' EXPRESSION-ARITHMETIQUE 'JUSQUA'
EXPRESSION-ARITHMETIQUE , FIN-DE-PROPOSITION-POUR
OU 'TANTQUE' EXPRESSION-BOOLEENNE , FIN-DE-PROPOSITION-POUR
EXPRESSION-ARITHMETIQUE ::= EXPRESSION-ARITHMETIQUE-BIS
EXPRESSION-ARITHMETIQUE-BIS ::= EXPRESSION-ARITHMETIQUE-SIMPLE
OU DEBUT-D'EXPRESSION-ARITHMETIQUE 'SINCN' EXPRESSION-ARITHMETIQUE-BIS
EXPRESSION-ARITHMETIQUE-SIMPLE ::= PROPOSITION-SI EXPRESSION-ARITHMETIQUE-SIMPLE
EXPRESSION-ARITHMETIQUE-SIMPLE-BIS ::= EXPRESSION-ARITHMETIQUE-SIMPLE-BIS
EXPRESSION-ARITHMETIQUE-SIMPLE-TER ::= EXPRESSION-ARITHMETIQUE-SIMPLE-TER
OU OPERATEUR-ADDITIF TERME
OU EXPRESSION-ARITHMETIQUE-SIMPLE-TER OPERATEUR-ADDITIF TERME
OPERATEUR-ADDITIF ::= +
OU -
TERME ::= TERME-BIS
TERME-BIS ::= FACTEUR
OU TERME-BIS OPERATEUR-MULTIPLICATIF FACTEUR
OPERATEUR-MULTIPLICATIF ::= *
OU /
OU 'OPERATEUR-DE-DIVISION-ENTIERE'
FACTEUR ::= FACTEUR-BIS
FACTEUR-BIS ::= PRIMAIRE-ARITHMETIQUE
OU FACTEUR-BIS 'OPERATEUR-D'EXPONENTIATION' PRIMAIRE-ARITHMETIQUE
PRIMAIRE-ARITHMETIQUE ::= NOMBRE
OU PARTIE-GAUCHE-ARITHMETIQUE
OU FONCTION-ARITHMETIQUE
OU PARENTHESE-GAUCHE FIN-DE-PRIMAIRE-ARITHMETIQUE
PARENTHESE-GAUCHE ::= (
FIN-DE-PRIMAIRE-ARITHMETIQUE ::= EXPRESSION-ARITHMETIQUE )
NOMBRE ::= NOMBRE-DECIMAL
OU NOMBRE-DECIMAL OPERATEUR-D'EXPONENTIATION-DECIMALE ENTIER
OPERATEUR-D'EXPONENTIATION-DECIMALE ::= '10'
OU '10' +
OU '10' -
NOMBRE-DECIMAL ::= ENTIER
OU . ENTIER
OU ENTIER . ENTIER
ENTIER ::= CHIFFRES
CHIFFRES ::= CHIFFRE
OU CHIFFRES CHIFFRE
PARTIE-GAUCHE-ARITHMETIQUE ::= IDENTIFICATEUR-ARITHMETIQUE
OU DEBUT-DE-PARTIE-GAUCHE-ARITHMETIQUE INDICES
DEBUT-DE-PARTIE-GAUCHE-ARITHMETIQUE ::= IDENTIFICATEUR-ARITHMETIQUE 'CROCHET-GAUCHE'
FONCTION-ARITHMETIQUE ::= DEBUT-DE-FONCTION-ARITHMETIQUE PARAMETRES-EFFECTIFS
DEBUT-DE-FONCTION-ARITHMETIQUE ::= IDENTIFICATEUR-ARITHMETIQUE (
EXPRESSION-BOOLEENNE ::= EXPRESSION-BOOLEENNE-BIS
EXPRESSION-BOOLEENNE-SIMPLE ::= EXPRESSION-BOOLEENNE-SIMPLE
OU DEBUT-D'EXPRESSION-BOOLEENNE 'SINCN' EXPRESSION-BOOLEENNE-BIS
DEBUT-D'EXPRESSION-BOOLEENNE ::= PROPOSITION-SI EXPRESSION-BOOLEENNE-SIMPLE
EXPRESSION-BOOLEENNE-SIMPLE ::= EXPRESSION-BOOLEENNE-SIMPLE-BIS
EXPRESSION-BOOLEENNE-SIMPLE-BIS ::= IMPLICATION
OU EXPRESSION-BOOLEENNE-SIMPLE-BIS 'EQUIVALENT' IMPLICATION
IMPLICATION ::= IMPLICATION-BIS
IMPLICATION-BIS ::= TERME-BOOLEEN
OU IMPLICATION-BIS 'IMPLIQUE' TERME-BOOLEEN
TERME-BOOLEEN ::= TERME-BOOLEEN-BIS
TERME-BOOLEEN-BIS ::= FACTEUR-BOOLEEN
OU TERME-BOOLEEN-BIS 'OU' FACTEUR-BOOLEEN
FACTEUR-BOOLEEN ::= FACTEUR-BOOLEEN-BIS
FACTEUR-BOOLEEN-BIS ::= SECONDAIRE-BOOLEEN
OU FACTEUR-BOOLEEN-BIS 'ET' SECONDAIRE-BOOLEEN
SECONDAIRE-BOOLEEN ::= PRIMAIRE-BOOLEEN
OU 'NON' PRIMAIRE-BOOLEEN
PRIMAIRE-BOOLEEN ::= 'VRAI'
OU 'FAUX'
OU PARTIE-GAUCHE-BOOLEENNE
OU RELATION
OU FONCTION-BOOLEENNE
OU PARENTHESE-GAUCHE FIN-DE-PRIMAIRE-BOOLEEN

```

PL n° 2 GRAMMAIRE ALGOL DE PRECEDENCE TOTALE

REGLES (SUITE)

```

FIN-DE-PRIMAIRE-BOOLEEN ::= EXPRESSION-BOOLEENNE )
RELATION ::= EXPRESSION-ARITHMETIQUE-SIMPLE-BIS OPERATEUR-DE-RELATION
OPERATEUR-DE-RELATION ::= 'INFERIEUR'
                        OU 'INFERIEUR-OU-EGAL'
                        OU '='
                        OU 'NON-EGAL'
                        OU 'SUPERIEUR-OU-EGAL'
                        OU 'SUPERIEUR'
FONCTION-BOOLEENNE ::= DEBUT-DE-FONCTION-BOOLEENNE PARAMETRES-EFFECTIFS
DEBUT-DE-FONCTION-BOOLEENNE ::= IDENTIFICATEUR-BOOLEENNE (
PARTIE-GAUCHE-BOOLEENNE ::= IDENTIFICATEUR-BOOLEENNE
DEBUT-DE-PARTIE-GAUCHE-BOOLEENNE ::= IDENTIFICATEUR-BOOLEENNE 'CROCHET-GAUCHE'
EXPRESSION-DE-DESIGNATION ::= EXPRESSION-DE-DESIGNATION-BIS
EXPRESSION-DE-DESIGNATION-BIS ::= EXPRESSION-DE-DESIGNATION-SIMPLE
DEBUT-D'EXPRESSION-DE-DESIGNATION ::= DEBUT-D'EXPRESSION-DE-DESIGNATION 'SINON' EXPRESSION-DE-DESIGNATION-BIS
EXPRESSION-DE-DESIGNATION-SIMPLE ::= PROPOSITION-SI EXPRESSION-DE-DESIGNATION-SIMPLE
INDICATEUR-D'AIGUILLAGE ::= IDENTIFICATEUR-DE-DESIGNATION
INDICES ::= IDENTIFICATEUR-DE-DESIGNATION
INDICE-FINAL ::= IDENTIFICATEUR-DE-DESIGNATION 'CROCHET-GAUCHE'
LISTE-D'IDENTIFICATEURS ::= INDICE-FINAL
IDENTIFICATEUR ::= EXPRESSION-ARITHMETIQUE , INDICES
IDENTIFICATEUR-BIS ::= EXPRESSION-ARITHMETIQUE 'CROCHET-DROIT'
IDENTIFICATEUR-TER ::= IDENTIFICATEUR
                        OU IDENTIFICATEUR , LISTE-D'IDENTIFICATEURS
                        OU IDENTIFICATEUR-BIS
                        OU IDENTIFICATEUR-TER
CHIFFRE ::= 0
           OU 1
           OU 2
           OU 3
           OU 4
           OU 5
           OU 6
           OU 7
           OU 8
           OU 9
LETTRE ::= A
         OU B
         OU C
         OU D
         OU E
         OU F
         OU G
         OU H
         OU I
         OU J
         OU K
         OU L
         OU M
         OU N
         OU O
         OU P
         OU Q
         OU R
         OU S
         OU T
         OU U
         OU V
         OU W
         OU X
         OU Y
         OU Z
    
```













## 5.2 Précédence de gauche à droite et recherche d'erreurs dans un programme

Algol [\*]

Nous avons été amené à résoudre le problème suivant : étant donné un programme Algol, trouver ses erreurs syntaxiques en un seul passage afin d'accélérer sa mise au point. Pour ceci nous avons utilisé l'analyseur à relations de précédence de gauche à droite décrit au paragraphe 4.5. Du fait que cet analyseur s'applique aussi à des grammaires comportant certaines ambiguïtés nous n'avons pas eu besoin d'introduire de nouveaux symboles terminaux dans la grammaire Algol. Par contre afin d'éviter l'écriture d'un programme spécial (éditeur) dont le rôle aurait été de reconnaître chaque symbole de base Algol et de le coder, nous avons introduit des règles de grammaire supplémentaires spécifiant la façon d'écrire chaque symbole de base Algol. Nous avons adopté les mêmes conventions que celles utilisées pour le compilateur Algol (réalisé par J.C. Boussard [Bou]) qui équipe notre calculateur (IBM 7044). De ce fait le vocabulaire terminal de notre grammaire se réduit à l'ensemble des caractères disponibles sur le matériel d'entrée-sortie.

Cependant afin d'obtenir des relations de précédence incompatibles deux à deux il a fallu dédoubler certains caractères. Ce fut le cas des lettres, suivant qu'elles sont utilisées pour écrire un symbole de base ou pour composer un identificateur, du caractère " " et du caractère '. Ces distinctions ainsi que la suppression de certaines séquences inutiles telles que les commentaires, sont faites par quelques instructions intercalées dans le programme général d'analyse.

Que se passe-t-il lorsque l'analyseur "cale" sur une erreur ? Du fait que nous avons complété d'une façon empirique les relations de précédence de façon à ce que pour tout couple d'opérateurs il y ait une relation de vérifiée, ceci ne peut se produire que lorsque l'analyseur ne trouve pas de règle à appliquer à une sous-chaîne. Après avoir signalé l'erreur on supprime cette sous-chaîne ou on la remplace par un symbole spécial qui se comporte comme la liste de tous

---

[\*] Pour plus de détails à ce sujet consulter l'article donné en référence [Co.1].

les opérandes de la grammaire et on poursuit l'analyse pour détecter d'autres erreurs.

Voici successivement la grammaire Algol utilisée, ses relations de précedence de gauche à droite, la relation  $\Rightarrow^*$  entre deux opérandes et un exemple de recherche d'erreurs dans un programme Algol. Il faut noter que l'analyseur a été programmé en Algol et que de ce fait il fonctionnerait au moins 6 fois plus vite s'il avait été écrit en langage machine.

PL N° 9 GRAMMAIRE ALGOL POUR RECHERCHE D'ERREURS

REGLES

```

PROGRAMME-DELIMITE ::= ///= PROGRAMME ///=
PROGRAMME          ::= BLOC
                   CU INSTRUCTION-COMPOSEE
                   CU BLOC 'POINT-VIRGULE'
                   OU INSTRUCTION-COMPOSEE 'POINT-VIRGULE'
BLOC                ::= IDENTIFICATEUR 'DEUX-POINTS' BLOC
                   CU 'DEBUT' INTERIEUR-DE-BLOC 'FIN'
INTERIEUR-DE-BLOC  ::= SUITE-DE-DECLARATIONS 'POINT-VIRGULE'
                   OU SUITE-DE-DECLARATIONS 'POINT-VIRGULE' INSTRUCTION
                   OU INTERIEUR-DE-BLOC 'POINT-VIRGULE' INSTRUCTION
                   OU SUITE-DE-DECLARATIONS 'POINT-VIRGULE' 'COMMENTAIRE' 'POINT-VIRGULE'
                   OU SUITE-DE-DECLARATIONS 'POINT-VIRGULE' 'COMMENTAIRE' 'POINT-VIRGULE'
                   INSTRUCTION
                   OU INTERIEUR-DE-BLOC 'POINT-VIRGULE' 'COMMENTAIRE' 'POINT-VIRGULE'
                   OU INTERIEUR-DE-BLOC 'POINT-VIRGULE' 'COMMENTAIRE' 'POINT-VIRGULE'
                   INSTRUCTION
INSTRUCTION-COMPOSEE ::= IDENTIFICATEUR 'DEUX-POINTS' INSTRUCTION-COMPOSEE
                   OU 'DEBUT' 'FIN'
                   CU 'DEBUT' SUITE-D-INSTRUCTIONS 'FIN'
SUITE-D-INSTRUCTIONS ::= INSTRUCTION
                   OU 'POINT-VIRGULE'
                   OU 'POINT-VIRGULE' INSTRUCTION
                   CL SUITE-D-INSTRUCTIONS 'POINT-VIRGULE'
                   CL SUITE-D-INSTRUCTIONS 'POINT-VIRGULE' INSTRUCTION
                   CL 'COMMENTAIRE' 'POINT-VIRGULE'
                   OU 'POINT-VIRGULE' 'COMMENTAIRE' 'POINT-VIRGULE'
                   CL 'COMMENTAIRE' 'POINT-VIRGULE' INSTRUCTION
                   OU 'POINT-VIRGULE' 'COMMENTAIRE' 'POINT-VIRGULE' INSTRUCTION
                   OU SUITE-D-INSTRUCTIONS 'POINT-VIRGULE' 'COMMENTAIRE' 'POINT-VIRGULE'
                   OU SUITE-D-INSTRUCTIONS 'POINT-VIRGULE' 'COMMENTAIRE' 'POINT-VIRGULE'
                   INSTRUCTION
INSTRUCTION         ::= INSTRUCTION-PCUR
                   OU INSTRUCTION-INCONDITIONNELLE
                   OU INSTRUCTION-CONDITIONNELLE
INSTRUCTION-POUR   ::= IDENTIFICATEUR 'DEUX-POINTS' INSTRUCTION-POUR
                   OU 'PCUR' VARIABLE := LISTE-DE-PCUR 'FAIRE'
                   OU 'POUR' VARIABLE := LISTE-DE-PCUR 'FAIRE' INSTRUCTION
LISTE-DE-POUR      ::= ELEMENT-D'UNE-LISTE-DE-POUR
ELEMENT-D'UNE-LISTE-DE-POUR ::= EXPRESSION-ARITHMETIQUE
                   CL EXPRESSION-ARITHMETIQUE 'TANTQUE' EXPRESSION-BOOLEENNE
                   OU EXPRESSION-ARITHMETIQUE 'PAS' EXPRESSION-ARITHMETIQUE 'JUSQUA'
                   EXPRESSION-ARITHMETIQUE
INSTRUCTION-INCONDITIONNELLE ::= BLOC
                   OU INSTRUCTION-COMPOSEE
                   OU INSTRUCTION-DE-BASE
INSTRUCTION-DE-BASE ::= IDENTIFICATEUR 'DEUX-POINTS'
                   OU IDENTIFICATEUR 'DEUX-POINTS' INSTRUCTION-DE-BASE
                   CL INSTRUCTION-D'AFFECTATION
                   CU INSTRUCTION-ALLER-A
                   CU INSTRUCTION-PROCEDURE
INSTRUCTION-D'AFFECTATION ::= LISTE-DE-PARTIES-GAUCHES := EXPRESSION-ARITHMETIQUE
                   OU LISTE-DE-PARTIES-GAUCHES := EXPRESSION-BOOLEENNE
LISTE-DE-PARTIES-GAUCHES ::= VARIABLE
                   OU LISTE-DE-PARTIES-GAUCHES := VARIABLE
INSTRUCTION-ALLER-A ::= 'ALLERA' EXPRESSION-DE-DESIGNATION
INSTRUCTION-PROCEDURE ::= IDENTIFICATEUR
                   OU IDENTIFICATEUR (PAR.EFFECTIFS)
IDENTIFICATEUR (PAR.EFFECTIFS) ::= IDENTIFICATEUR 'PARENTHESE-GAUCHE' LISTE-DE-PARAMETRES-EFFECTIFS
                   'PARENTHESE-DROITE'
                   OU IDENTIFICATEUR (PAR.EFFECTIFS) LETTRES 'DEUX-POINTS'
                   (LISTE-DE-PARAMETRES-EFFECTIFS)
IDENTIFICATEUR (PAR.EFFECTIFS) LETTRES ::= IDENTIFICATEUR 'PARENTHESE-GAUCHE' LISTE-DE-PARAMETRES-EFFECTIFS
                   'PARENTHESE-DROITE' CHAINE-DE-LETTRES
                   (LISTE-DE-PAR.EFFECTIFS) LETTRES 'DEUX-POINTS'
                   (LISTE-DE-PAR.EFFECTIFS) LETTRES ::= 'PARENTHESE-GAUCHE' LISTE-DE-PARAMETRES-EFFECTIFS 'PARENTHESE-DROITE'
                   CHAINE-DE-LETTRES
CHAINE-DE-LETTRES ::= LETTRE
                   OU CHAINE-DE-LETTRES LETTRE
INSTRUCTION-CONDITIONNELLE ::= IDENTIFICATEUR 'DEUX-POINTS' INSTRUCTION-CONDITIONNELLE
                   CU PROPOSITION-SI
                   CU PROPOSITION-SI INSTRUCTION-INCONDITIONNELLE
                   CU PROPOSITION-SI INSTRUCTION-PCUR
                   CU PROPOSITION-SI 'SINON'
                   CU PROPOSITION-SI 'SINON' INSTRUCTION
                   CU PROPOSITION-SI INSTRUCTION-INCONDITIONNELLE 'SINON'
                   CU PROPOSITION-SI INSTRUCTION-INCONDITIONNELLE 'SINON' INSTRUCTION
PROPOSITION-SI     ::= 'SI' EXPRESSION-BOOLEENNE 'ALORS'
SUITE-DE-DECLARATIONS ::= DECLARATION-AUTRE-QUE-DE-PROCEDURE
                   OU SUITE-DE-DECLARATIONS 'POINT-VIRGULE' DECLARATION-AUTRE-QUE-DE-PROCEDURE
                   CL SUITE-DE-DECLARATIONS-INCOMPLETE 'POINT-VIRGULE'
                   CL SUITE-DE-DECLARATIONS-INCOMPLETE 'POINT-VIRGULE' CORPS-DE-PROCEDURE
                   CL 'COMMENTAIRE' 'POINT-VIRGULE' DECLARATION-AUTRE-QUE-DE-PROCEDURE
                   CL SUITE-DE-DECLARATIONS 'POINT-VIRGULE' 'COMMENTAIRE' 'POINT-VIRGULE'
                   DECLARATION-AUTRE-QUE-DE-PROCEDURE
                   CU SUITE-DE-DECLARATIONS-INCOMPLETE 'POINT-VIRGULE' 'COMMENTAIRE'
                   'POINT-VIRGULE'
                   OU SUITE-DE-DECLARATIONS-INCOMPLETE 'POINT-VIRGULE' 'COMMENTAIRE'
                   'POINT-VIRGULE' CORPS-DE-PROCEDURE
CORPS-DE-PROCEDURE ::= INSTRUCTION
                   CL 'CODE' 'PCODE'
DECLARATION-AUTRE-QUE-DE-PROCEDURE ::= DECLARATION-DE-TYPE
                   OU DECLARATION-DE-TABLEAU
                   CU DECLARATION-D'ATGULLAGE

```

REGLES (SUITE)

```

DECLARATION-DE-TYPE ::= TYPE LISTE-D'IDENTIFICATEURS
                    OL 'REMANENT' TYPE LISTE-D'IDENTIFICATEURS
                    TYPE ::= 'REEL'
                    CU 'ENTIER'
                    CL 'BOCLEEN'
DECLARATION-DE-TABLEAU ::= 'TABLEAU' LISTE-DE-TABLEAU
                    OU 'REMANENT' 'TABLEAU' LISTE-DE-TABLEAU
                    CU TYPE 'TABLEAU' LISTE-DE-TABLEAU
                    CL 'REMANENT' TYPE 'TABLEAU' LISTE-DE-TABLEAU
LISTE-DE-TABLEAU ::= IDENTIFICATEUR-AVEC-BORNES
                    OU IDENTIFICATEUR-AVEC-BORNES , LISTE-DE-TABLEAU
                    CU IDENTIFICATEUR , LISTE-DE-TABLEAU
IDENTIFICATEUR-AVEC-BORNES ::= IDENTIFICATEUR 'CROCHET-GAUCHE' LISTE-DE-PAIRES-DE-BORNES
                    'CROCHET-DROIT'
LISTE-DE-PAIRES-DE-BORNES ::= PAIRE-DE-BORNES
                    CU PAIRE-DE-BORNES , LISTE-DE-PAIRES-DE-BORNES
PAIRE-DE-BORNES ::= EXPRESSION-ARITHMETIQUE 'DEUX-POINTS-SEPARATEUR-DE-BORNES'
                    EXPRESSION-ARITHMETIQUE
DECLARATION-D'AIGUILLAGE ::= 'AIGUILLAGE' IDENTIFICATEUR := LISTE-D'AIGUILLAGES
LISTE-D'AIGUILLAGES ::= EXPRESSION-DE-DESIGNATION
                    CU EXPRESSION-DE-DESIGNATION , LISTE-D'AIGUILLAGES
SUITE-DE-DECLARATIONS-INCOMPLETE ::= SUITE-DE-DECLARATIONS-BIEN-INCOMPLETE
                    OL SUITE-DE-DECLARATIONS-INCOMPLETE 'PCINT-VIRGULE' SPECIFICATION
                    CU SUITE-DE-DECLARATIONS-INCOMPLETE 'PCINT-VIRGULE' 'COMMENTAIRE'
                    'POINT-VIRGULE' SPECIFICATION
SUITE-DE-DECLARATIONS-BIEN-INCOMPLETE ::= SUITE-DE-DECLARATIONS-TRES-INCOMPLETE
                    CL SUITE-DE-DECLARATIONS-TRES-INCOMPLETE 'PCINT-VIRGULE' PARTIE-VALEUR
                    OU SUITE-DE-DECLARATIONS-TRES-INCOMPLETE 'PCINT-VIRGULE' 'COMMENTAIRE'
                    'POINT-VIRGULE' PARTIE-VALEUR
                    PARTIE-VALEUR ::= 'VALEUR' LISTE-D'IDENTIFICATEURS
SUITE-DE-DECLARATIONS-TRES-INCOMPLETE ::= ENTETE-DE-PROCEDURE
                    OU SUITE-DE-DECLARATIONS 'PCINT-VIRGULE' ENTETE-DE-PROCEDURE
                    CU 'COMMENTAIRE' 'PCINT-VIRGULE' ENTETE-DE-PROCEDURE
                    OU SUITE-DE-DECLARATIONS 'PCINT-VIRGULE' 'COMMENTAIRE' 'PCINT-VIRGULE'
                    ENTETE-DE-PROCEDURE
ENTETE-DE-PROCEDURE ::= 'PROCEDURE' IDENTIFICATEUR
                    CU 'PROCEDURE' IDENTIFICATEUR(PAR.FORMELS)
                    OU TYPE 'PROCEDURE' IDENTIFICATEUR
                    CL TYPE 'PROCEDURE' IDENTIFICATEUR(PAR.FORMELS)
IDENTIFICATEUR(PAR.FORMELS) ::= IDENTIFICATEUR 'PARENTHESE-GAUCHE' LISTE-D'IDENTIFICATEURS
                    'PARENTHESE-DROITE'
                    OU IDENTIFICATEUR(PAR.FORMELS) LETTRES 'DEUX-POINTS'
                    (LISTE-DE-PARAMETRES-FORMELS)
IDENTIFICATEUR(PAR.FORMELS) LETTRES ::= IDENTIFICATEUR 'PARENTHESE-GAUCHE' LISTE-D'IDENTIFICATEURS
                    'PARENTHESE-DROITE' CHAINE-DE-LETTRES
                    (LISTE-DE-PARAMETRES-FORMELS) ::= 'PARENTHESE-GAUCHE' LISTE-D'IDENTIFICATEURS 'PARENTHESE-DROITE'
                    (LISTE-DE-PARAMETRES-FORMELS) LETTRES 'DEUX-POINTS'
                    (LISTE-DE-PARAMETRES-FORMELS)
                    (LISTE-DE-PARAMETRES-FORMELS) LETTRES ::= 'PARENTHESE-GAUCHE' LISTE-D'IDENTIFICATEURS 'PARENTHESE-DROITE'
                    CHAINE-DE-LETTRES
SPECIFICATION ::= TYPE LISTE-D'IDENTIFICATEURS
                    OU 'CHAINE' LISTE-D'IDENTIFICATEURS
                    CU 'ETIQUETTE' LISTE-D'IDENTIFICATEURS
                    OU 'AIGUILLAGE' LISTE-D'IDENTIFICATEURS
                    OU 'TABLEAU' LISTE-D'IDENTIFICATEURS
                    OU 'PROCEDURE' LISTE-D'IDENTIFICATEURS
                    CU TYPE 'TABLEAU' LISTE-D'IDENTIFICATEURS
                    OU TYPE 'PROCEDURE' LISTE-D'IDENTIFICATEURS
E-D'IDENTIFICATEURS ::= IDENTIFICATEUR
                    OU IDENTIFICATEUR , LISTE-D'IDENTIFICATEURS
IDENTIFICATEUR ::= LETTRE
                    OU IDENTIFICATEUR LETTRE
                    CL IDENTIFICATEUR CHIFFRE
EXPRESSION ::= EXPRESSION-ARITHMETIQUE
                    CU EXPRESSION-BOCLEENNE
                    OU EXPRESSION-DE-DESIGNATION
EXPRESSION-ARITHMETIQUE ::= EXPRESSION-ARITHMETIQUE-SIMPLE
                    CU PROPOSITION-SI EXPRESSION-ARITHMETIQUE-SIMPLE 'SINCN'
                    EXPRESSION-ARITHMETIQUE
EXPRESSION-BOCLEENNE ::= EXPRESSION-BOCLEENNE-SIMPLE
                    CU PROPOSITION-SI EXPRESSION-BOCLEENNE-SIMPLE 'SINCN' EXPRESSION-BOCLEENNE
EXPRESSION-DE-DESIGNATION ::= EXPRESSION-DE-DESIGNATION-SIMPLE
                    CU PROPOSITION-SI EXPRESSION-DE-DESIGNATION-SIMPLE 'SINCN'
                    EXPRESSION-DE-DESIGNATION
EXPRESSION-ARITHMETIQUE-SIMPLE ::= TERME
                    OU OPERATEUR-ADDITIF TERME
                    CU EXPRESSION-ARITHMETIQUE-SIMPLE OPERATEUR-ADDITIF TERME
OPERATEUR-ADDITIF ::= +
                    CU -
TERME ::= FACTEUR
                    OU TERME OPERATEUR-MULTIPLICATIF FACTEUR
OPERATEUR-MULTIPLICATIF ::= *
                    CU /
                    CU 'OPERATEUR-DE-DIVISION-ENTIERE'
FACTEUR ::= PRIMAIRE-ARITHMETIQUE
                    CL FACTEUR 'PUISSANCE' PRIMAIRE-ARITHMETIQUE
PRIMAIRE-ARITHMETIQUE ::= VARIABLE
                    CU INDICATEUR-DE-FONCTION
                    CU NOMBRE-SANS-SIGNE
                    CU 'PARENTHESE-GAUCHE' EXPRESSION-ARITHMETIQUE 'PARENTHESE-DROITE'
NOMBRE-SANS-SIGNE ::= NOMBRE-DECIMAL
                    OU OPERATEUR-DECIMAL-D'EXPOONENTIATION ENTIER-SANS-SIGNE
                    OU NOMBRE-DECIMAL OPERATEUR-DECIMAL-D'EXPOONENTIATION ENTIER-SANS-SIGNE
OPERATEUR-DECIMAL-D'EXPOONENTIATION ::= '10'
                    CU '10' +
                    OU '10' -
    
```

PL n° 11 GRAMMAIRE ALGOL POUR RECHERCHE D'ERREURS

REGLES (SUITE)

```

NOMBRE-DECIMAL ::= ENTIER-SANS-SIGNE
                OU 'POINT-DECIMAL' ENTIER-SANS-SIGNE
                OU ENTIER-SANS-SIGNE 'POINT-DECIMAL' ENTIER-SANS-SIGNE
ENTIER-SANS-SIGNE ::= CHIFFRE
                OU ENTIER-SANS-SIGNE CHIFFRE
EXPRESSION-BOOLEENNE-SIMPLE ::= IMPLICATION
                OU EXPRESSION-BCCLEENNE-SIMPLE 'EQUIVALENT' IMPLICATION
IMPLICATION ::= TERME-BCCLEEN
                OU IMPLICATION 'IMPLIQUE' TERME-BOOLEEN
TERME-BOOLEEN ::= FACTEUR-BOOLEEN
                OU TERME-BCCLEEN 'OU' FACTEUR-BOOLEEN
FACTEUR-BOOLEEN ::= SECONDAIRE-BOOLEEN
                OU FACTEUR-BOOLEEN 'ET' SECONDAIRE-BOOLEEN
SECONDAIRE-BOOLEEN ::= PRIMAIRE-BCCLEEN
                OU 'NON' PRIMAIRE-BOOLEEN
PRIMAIRE-BCCLEEN ::= VARIABLE
                OU INDICATEUR-DE-FONCTION
                OU VALEUR-LOGIQUE
                OU 'PARENTHESE-GAUCHE' EXPRESSION-BCCLEENNE 'PARENTHESE-DROITE'
                OU EXPRESSION-ARITHMETIQUE-SIMPLE OPERATEUR-DE-RELATION
                OU EXPRESSION-ARITHMETIQUE-SIMPLE
OPERATEUR-DE-RELATION ::= 'VRAI'
                OU 'FAUX'
                ::= 'INFERIEUR'
                OU 'INFERIEUR-CU-EGAL'
                OU 'EGAL'
                OU 'DIFFERENT'
                OU 'SUPERIEUR-CU-EGAL'
                OU 'SUPERIEUR'
VARIABLE ::= IDENTIFICATEUR
                OU IDENTIFICATEUR 'CROCHET-GAUCHE' LISTE-D'EXPRESSIONS-ARITHMETIQUES
                'CROCHET-DROITE'
LISTE-D'EXPRESSIONS-ARITHMETIQUES ::= EXPRESSION-ARITHMETIQUE
                OU EXPRESSION-ARITHMETIQUE , LISTE-D'EXPRESSIONS-ARITHMETIQUES
INDICATEUR-DE-FONCTION ::= IDENTIFICATEUR
                OU IDENTIFICATEUR(PAR.EFFECTIFS)
LISTE-DE-PARAMETRES-EFFECTIFS ::= PARAMETRE-EFFECTIF
                OU
                OU , LISTE-DE-PARAMETRES-EFFECTIFS
                PARAMETRE-EFFECTIF ,
                OU PARAMETRE-EFFECTIF , LISTE-DE-PARAMETRES-EFFECTIFS
PARAMETRE-EFFECTIF ::= IDENTIFICATEUR
                OU EXPRESSION
                OU 'CROCHET-DE-CHAINE-GAUCHE' 'CROCHET-DE-CHAINE-DROITE'
                OU DEUX-APCSTROPHES DEUX-APCSTROPHES
EXPRESSION-DE-DESIGNATION-SIMPLE ::= 'PARENTHESE-GAUCHE' EXPRESSION-DE-DESIGNATION 'PARENTHESE-DROITE'
                OU IDENTIFICATEUR
                OU IDENTIFICATEUR 'CROCHET-GAUCHE' EXPRESSION-ARITHMETIQUE 'CROCHET-DROITE'
LETTRE ::= A
                OU B
                OU C
                OU D
                OU E
                OU F
                OU G
                OU H
                OU I
                OU J
                OU K
                OU L
                OU M
                OU N
                OU O
                OU P
                OU Q
                OU R
                OU S
                OU T
                OU U
                OU V
                OU W
                OU X
                OU Y
                OU Z
CHIFFRE ::= 0
                OU 1
                OU 2
                OU 3
                OU 4
                OU 5
                OU 6
                OU 7
                OU 8
                OU 9
'ALLERA' ::= ' A L L E R A '
                OU ' G O T C '
'SI' ::= ' S I '
                OU ' I F '
'ALORS' ::= ' A L C R S '
                OU ' T H E N '
'SIACN' ::= ' S I N O N '
                OU ' E L S E '
'PCUR' ::= ' P C L R '
                OU ' F C R '
'PAS' ::= ' P A S '
                OU ' S T E P '

```

PL DE 12 GRAMMAIRE ALGOL POUR RECHERCHE D'ERREURS  
REGLES (SUITE)

```

'JUSQUA' ::= ' J U S Q U A '
           OU ' U N T I L '
           OU ' A '
'TANTQUE' ::= ' T A N T Q U E '
           OU ' W H I L E '
'FAIRE'   ::= ' F A I R E '
           OU ' D O '
'REMANENT' ::= ' P E R S I S T A N T '
           OU ' O W N '
           OU ' R E M A N E N T '
'BCCLEEN' ::= ' B O C C L E E N '
           OU ' B O C C L E A N '
'ENTIER'  ::= ' E N T I E R '
           OU ' I N T E G E R '
'REEL'    ::= ' R E E L '
           OU ' R E A L '
'TABLEAU' ::= ' T A B L E A U '
           OU ' A R R A Y '
'AIGUILLAGE' ::= ' A I G U I L L A G E '
           OU ' S W I T C H '
'PROCEDURE' ::= ' P R C C E D U R E '
'CHAINE'   ::= ' C H A I N E '
           OU ' S T R I N G '
'ETIQUETTE' ::= ' E T I Q U E T T E '
           OU ' L A B E L '
'VALEUR'   ::= ' V A L E U R '
           OU ' V A L L E '
'COMMENTAIRE' ::= ' C C M M E N T A I R E '
           OU ' C C M M E N T '
'DEBUT'    ::= ' D E B U T '
           OU ' B E G I N '
'FIN'      ::= ' F I N '
           OU ' E N C '
'CODE'     ::= ' C C C D E '
'FCODE'    ::= ' F C C D E '
'PARENTHESE-GAUCHE' ::= (
'PARENTHESE-DROITE' ::= )
'CROCHET-GAUCHE'   ::= [
           OU CAR-CROCHET-GAUCHE
'CROCHET-DROIT'    ::= ]
           OU CAR-CROCHET-DROIT
'CROCHET-DE-CHAINE-GAUCHE' ::= {
'CROCHET-DE-CHAINE-DROIT' ::= }
DEUX-APCSTROPHES ::= ' '
           OU CAR-GUILLET
'INFERIEUR' ::= ' I N F E R '
           OU ' L E S S '
           OU ' I N F '
           OU CAR-INFERIEUR
'INFERIEUR-OU-EGAL' ::= ' I N F E G '
           OU ' I = '
           OU ' N C T G R E A T E R '
           OU CAR-INFERIEUR =
'EGAL' ::= =
'DIFFERENT' ::= ' N C N E G '
           OU ' N C T E Q U A L '
           OU ' N = '
'SUPERIEUR-OU-EGAL' ::= ' S U P E G '
           OU ' S = '
           OU ' N C T L E S S '
           OU CAR-SUPERIEUR =
'SUPERIEUR' ::= ' S U P E R '
           OU ' G R E A T E R '
           OU ' S U P '
           OU CAR-SUPERIEUR
'OPERATEUR-DE-DIVISION-ENTIERE' ::= /
'PUISSANCE' ::= *
           OU CAR-POINT-D'EXCLAMATION
'10' ::= :
           OU CAR-DELTA
           OU ' 1 0 '
'1' ::= ' 1 '
'EQUIVALENT' ::= ' I D E N T '
           OU ' E Q U I V '
'IMPLIQUE' ::= ' I M P L '
'OU' ::= ' O R '
           OU ' O U '
'ET' ::= ' A N D '
           OU ' E T '
'NON' ::= ' N O N '
           OU ' N O T '
'VRAI' ::= ' V R A I '
           OU ' T R U E '
           OU ' 1 '
'FAUX' ::= ' F A L X '
           OU ' F A L S E '
           OU ' 0 '
'POINT-VIRGULE' ::= ;
           OU CAR-POINT-VIRGULE
'DEUX-POINTS' ::= :
           OU CAR-DEUX-POINTS
'DEUX-POINTS-SEPARATEUR-DE-BCRNES' ::= :
           OU CAR-DEUX-POINTS =
'POINT-DECIMAL' ::= .

```











PL n° 18 EXEMPLE DE RECHERCHE D'ERREURS  
DANS UN PROGRAMME ALGOL

```

2  'DEBUT' 'REEL' I ::
3  'DEBUT' 'TABLEAU' M.(1:1). ::
4  'ENTIER' I,J,K :: 'REEL' X,Y,Z :: 'BOOLEEN' P,Q,R ::
5  'AIGUILLAGE' S=L1 ::
6  'REEL' 'PROCEDURE' A(B,C) :: 'ENTIER' B :: 'ETIQUETTE' C ::
    -----ERREUR. LA SEQUENCE SUIVANTE DELIMITEE PAR LES CARACTERES 1 DE
    LA LIGNE 5 INCLUS ET 17 DE LA LIGNE 5 EXCLUS N A PAS DE SENS:
    'AIGUILLAGE'
    'PRIMAIRE-BOOLEEN'
    ON ATTRIBUE A CETTE SEQUENCE UN SENS COMPATIBLE AVEC LE CONTEXTE.

7  'VALEUR' B ::
8  'DEBUT' 'ENTIER' R ::
    -----ERREUR. LA SEQUENCE SUIVANTE DELIMITEE PAR LES CARACTERES 8 DE
    LA LIGNE 3 INCLUS ET 10 DE LA LIGNE 7 EXCLUS N A PAS DE SENS:
    'SUITE-DE-DECLARATIONS-INCOMPLETE'
    'POINT-VIRGULE'
    'PARTIE-VALEUR'
    ON ATTRIBUE A CETTE SEQUENCE UN SENS COMPATIBLE AVEC LE CONTEXTE.

9  X:=(B**2)**(-2) ::
    -----ERREUR. LA SEQUENCE SUIVANTE DELIMITEE PAR LES CARACTERES 5 DE
    LA LIGNE 9 INCLUS ET 8 DE LA LIGNE 9 EXCLUS N A PAS DE SENS:
    'CHAINE-DE-LETTRES OU IDENTIFICATEUR'
    'PUISSANCE'
    ON ATTRIBUE A CETTE SEQUENCE UN SENS COMPATIBLE AVEC LE CONTEXTE.

10 'ALLERA' 'SI' P 'ALORS' S.(B). 'SINON' L2 ::
11 L1: Y:=P ::
12 'SI' M(5)=X/'Y 'ALORS'
13 'POUR' B=1 'PAS' 0 'JUSQUA' A(B,C) 'FAIRE'
14 L2: 'COMMENTAIRE' MAINTENANT ON VA S'AMUSER ::
    -----ERREUR. LA SEQUENCE SUIVANTE DELIMITEE PAR LES CARACTERES 7 DE
    LA LIGNE 13 INCLUS ET 30 DE LA LIGNE 13 EXCLUS N A PAS DE SENS:
    'PRIMAIRE-BOOLEEN'
    'PAS'
    'ENTIER-SANS-SIGNE'
    'JUSQUA'
    'IDENTIFICATEUR(PAR.FORMELS) OU IDENTIFICATEUR(PAR.EFFECTIFS)'
    ON ATTRIBUE A CETTE SEQUENCE UN SENS COMPATIBLE AVEC LE CONTEXTE.

15 'DEBUT' 'REMANENT' 'TABLEAU'
16 N.('SI' R 'IDENT' 'VRAI' 'ALORS' 2:10 'SINON' 1:10). ::
    -----ERREUR. LA SEQUENCE SUIVANTE DELIMITEE PAR LES CARACTERES 4 DE
    LA LIGNE 16 INCLUS ET 41 DE LA LIGNE 16 EXCLUS N A PAS DE SENS:
    'PROPOSITION-SI'
    'PAIRE-DE-BORNES'
    'SINON'
    'ENTIER-SANS-SIGNE'
    ON ATTRIBUE A CETTE SEQUENCE UN SENS COMPATIBLE AVEC LE CONTEXTE.

17 'SI' X 'NONEG' 0 'ALORS' 'SI' Y=1 'OU' 2 'ALORS' 'ALLERA' C
    -----ERREUR. LA SEQUENCE SUIVANTE DELIMITEE PAR LES CARACTERES 25 DE
    LA LIGNE 17 INCLUS ET 33 DE LA LIGNE 17 EXCLUS N A PAS DE SENS:
    'PRIMAIRE-BOOLEEN'
    'OU'
    'ENTIER-SANS-SIGNE'
    ON ATTRIBUE A CETTE SEQUENCE UN SENS COMPATIBLE AVEC LE CONTEXTE.

18 'FIN' ::
    -----ERREUR. LA SEQUENCE SUIVANTE DELIMITEE PAR LES CARACTERES 1 DE
    LA LIGNE 17 INCLUS ET 1 DE LA LIGNE 18 EXCLUS N A PAS DE SENS:
    'PROPOSITION-SI'
    'PROPOSITION-SI'
    'INSTRUCTION-ALLER-A'
    ON ATTRIBUE A CETTE SEQUENCE UN SENS COMPATIBLE AVEC LE CONTEXTE.

19 'RETOUR' 'FIN' DE LA PROCEDURE A ::
    -----ERREUR. LA SEQUENCE SUIVANTE DELIMITEE PAR LES CARACTERES 1 DE
    LA LIGNE 13 INCLUS ET 6 DE LA LIGNE 18 EXCLUS N A PAS DE SENS:
    'POUR'
    'SEQUENCE CITEE DANS UNE ERREUR PRECEDENTE'
    'FAIRE'
    'INSTRUCTION-DE-BASE'
    'COMMENTAIRE'
    'POINT-VIRGULE'
    'BLOC'
    ON ATTRIBUE A CETTE SEQUENCE UN SENS COMPATIBLE AVEC LE CONTEXTE.
    -----ERREUR. LA SEQUENCE SUIVANTE DELIMITEE PAR LES CARACTERES 1 DE
    LA LIGNE 19 INCLUS ET 9 DE LA LIGNE 19 EXCLUS N A PAS DE SENS:
    'R'
    'E'
    'T'
    'C'
    'U'
    'R'
    CETTE SEQUENCE EST SUPPRIMEE DU PROGRAMME.

20 X:=I:=0 ::
21 'SI' X=0 'ALORS'
22 'POUR' Q:= 'VRAI', 'FAUX' 'FAIRE'
23 L3: Z:=(A(B)LE 2EME PARAMETRE EST:(L3) 'SINON' 'FIN'
    -----ERREUR. LA SEQUENCE SUIVANTE DELIMITEE PAR LES CARACTERES 7 DE
    LA LIGNE 23 INCLUS ET 29 DE LA LIGNE 23 EXCLUS N A PAS DE SENS:
    'CHAINE-DE-LETTRES OU IDENTIFICATEUR'
    'PARENTHESE-GAUCHE'
    'CHAINE-DE-LETTRES OU IDENTIFICATEUR'
    'PARENTHESE-DROITE'
    'IDENTIFICATEUR'
    ON ATTRIBUE A CETTE SEQUENCE UN SENS COMPATIBLE AVEC LE CONTEXTE.

24 'FIN' DU PROGRAMME ::
    -----ERREUR. LA SEQUENCE SUIVANTE DELIMITEE PAR LES CARACTERES 1 DE
    LA LIGNE 22 INCLUS ET 41 DE LA LIGNE 23 EXCLUS N A PAS DE SENS:
    'POUR'
    'CHAINE-DE-LETTRES OU IDENTIFICATEUR'
    'LISTE-DE-PARAMETRES-EFFECTIFS'
    'FAIRE'
    'INSTRUCTION-DE-BASE'
    'SINON'
    ON ATTRIBUE A CETTE SEQUENCE UN SENS COMPATIBLE AVEC LE CONTEXTE.

```

L ANALYSE DE CE PROGRAMME A DURE 0 MINUTES ET 7 SECONDES.

### 5.3 Précédence de gauche à droite et Algol de N. Wirth et C.A.R. Hoare

N. Wirth et C.A.R. Hoare [WH] ont proposé un nouvel Algol plus complet en vue de remplacer Algol 60. Cet Algol est décrit à l'aide d'une grammaire "context-free" et d'un certain nombre de petits tableaux spécifiant le type d'un résultat obtenu par une opération portant sur deux opérandes. Par exemple la somme d'un "entier" et d'un "réel-long" et un "réel-long". Nous avons supprimé ces tableaux en rajoutant à la grammaire un certain nombre de règles permettant d'exprimer toutes les contraintes exprimées dans ceux-ci. Nous avons transformé cette nouvelle grammaire en une grammaire équivalente analysable par l'algorithme décrit au paragraphe 4.5.

Voici successivement, la grammaire obtenue, ses relations de précédence de gauche à droite, et la relation  $\Rightarrow^*$  entre ses opérandes.



## REGLES (SUITE)

```

LONG-REAL-PROCEDURE-BODY ::= LONG-REAL-EXPRESSION
                           OR 'BEGIN' LONG-REAL-PROCEDURE-BODY-INTERIOR 'END'
SHORT-COMPLEX-PROCEDURE-BODY ::= SHORT-COMPLEX-EXPRESSION
                              OR 'BEGIN' SHORT-COMPLEX-PROCEDURE-BODY-INTERIOR 'END'
LONG-COMPLEX-PROCEDURE-BODY ::= LONG-COMPLEX-EXPRESSION
                              OR 'BEGIN' LONG-COMPLEX-PROCEDURE-BODY-INTERIOR 'END'
LOGICAL-PROCEDURE-BODY ::= LOGICAL-EXPRESSION
                          OR 'BEGIN' LOGICAL-PROCEDURE-BODY-INTERIOR 'END'
BIT-PROCEDURE-BODY ::= BIT-EXPRESSION
                    OR 'BEGIN' BIT-PROCEDURE-BODY-INTERIOR 'END'
STRING-PROCEDURE-BODY ::= STRING-EXPRESSION
                       OR 'BEGIN' STRING-PROCEDURE-BODY-INTERIOR 'END'
REFERENCE-PROCEDURE-BODY ::= REFERENCE-EXPRESSION
                           OR 'BEGIN' REFERENCE-PROCEDURE-BODY-INTERIOR 'END'
INTEGER-PROCEDURE-BODY-INTERIOR ::= INTEGER-EXPRESSION
                                  OR .. INTEGER-EXPRESSION
                                  OR 'ALSO' INTEGER-EXPRESSION
                                  OR BLOCK-INTERIOR-OR-DECLARATION-PART .. INTEGER-EXPRESSION
                                  OR BLOCK-INTERIOR 'ALSO' INTEGER-EXPRESSION
                                  OR INCOMPLETE-BLOCK-INTERIOR : INTEGER-EXPRESSION
SHORT-REAL-PROCEDURE-BODY-INTERIOR ::= SHORT-REAL-EXPRESSION
                                      OR .. SHORT-REAL-EXPRESSION
                                      OR 'ALSO' SHORT-REAL-EXPRESSION
                                      OR BLOCK-INTERIOR-OR-DECLARATION-PART .. SHORT-REAL-EXPRESSION
                                      OR BLOCK-INTERIOR 'ALSO' SHORT-REAL-EXPRESSION
                                      OR INCOMPLETE-BLOCK-INTERIOR : SHORT-REAL-EXPRESSION
LONG-REAL-PROCEDURE-BODY-INTERIOR ::= LONG-REAL-EXPRESSION
                                    OR .. LONG-REAL-EXPRESSION
                                    OR 'ALSO' LONG-REAL-EXPRESSION
                                    OR BLOCK-INTERIOR-OR-DECLARATION-PART .. LONG-REAL-EXPRESSION
                                    OR BLOCK-INTERIOR 'ALSO' LONG-REAL-EXPRESSION
                                    OR INCOMPLETE-BLOCK-INTERIOR : LONG-REAL-EXPRESSION
SHORT-COMPLEX-PROCEDURE-BODY-INTERIOR ::= SHORT-COMPLEX-EXPRESSION
                                         OR .. SHORT-COMPLEX-EXPRESSION
                                         OR 'ALSO' SHORT-COMPLEX-EXPRESSION
                                         OR BLOCK-INTERIOR-OR-DECLARATION-PART .. SHORT-COMPLEX-EXPRESSION
                                         OR BLOCK-INTERIOR 'ALSO' SHORT-COMPLEX-EXPRESSION
                                         OR INCOMPLETE-BLOCK-INTERIOR : SHORT-COMPLEX-EXPRESSION
LONG-COMPLEX-PROCEDURE-BODY-INTERIOR ::= LONG-COMPLEX-EXPRESSION
                                        OR .. LONG-COMPLEX-EXPRESSION
                                        OR 'ALSO' LONG-COMPLEX-EXPRESSION
                                        OR BLOCK-INTERIOR-OR-DECLARATION-PART .. LONG-COMPLEX-EXPRESSION
                                        OR BLOCK-INTERIOR 'ALSO' LONG-COMPLEX-EXPRESSION
                                        OR INCOMPLETE-BLOCK-INTERIOR : LONG-COMPLEX-EXPRESSION
LOGICAL-PROCEDURE-BODY-INTERIOR ::= LOGICAL-EXPRESSION
                                   OR .. LOGICAL-EXPRESSION
                                   OR 'ALSO' LOGICAL-EXPRESSION
                                   OR BLOCK-INTERIOR-OR-DECLARATION-PART .. LOGICAL-EXPRESSION
                                   OR BLOCK-INTERIOR 'ALSO' LOGICAL-EXPRESSION
                                   OR INCOMPLETE-BLOCK-INTERIOR : LOGICAL-EXPRESSION
BIT-PROCEDURE-BODY-INTERIOR ::= BIT-EXPRESSION
                              OR .. BIT-EXPRESSION
                              OR 'ALSO' BIT-EXPRESSION
                              OR BLOCK-INTERIOR-OR-DECLARATION-PART .. BIT-EXPRESSION
                              OR BLOCK-INTERIOR 'ALSO' BIT-EXPRESSION
                              OR INCOMPLETE-BLOCK-INTERIOR : BIT-EXPRESSION
STRING-PROCEDURE-BODY-INTERIOR ::= STRING-EXPRESSION
                                  OR .. STRING-EXPRESSION
                                  OR 'ALSO' STRING-EXPRESSION
                                  OR BLOCK-INTERIOR-OR-DECLARATION-PART .. STRING-EXPRESSION
                                  OR BLOCK-INTERIOR 'ALSO' STRING-EXPRESSION
                                  OR INCOMPLETE-BLOCK-INTERIOR : STRING-EXPRESSION
REFERENCE-PROCEDURE-BODY-INTERIOR ::= REFERENCE-EXPRESSION
                                     OR .. REFERENCE-EXPRESSION
                                     OR 'ALSO' REFERENCE-EXPRESSION
                                     OR BLOCK-INTERIOR-OR-DECLARATION-PART .. REFERENCE-EXPRESSION
                                     OR BLOCK-INTERIOR 'ALSO' REFERENCE-EXPRESSION
                                     OR INCOMPLETE-BLOCK-INTERIOR : REFERENCE-EXPRESSION
FORMAL-PARAMETER-LIST ::= FORMAL-PARAMETER-SEGMENT
                         OR FORMAL-PARAMETER-LIST .. FORMAL-PARAMETER-SEGMENT
FORMAL-PARAMETER-SEGMENT ::= SIMPLE-TYPE-FOLLOWED-BY-IDENTIFIER
                            OR TYPE 'ARRAY' IDENTIFIER
                            OR 'PROCEDURE' IDENTIFIER
                            OR SIMPLE-TYPE 'PROCEDURE' IDENTIFIER
                            OR SIMPLE-TYPE 'VALUE' IDENTIFIER
                            OR SIMPLE-TYPE 'RESULT' IDENTIFIER
                            OR SIMPLE-TYPE 'VALUE' 'RESULT' IDENTIFIER
                            OR FORMAL-PARAMETER-SEGMENT , IDENTIFIER
TYPE ::= SIMPLE-TYPE
       OR TYPE 'ARRAY'
SIMPLE-TYPE ::= SIMPLE-INTEGER-TYPE
              OR SIMPLE-SHORT-REAL-TYPE
              OR SIMPLE-LONG-REAL-TYPE
              OR SIMPLE-SHORT-COMPLEX-TYPE
              OR SIMPLE-LONG-COMPLEX-TYPE
              OR SIMPLE-LOGICAL-TYPE
              OR SIMPLE-BIT-TYPE
              OR SIMPLE-STRING-TYPE
              OR SIMPLE-REFERENCE-TYPE
SIMPLE-INTEGER-TYPE ::= 'INTEGER'
SIMPLE-SHORT-REAL-TYPE ::= 'REAL'
SIMPLE-LONG-REAL-TYPE ::= 'LONG' 'REAL'
SIMPLE-SHORT-COMPLEX-TYPE ::= 'COMPLEX'
SIMPLE-LONG-COMPLEX-TYPE ::= 'LONG' 'COMPLEX'
SIMPLE-LOGICAL-TYPE ::= 'LOGICAL'
SIMPLE-BIT-TYPE ::= 'BITS'
SIMPLE-STRING-TYPE ::= 'BITS' ( INTEGER-NUMBER )
                   OR 'STRING'

```



## REGLES (SUITE)

```

INTEGER-EXPRESSION ::= SIMPLE-INTEGGER-EXPRESSION
OR CASE-CLAUSE ( INTEGER-EXPRESSION-LIST )
OR IF-CLAUSE SIMPLE-INTEGGER-EXPRESSION 'ELSE' INTEGER-EXPRESSION
SHORT-REAL-EXPRESSION ::= SIMPLE-SHORT-REAL-EXPRESSION
OR CASE-CLAUSE ( SHORT-REAL-EXPRESSION-LIST )
OR IF-CLAUSE SIMPLE-SHORT-REAL-EXPRESSION 'ELSE'
NUMERICAL-NON-COMPLEX-EXPRESSION
OR IF-CLAUSE SIMPLE-INTEGGER-OR-LONG-REAL-EXPRESSION 'ELSE'
SHORT-REAL-EXPRESSION
LONG-REAL-EXPRESSION ::= SIMPLE-LONG-REAL-EXPRESSION
OR CASE-CLAUSE ( LONG-REAL-EXPRESSION-LIST )
OR IF-CLAUSE SIMPLE-LONG-REAL-EXPRESSION 'ELSE' INTEGER-EXPRESSION
OR IF-CLAUSE SIMPLE-INTEGGER-OR-LONG-REAL-EXPRESSION 'ELSE'
LONG-REAL-EXPRESSION
SHORT-COMPLEX-EXPRESSION ::= SIMPLE-SHORT-COMPLEX-EXPRESSION
OR CASE-CLAUSE ( SHORT-COMPLEX-EXPRESSION-LIST )
OR IF-CLAUSE SIMPLE-SHORT-COMPLEX-EXPRESSION 'ELSE'
NUMERICAL-EXPRESSION
OR IF-CLAUSE SIMPLE-NUMERICAL-NON-SHORT-COMPLEX-EXPRESSION 'ELSE'
SHORT-COMPLEX-EXPRESSION
OR IF-CLAUSE SIMPLE-LONG-COMPLEX-EXPRESSION 'ELSE'
SHORT-REAL-EXPRESSION
OR IF-CLAUSE SIMPLE-SHORT-REAL-EXPRESSION 'ELSE'
LONG-COMPLEX-EXPRESSION
LONG-COMPLEX-EXPRESSION ::= SIMPLE-LONG-COMPLEX-EXPRESSION
OR CASE-CLAUSE ( LONG-COMPLEX-EXPRESSION-LIST )
OR IF-CLAUSE SIMPLE-LONG-COMPLEX-EXPRESSION 'ELSE'
INTEGER-OR-LONG-NUMERICAL-EXPRESSION
OR IF-CLAUSE SIMPLE-INTEGGER-OR-LONG-REAL-EXPRESSION 'ELSE'
LONG-COMPLEX-EXPRESSION
LOGICAL-EXPRESSION ::= SIMPLE-LOGICAL-EXPRESSION
OR CASE-CLAUSE ( LOGICAL-EXPRESSION-LIST )
OR IF-CLAUSE SIMPLE-LOGICAL-EXPRESSION 'ELSE' LOGICAL-EXPRESSION
BIT-EXPRESSION ::= SIMPLE-BIT-EXPRESSION
OR CASE-CLAUSE ( BIT-EXPRESSION-LIST )
OR IF-CLAUSE SIMPLE-BIT-EXPRESSION 'ELSE' BIT-EXPRESSION
STRING-EXPRESSION ::= SIMPLE-STRING-EXPRESSION
OR CASE-CLAUSE ( STRING-EXPRESSION-LIST )
OR IF-CLAUSE SIMPLE-STRING-EXPRESSION 'ELSE' STRING-EXPRESSION
REFERENCE-EXPRESSION ::= SIMPLE-REFERENCE-EXPRESSION
OR CASE-CLAUSE ( REFERENCE-EXPRESSION-LIST )
OR IF-CLAUSE SIMPLE-REFERENCE-EXPRESSION 'ELSE' REFERENCE-EXPRESSION
NUMERICAL-NON-SHORT-COMPLEX-EXPRESSION-LIST ::= SHORT-REAL-EXPRESSION-LIST
OR LONG-COMPLEX-EXPRESSION-LIST
OR INTEGER-OR-LONG-REAL-EXPRESSION-LIST
INTEGER-OR-LONG-REAL-EXPRESSION-LIST ::= INTEGER-EXPRESSION-LIST
OR LONG-REAL-EXPRESSION-LIST
INTEGER-EXPRESSION-LIST ::= INTEGER-EXPRESSION
OR INTEGER-EXPRESSION-LIST , INTEGER-EXPRESSION
SHORT-REAL-EXPRESSION-LIST ::= SHORT-REAL-EXPRESSION
OR SHORT-REAL-EXPRESSION-LIST , NUMERICAL-NON-COMPLEX-EXPRESSION
OR INTEGER-OR-LONG-REAL-EXPRESSION-LIST , SHORT-REAL-EXPRESSION
LONG-REAL-EXPRESSION-LIST ::= LONG-REAL-EXPRESSION
OR LONG-REAL-EXPRESSION-LIST , INTEGER-EXPRESSION
OR INTEGER-OR-LONG-REAL-EXPRESSION-LIST , LONG-REAL-EXPRESSION
SHORT-COMPLEX-EXPRESSION-LIST ::= SHORT-COMPLEX-EXPRESSION
OR SHORT-COMPLEX-EXPRESSION-LIST , NUMERICAL-EXPRESSION
OR NUMERICAL-NON-SHORT-COMPLEX-EXPRESSION-LIST ,
SHORT-COMPLEX-EXPRESSION
OR SHORT-REAL-EXPRESSION-LIST , LONG-COMPLEX-EXPRESSION
OR LONG-COMPLEX-EXPRESSION-LIST , SHORT-REAL-EXPRESSION
LONG-COMPLEX-EXPRESSION-LIST ::= LONG-COMPLEX-EXPRESSION
OR LONG-COMPLEX-EXPRESSION-LIST ,
INTEGER-OR-LONG-NUMERICAL-EXPRESSION
OR INTEGER-OR-LONG-REAL-EXPRESSION-LIST , LONG-COMPLEX-EXPRESSION
LOGICAL-EXPRESSION-LIST ::= LOGICAL-EXPRESSION
OR LOGICAL-EXPRESSION-LIST , LOGICAL-EXPRESSION
BIT-EXPRESSION-LIST ::= BIT-EXPRESSION
OR BIT-EXPRESSION-LIST , BIT-EXPRESSION
STRING-EXPRESSION-LIST ::= STRING-EXPRESSION
OR STRING-EXPRESSION-LIST , STRING-EXPRESSION
REFERENCE-EXPRESSION-LIST ::= REFERENCE-EXPRESSION
OR REFERENCE-EXPRESSION-LIST , REFERENCE-EXPRESSION
SIMPLE-NUMERICAL-EXPRESSION ::= SIMPLE-SHORT-COMPLEX-EXPRESSION
OR SIMPLE-NUMERICAL-NON-SHORT-COMPLEX-EXPRESSION
SIMPLE-NUMERICAL-NON-SHORT-COMPLEX-EXPRESSION ::= SIMPLE-LONG-COMPLEX-EXPRESSION
OR SIMPLE-NUMERICAL-NON-COMPLEX-EXPRESSION
SIMPLE-NUMERICAL-NON-COMPLEX-EXPRESSION ::= SIMPLE-SHORT-REAL-EXPRESSION
OR SIMPLE-INTEGGER-OR-LONG-REAL-EXPRESSION
SIMPLE-INTEGGER-OR-LONG-REAL-EXPRESSION ::= SIMPLE-INTEGGER-EXPRESSION
OR SIMPLE-LONG-REAL-EXPRESSION
SIMPLE-INTEGGER-EXPRESSION ::= INTEGER-TERM
OR SIGN INTEGER-TERM
SIMPLE-SHORT-REAL-EXPRESSION ::= SIMPLE-INTEGGER-EXPRESSION SIGN INTEGER-TERM
OR SHORT-REAL-TERM
OR SIGN SHORT-REAL-TERM
OR SIMPLE-SHORT-REAL-EXPRESSION SIGN NUMERICAL-NON-COMPLEX-TERM
OR SIMPLE-INTEGGER-OR-LONG-REAL-EXPRESSION SIGN SHORT-REAL-TERM
SIMPLE-LONG-REAL-EXPRESSION ::= LONG-REAL-TERM
OR SIGN LONG-REAL-TERM
OR SIMPLE-LONG-REAL-EXPRESSION SIGN INTEGER-TERM
OR SIMPLE-INTEGGER-OR-LONG-REAL-EXPRESSION SIGN LONG-REAL-TERM
SIMPLE-SHORT-COMPLEX-EXPRESSION ::= SHORT-COMPLEX-TERM
OR SIGN SHORT-COMPLEX-TERM
OR SIMPLE-SHORT-COMPLEX-EXPRESSION SIGN
NUMERICAL-NON-SHORT-COMPLEX-TERM
OR SIMPLE-NUMERICAL-EXPRESSION SIGN SHORT-COMPLEX-TERM
OR SIMPLE-LONG-COMPLEX-EXPRESSION SIGN SHORT-REAL-TERM
OR SIMPLE-SHORT-REAL-EXPRESSION SIGN LONG-COMPLEX-TERM

```

## REGLES (SUITE)

```

SIMPLE-LONG-COMPLEX-EXPRESSION ::= LONG-COMPLEX-TERM
OR SIGN LONG-COMPLEX-TERM
OR SIMPLE-LONG-COMPLEX-EXPRESSION SIGN INTEGER-OR-LONG-NUMERICAL-TERM
OR SIMPLE-INTEGER-OR-LONG-REAL-EXPRESSION SIGN LONG-COMPLEX-TERM
SIGN ::= +
OR -
NUMERICAL-NON-SHORT-COMPLEX-TERM ::= LCNG-COMPLEX-TERM
OR NUMERICAL-NON-COMPLEX-TERM
NUMERICAL-NON-COMPLEX-TERM ::= INTEGER-TERM
OR REAL-TERM
INTEGER-OR-LONG-NUMERICAL-TERM ::= INTEGER-TERM
OR LCNG-REAL-TERM
OR LCNG-COMPLEX-TERM
REAL-TERM ::= SHORT-REAL-TERM
OR LCNG-REAL-TERM
COMPLEX-TERM ::= SHORT-COMPLEX-TERM
OR LONG-COMPLEX-TERM
INTEGER-TERM ::= INTEGER-FACTOR
OR INTEGER-TERM * INTEGER-FACTOR
OR INTEGER-TERM 'DIV' INTEGER-FACTOR
OR INTEGER-TERM 'REM' INTEGER-FACTOR
SHORT-REAL-TERM ::= SHORT-REAL-FACTOR
OR NUMERICAL-NON-COMPLEX-TERM / SHORT-REAL-FACTOR
OR SHORT-REAL-TERM / INTEGER-OR-LCNG-REAL-FACTOR
OR INTEGER-TERM / INTEGER-FACTOR
LONG-REAL-TERM ::= LONG-REAL-FACTOR
OR NUMERICAL-NON-COMPLEX-TERM * REAL-FACTOR
OR REAL-TERM * INTEGER-FACTOR
OR LONG-REAL-TERM / INTEGER-OR-LONG-REAL-FACTOR
OR INTEGER-TERM / LONG-REAL-FACTOR
SHORT-COMPLEX-TERM ::= SHORT-COMPLEX-FACTOR
OR SHORT-COMPLEX-TERM / NUMERICAL-FACTOR
OR NUMERICAL-NON-SHORT-COMPLEX-TERM / SHORT-COMPLEX-FACTOR
OR LONG-COMPLEX-TERM / SHORT-REAL-FACTOR
OR SHORT-REAL-TERM / LONG-COMPLEX-FACTOR
LONG-COMPLEX-TERM ::= LONG-COMPLEX-FACTOR
OR COMPLEX-TERM * NUMERICAL-FACTOR
OR NUMERICAL-NON-COMPLEX-TERM * COMPLEX-FACTOR
OR LONG-COMPLEX-TERM / INTEGER-OR-LCNG-REAL-FACTOR
OR INTEGER-OR-LONG-NUMERICAL-TERM / LONG-COMPLEX-FACTOR
NUMERICAL-FACTOR ::= INTEGER-FACTOR
OR REAL-FACTOR
OR COMPLEX-FACTOR
INTEGER-OR-LONG-REAL-FACTOR ::= INTEGER-FACTOR
OR LONG-REAL-FACTOR
REAL-FACTOR ::= SHORT-REAL-FACTOR
OR LONG-REAL-FACTOR
COMPLEX-FACTOR ::= SHORT-COMPLEX-FACTOR
OR LONG-COMPLEX-FACTOR
INTEGER-FACTOR ::= INTEGER-SECONDARY
SHORT-REAL-FACTOR ::= SHORT-REAL-SECONDARY
OR INTEGER-FACTOR ** INTEGER-SECONDARY
OR SHORT-REAL-FACTOR ** INTEGER-SECONDARY
LCNG-REAL-FACTOR ::= LONG-REAL-SECONDARY
OR LONG-REAL-FACTOR ** INTEGER-SECONDARY
SHORT-COMPLEX-FACTOR ::= SHORT-COMPLEX-SECONDARY
OR SHORT-COMPLEX-FACTOR ** INTEGER-SECONDARY
LONG-COMPLEX-FACTOR ::= LONG-COMPLEX-SECONDARY
OR LONG-COMPLEX-FACTOR ** INTEGER-SECONDARY
INTEGER-SECONDARY ::= INTEGER-PRIMARY
OR INTEGER-NUMBER
OR 'ABS' INTEGER-PRIMARY
SHORT-REAL-SECONDARY ::= SHORT-REAL-PRIMARY
OR SHORT-REAL-NUMBER
OR 'ABS' SHORT-REAL-PRIMARY
OR 'ABS' SHORT-COMPLEX-PRIMARY
LONG-REAL-SECONDARY ::= LONG-REAL-PRIMARY
OR LCNG-REAL-NUMBER
OR 'ABS' LONG-REAL-PRIMARY
OR 'ABS' LONG-COMPLEX-PRIMARY
SHORT-COMPLEX-SECONDARY ::= SHORT-COMPLEX-PRIMARY
OR SHORT-COMPLEX-NUMBER
LONG-COMPLEX-SECONDARY ::= LONG-COMPLEX-PRIMARY
OR LONG-COMPLEX-NUMBER
INTEGER-PRIMARY ::= INTEGER-VARIABLE
OR INTEGER-FUNCTION-IDENTIFIER
OR INTEGER-FUNCTION-IDENTIFIER ( )
OR INTEGER-FUNCTION-IDENTIFIER ( ACTUAL-PARAMETER-LIST )
OR ( INTEGER-EXPRESSION )
SHORT-REAL-PRIMARY ::= SHORT-REAL-VARIABLE
OR SHORT-REAL-FUNCTION-IDENTIFIER
OR SHORT-REAL-FUNCTION-IDENTIFIER ( )
OR SHORT-REAL-FUNCTION-IDENTIFIER ( ACTUAL-PARAMETER-LIST )
OR ( SHORT-REAL-EXPRESSION )
LONG-REAL-PRIMARY ::= LONG-REAL-VARIABLE
OR LONG-REAL-FUNCTION-IDENTIFIER
OR LONG-REAL-FUNCTION-IDENTIFIER ( )
OR LONG-REAL-FUNCTION-IDENTIFIER ( ACTUAL-PARAMETER-LIST )
OR 'LCNG' SHORT-REAL-PRIMARY
OR 'LCNG' LCNG-REAL-PRIMARY
OR ( LCNG-REAL-EXPRESSION )
SHORT-COMPLEX-PRIMARY ::= SHORT-COMPLEX-VARIABLE
OR SHORT-COMPLEX-FUNCTION-IDENTIFIER
OR SHORT-COMPLEX-FUNCTION-IDENTIFIER ( )
OR SHORT-COMPLEX-FUNCTION-IDENTIFIER ( ACTUAL-PARAMETER-LIST )
OR ( SHORT-COMPLEX-EXPRESSION )

```

PL no 24 GRAMMAIRE D'UN NOUVEL ALGOL  
 REGLES (SUITE)

```

LONG-COMPLEX-PRIMARY ::= LONG-COMPLEX-VARIABLE
OR LONG-COMPLEX-FUNCTION-IDENTIFIER
OR LONG-COMPLEX-FUNCTION-IDENTIFIER ( )
OR LONG-COMPLEX-FUNCTION-IDENTIFIER ( ACTUAL-PARAMETER-LIST )
OR 'LONG' SHORT-COMPLEX-PRIMARY
OR 'LONG' LONG-COMPLEX-PRIMARY
OR ( LONG-COMPLEX-EXPRESSION )

SIMPLE-LOGICAL-EXPRESSION ::= LOGICAL-TERM
OR RELATION

RELATION ::= LOGICAL-TERM EQUALITY-OPERATOR LOGICAL-TERM
OR SIMPLE-BIT-EXPRESSION EQUALITY-OPERATOR SIMPLE-BIT-EXPRESSION
OR SIMPLE-STRING-EXPRESSION EQUALITY-OPERATOR
SIMPLE-STRING-EXPRESSION
OR SIMPLE-REFERENCE-EXPRESSION EQUALITY-OPERATOR
SIMPLE-REFERENCE-EXPRESSION
OR SIMPLE-NUMERICAL-EXPRESSION EQUALITY-OPERATOR
SIMPLE-NUMERICAL-EXPRESSION
OR SIMPLE-NUMERICAL-NON-COMPLEX-EXPRESSION ORDER-RELATION-OPERATOR
SIMPLE-NUMERICAL-NON-COMPLEX-EXPRESSION

EQUALITY-OPERATOR ::= =
OR 'NCT='

ORDER-RELATION-OPERATOR ::= 'GREATER'
OR 'NCTGREATER'
OR 'LESS'
OR 'NCTLESS'

LOGICAL-TERM ::= LOGICAL-FACTOR
OR LOGICAL-TERM 'OR' LOGICAL-FACTOR

LOGICAL-FACTOR ::= LOGICAL-SECONDARY
OR LOGICAL-FACTOR 'AND' LOGICAL-SECONDARY

LOGICAL-SECONDARY ::= LOGICAL-PRIMARY
OR 'NCT' LOGICAL-PRIMARY

LOGICAL-PRIMARY ::= 'TRUE'
OR 'FALSE'
OR LOGICAL-VARIABLE
OR LOGICAL-FUNCTION-IDENTIFIER
OR LOGICAL-FUNCTION-IDENTIFIER ( )
OR LOGICAL-FUNCTION-IDENTIFIER ( ACTUAL-PARAMETER-LIST )
OR ( LOGICAL-EXPRESSION )

SIMPLE-BIT-EXPRESSION ::= BIT-TERM
OR SIMPLE-BIT-EXPRESSION 'OR' BIT-TERM

BIT-TERM ::= BIT-FACTOR
OR BIT-TERM 'AND' BIT-FACTOR

BIT-FACTOR ::= BIT-SECONDARY
OR 'NOT' BIT-SECONDARY

BIT-SECONDARY ::= BIT-PRIMARY
OR BIT-SECONDARY ** INTEGER-SECONDARY
OR BIT-SECONDARY // INTEGER-SECONDARY

BIT-PRIMARY ::= BIT-SEQUENCE
OR BIT-VARIABLE
OR BIT-FUNCTION-IDENTIFIER
OR BIT-FUNCTION-IDENTIFIER ( )
OR BIT-FUNCTION-IDENTIFIER ( ACTUAL-PARAMETER-LIST )
OR ( BIT-EXPRESSION )

BIT-SEQUENCE ::= 'B' BIT
OR BIT-SEQUENCE BIT

SIMPLE-STRING-EXPRESSION ::= STRING-PRIMARY
OR SIMPLE-STRING-EXPRESSION 'CAT' STRING-PRIMARY

STRING-PRIMARY ::= ' '
OR STRING-VARIABLE
OR STRING-FUNCTION-IDENTIFIER
OR STRING-FUNCTION-IDENTIFIER ( )
OR STRING-FUNCTION-IDENTIFIER ( ACTUAL-PARAMETER-LIST )
OR ( STRING-EXPRESSION )

SIMPLE-REFERENCE-EXPRESSION ::= 'NULL'
OR REFERENCE-VARIABLE
OR REFERENCE-FUNCTION-IDENTIFIER
OR REFERENCE-FUNCTION-IDENTIFIER ( )
OR REFERENCE-FUNCTION-IDENTIFIER ( ACTUAL-PARAMETER-LIST )
OR RECORD-CLASS-IDENTIFIER
OR RECORD-CLASS-IDENTIFIER ( EXPRESSION-LIST )
OR ( REFERENCE-EXPRESSION )

EXPRESSION-LIST ::= EXPRESSION
OR EXPRESSION-LIST , EXPRESSION

INTEGER-VARIABLE ::= INTEGER-VARIABLE-IDENTIFIER
OR INTEGER-FIELD-IDENTIFIER ( REFERENCE-EXPRESSION )
OR INTEGER-ARRAY-DESIGNATOR .( INTEGER-EXPRESSION ).

REAL-VARIABLE ::= SHORT-REAL-VARIABLE
OR LONG-REAL-VARIABLE

COMPLEX-VARIABLE ::= SHORT-COMPLEX-VARIABLE
OR LONG-COMPLEX-VARIABLE

SHORT-REAL-VARIABLE ::= SHORT-REAL-VARIABLE-IDENTIFIER
OR SHORT-REAL-FIELD-IDENTIFIER ( REFERENCE-EXPRESSION )
OR SHORT-REAL-ARRAY-DESIGNATOR .( INTEGER-EXPRESSION ).

LONG-REAL-VARIABLE ::= LONG-REAL-VARIABLE-IDENTIFIER
OR LONG-REAL-FIELD-IDENTIFIER ( REFERENCE-EXPRESSION )
OR LONG-REAL-ARRAY-DESIGNATOR .( INTEGER-EXPRESSION ).

SHORT-COMPLEX-VARIABLE ::= SHORT-COMPLEX-VARIABLE-IDENTIFIER
OR SHORT-COMPLEX-FIELD-IDENTIFIER ( REFERENCE-EXPRESSION )
OR SHORT-COMPLEX-ARRAY-DESIGNATOR .( INTEGER-EXPRESSION ).

LONG-COMPLEX-VARIABLE ::= LONG-COMPLEX-VARIABLE-IDENTIFIER
OR LONG-COMPLEX-FIELD-IDENTIFIER ( REFERENCE-EXPRESSION )
OR LONG-COMPLEX-ARRAY-DESIGNATOR .( INTEGER-EXPRESSION ).

LOGICAL-VARIABLE ::= LOGICAL-VARIABLE-IDENTIFIER
OR LOGICAL-FIELD-IDENTIFIER ( REFERENCE-EXPRESSION )
OR LOGICAL-ARRAY-DESIGNATOR .( INTEGER-EXPRESSION ).

BIT-VARIABLE ::= BIT-VARIABLE-IDENTIFIER
OR BIT-FIELD-IDENTIFIER ( REFERENCE-EXPRESSION )
OR BIT-ARRAY-DESIGNATOR .( INTEGER-EXPRESSION ).

STRING-VARIABLE ::= STRING-VARIABLE-IDENTIFIER
OR STRING-FIELD-IDENTIFIER ( REFERENCE-EXPRESSION )
OR STRING-ARRAY-DESIGNATOR .( INTEGER-EXPRESSION ).

```

PL n° 25 GRAMMAIRE D'UN NOUVEL ALGOL  
 REGLES (SUITE)

```

REFERENCE-VARIABLE ::= REFERENCE-VARIABLE-IDENTIFIER
                    OR REFERENCE-FIELD-IDENTIFIER ( REFERENCE-EXPRESSION )
                    OR REFERENCE-ARRAY-DESIGNATOR .( INTEGER-EXPRESSION ).
INTEGER-ARRAY-DESIGNATOR ::= INTEGER-ARRAY-IDENTIFIER
                           OR INTEGER-ARRAY-DESIGNATOR .( INTEGER-EXPRESSION ).
SHORT-REAL-ARRAY-DESIGNATOR ::= SHORT-REAL-ARRAY-IDENTIFIER
                              OR SHORT-REAL-ARRAY-DESIGNATOR .( INTEGER-EXPRESSION ).
LONG-REAL-ARRAY-DESIGNATOR ::= LONG-REAL-ARRAY-IDENTIFIER
                             OR LONG-REAL-ARRAY-DESIGNATOR .( INTEGER-EXPRESSION ).
SHORT-COMPLEX-ARRAY-DESIGNATOR ::= SHORT-COMPLEX-ARRAY-IDENTIFIER
                                  OR SHORT-COMPLEX-ARRAY-DESIGNATOR .( INTEGER-EXPRESSION ).
LONG-COMPLEX-ARRAY-DESIGNATOR ::= LONG-COMPLEX-ARRAY-IDENTIFIER
                                 OR LONG-COMPLEX-ARRAY-DESIGNATOR .( INTEGER-EXPRESSION ).
LOGICAL-ARRAY-DESIGNATOR ::= LOGICAL-ARRAY-IDENTIFIER
                            OR LOGICAL-ARRAY-DESIGNATOR .( INTEGER-EXPRESSION ).
BIT-ARRAY-DESIGNATOR ::= BIT-ARRAY-IDENTIFIER
                      OR BIT-ARRAY-DESIGNATOR .( INTEGER-EXPRESSION ).
STRING-ARRAY-DESIGNATOR ::= STRING-ARRAY-IDENTIFIER
                          OR STRING-ARRAY-DESIGNATOR .( INTEGER-EXPRESSION ).
REFERENCE-ARRAY-DESIGNATOR ::= REFERENCE-ARRAY-IDENTIFIER
                             OR REFERENCE-ARRAY-DESIGNATOR .( INTEGER-EXPRESSION ).
INTEGER-NUMBER ::= BIT
                OR DIGIT-GREATER-THEN-1
                OR INTEGER-NUMBER BIT
                OR INTEGER-NUMBER DIGIT-GREATER-THEN-1
BIT ::= 0
     OR 1
DIGIT-GREATER-THEN-1 ::= 2
                      OR 3
                      OR 4
                      OR 5
                      OR 6
                      OR 7
                      OR 8
                      OR 9
SHORT-REAL-NUMBER ::= UNSCALED-REAL
                    OR UNSCALED-REAL DECIMAL-EXPONENTIATION-OPERATOR INTEGER-NUMBER
                    OR INTEGER-NUMBER DECIMAL-EXPONENTIATION-OPERATOR INTEGER-NUMBER
DECIMAL-EXPONENTIATION-OPERATOR ::= '10'
                                   OR '10' +
                                   OR '10' -
UNSCALED-REAL ::= . INTEGER-NUMBER
               OR INTEGER-NUMBER . INTEGER-NUMBER
LONG-REAL-NUMBER ::= 'LONG' SHORT-REAL-NUMBER
                  OR 'LONG' INTEGER-NUMBER
SHORT-COMPLEX-NUMBER ::= INTEGER-OR-SHORT-REAL-NUMBER IMAGINARY-OPERATOR
                       OR INTEGER-OR-SHORT-REAL-NUMBER
IMAGINARY-OPERATOR ::= 'I'
                    OR 'I' +
                    OR 'I' -
INTEGER-OR-SHORT-REAL-NUMBER ::= INTEGER-NUMBER
                              OR SHORT-REAL-NUMBER
LONG-COMPLEX-NUMBER ::= 'LONG' SHORT-COMPLEX-NUMBER
INTEGER-VARIABLE-IDENTIFIER ::= IDENTIFIER
SHORT-REAL-VARIABLE-IDENTIFIER ::= IDENTIFIER
LONG-REAL-VARIABLE-IDENTIFIER ::= IDENTIFIER
SHORT-COMPLEX-VARIABLE-IDENTIFIER ::= IDENTIFIER
LONG-COMPLEX-VARIABLE-IDENTIFIER ::= IDENTIFIER
LOGICAL-VARIABLE-IDENTIFIER ::= IDENTIFIER
BIT-VARIABLE-IDENTIFIER ::= IDENTIFIER
STRING-VARIABLE-IDENTIFIER ::= IDENTIFIER
REFERENCE-VARIABLE-IDENTIFIER ::= IDENTIFIER
INTEGER-ARRAY-IDENTIFIER ::= IDENTIFIER
SHORT-REAL-ARRAY-IDENTIFIER ::= IDENTIFIER
LONG-REAL-ARRAY-IDENTIFIER ::= IDENTIFIER
SHORT-COMPLEX-ARRAY-IDENTIFIER ::= IDENTIFIER
LONG-COMPLEX-ARRAY-IDENTIFIER ::= IDENTIFIER
LOGICAL-ARRAY-IDENTIFIER ::= IDENTIFIER
BIT-ARRAY-IDENTIFIER ::= IDENTIFIER
STRING-ARRAY-IDENTIFIER ::= IDENTIFIER
REFERENCE-ARRAY-IDENTIFIER ::= IDENTIFIER
PROCEDURE-IDENTIFIER ::= IDENTIFIER
INTEGER-FUNCTION-IDENTIFIER ::= IDENTIFIER
SHORT-REAL-FUNCTION-IDENTIFIER ::= IDENTIFIER
LONG-REAL-FUNCTION-IDENTIFIER ::= IDENTIFIER
SHORT-COMPLEX-FUNCTION-IDENTIFIER ::= IDENTIFIER
LONG-COMPLEX-FUNCTION-IDENTIFIER ::= IDENTIFIER
LOGICAL-FUNCTION-IDENTIFIER ::= IDENTIFIER
BIT-FUNCTION-IDENTIFIER ::= IDENTIFIER
STRING-FUNCTION-IDENTIFIER ::= IDENTIFIER
REFERENCE-FUNCTION-IDENTIFIER ::= IDENTIFIER
RECORD-CLASS-IDENTIFIER ::= IDENTIFIER
INTEGER-FIELD-IDENTIFIER ::= IDENTIFIER
SHORT-REAL-FIELD-IDENTIFIER ::= IDENTIFIER
LONG-REAL-FIELD-IDENTIFIER ::= IDENTIFIER
SHORT-COMPLEX-FIELD-IDENTIFIER ::= IDENTIFIER
LONG-COMPLEX-FIELD-IDENTIFIER ::= IDENTIFIER
LOGICAL-FIELD-IDENTIFIER ::= IDENTIFIER
BIT-FIELD-IDENTIFIER ::= IDENTIFIER
STRING-FIELD-IDENTIFIER ::= IDENTIFIER
REFERENCE-FIELD-IDENTIFIER ::= IDENTIFIER
LABEL-IDENTIFIER ::= IDENTIFIER
CONTROL-IDENTIFIER ::= IDENTIFIER
IDENTIFIER ::= LETTER
            OR IDENTIFIER LETTER
            OR IDENTIFIER BIT
            OR IDENTIFIER DIGIT-GREATER-THEN-1
LETTER ::= A
        OR B
        OR C
  
```





RELATION → ENTRE LES OPERANDES (SUITE)

SIMPLE-NUMERICAL-EXPRESSION	243	.....1.....1.....
SIMPLE-NUMERICAL-NON-COMPLEX-EXPRESSION	244	.....1.....1.....
INTEGER-TERM	245	.....1.....1.....
SHORT-REAL-TERM	246	.....1.....1.....
NUMERICAL-NON-COMPLEX-TERM	247	.....1.....1.....
LONG-REAL-TERM	248	.....1.....1.....
SHORT-COMPLEX-TERM	249	.....1.....1.....
NUMERICAL-NON-SHORT-COMPLEX-TERM	250	.....1.....1.....
LONG-COMPLEX-TERM	251	.....1.....1.....
INTEGER-OR-LONG-NUMERICAL-TERM	252	.....1.....1.....
REAL-TERM	253	.....1.....1.....
COMPLEX-TERM	254	.....1.....1.....
INTEGER-FACTOR	255	.....1.....1.....
SHORT-REAL-FACTOR	256	.....1.....1.....
INTEGER-OR-LONG-REAL-FACTOR	257	.....1.....1.....
LONG-REAL-FACTOR	258	.....1.....1.....
REAL-FACTOR	259	.....1.....1.....
SHORT-COMPLEX-FACTOR	260	.....1.....1.....
NUMERICAL-FACTOR	261	.....1.....1.....
LONG-COMPLEX-FACTOR	262	.....1.....1.....
COMPLEX-FACTOR	263	.....1.....1.....
INTEGER-SECONDARY	264	.....1.....1.....
SHORT-REAL-SECONDARY	265	.....1.....1.....
LONG-REAL-SECONDARY	266	.....1.....1.....
SHORT-COMPLEX-SECONDARY	267	.....1.....1.....
LONG-COMPLEX-SECONDARY	268	.....1.....1.....
INTEGER-PRIMARY	269	.....1.....1.....
SHORT-REAL-PRIMARY	270	.....1.....1.....
SHORT-REAL-NUMBER	271	.....1.....1.....
SHORT-COMPLEX-PRIMARY	272	.....1.....1.....
LONG-REAL-PRIMARY	273	.....1.....1.....
LONG-REAL-NUMBER	274	.....1.....1.....
LONG-COMPLEX-PRIMARY	275	.....1.....1.....
SHORT-COMPLEX-NUMBER	276	.....1.....1.....
LONG-COMPLEX-NUMBER	277	.....1.....1.....
SHORT-REAL-VARIABLE	278	.....1.....1.....
LONG-REAL-VARIABLE	279	.....1.....1.....
SHORT-COMPLEX-VARIABLE	280	.....1.....1.....
LONG-COMPLEX-VARIABLE	281	.....1.....1.....
LOGICAL-TERM	282	.....1.....1.....
RELATION	283	.....1.....1.....
LOGICAL-FACTOR	284	.....1.....1.....
LOGICAL-SECONDARY	285	.....1.....1.....
LOGICAL-PRIMARY	286	.....1.....1.....
BIT-TERM	287	.....1.....1.....
BIT-FACTOR	288	.....1.....1.....
BIT-SECONDARY	289	.....1.....1.....
BIT-PRIMARY	290	.....1.....1.....
BIT-SEQUENCE	291	.....1.....1.....
STRING-PRIMARY	292	.....1.....1.....
EXPRESSION-LIST	293	.....1.....1.....1.1.1.1.1
INTEGER-VARIABLE-IDENTIFIER	294	.....1.....1.....
INTEGER-FIELD-IDENTIFIER	295	.....1.....1.....
SHORT-REAL-VARIABLE-IDENTIFIER	296	.....1.....1.....
SHORT-REAL-FIELD-IDENTIFIER	297	.....1.....1.....
LONG-REAL-VARIABLE-IDENTIFIER	298	.....1.....1.....
LONG-REAL-FIELD-IDENTIFIER	299	.....1.....1.....
SHORT-COMPLEX-VARIABLE-IDENTIFIER	300	.....1.....1.....
SHORT-COMPLEX-FIELD-IDENTIFIER	301	.....1.....1.....
LONG-COMPLEX-VARIABLE-IDENTIFIER	302	.....1.....1.....
LONG-COMPLEX-FIELD-IDENTIFIER	303	.....1.....1.....
LOGICAL-VARIABLE-IDENTIFIER	304	.....1.....1.....
LOGICAL-FIELD-IDENTIFIER	305	.....1.....1.....
BIT-VARIABLE-IDENTIFIER	306	.....1.....1.....
BIT-FIELD-IDENTIFIER	307	.....1.....1.....
STRING-VARIABLE-IDENTIFIER	308	.....1.....1.....
STRING-FIELD-IDENTIFIER	309	.....1.....1.....
REFERENCE-FIELD-IDENTIFIER	310	.....1.....1.....
INTEGER-ARRAY-IDENTIFIER	311	.....1.....1.....
SHORT-REAL-ARRAY-IDENTIFIER	312	.....1.....1.....
LONG-REAL-ARRAY-IDENTIFIER	313	.....1.....1.....
SHORT-COMPLEX-ARRAY-IDENTIFIER	314	.....1.....1.....
LONG-COMPLEX-ARRAY-IDENTIFIER	315	.....1.....1.....
LOGICAL-ARRAY-IDENTIFIER	316	.....1.....1.....
BIT-ARRAY-IDENTIFIER	317	.....1.....1.....
STRING-ARRAY-IDENTIFIER	318	.....1.....1.....
REFERENCE-ARRAY-IDENTIFIER	319	.....1.....1.....
UNSCALED-REAL	320	.....1.....1.....
INTEGER-OR-SHORT-REAL-NUMBER	321	.....1.....1.....

RELATION  $\Rightarrow^*$  ENTRE LES OPERANDES (SUITE)

REFERENCE-EXPRESSION	164	.....1.....
REFERENCE-PROCEDURE-BODY-INTERIOR	165	.....1.....
FORMAL-PARAMETER-SEGMENT	166	.....1.....
TYPE	167	.....1.....11111111.....
SIMPLE-STATEMENT	168	.....1.....
IF-STATEMENT	169	.....1.....
CASE-STATEMENT	170	.....1.....
ITERATIVE-STATEMENT	171	.....1.....
NUMERICAL-ASSIGNMENT-STATEMENT	172	.....1.....
LOGICAL-ASSIGNMENT-STATEMENT	173	.....1.....
BIT-ASSIGNMENT-STATEMENT	174	.....1.....
STRING-ASSIGNMENT-STATEMENT	175	.....1.....
REFERENCE-ASSIGNMENT-STATEMENT	176	.....1.....
PROCEDURE-STATEMENT	177	.....1.....
GO-TO-STATEMENT	178	.....1.....
COMPLEX-ASSIGNMENT-STATEMENT	179	.....1.....
NUMERICAL-NON-COMPLEX-ASSIGNMENT-STATEMENT	180	.....1.....
INTEGER-ASSIGNMENT-STATEMENT	181	.....1.....
REAL-ASSIGNMENT-STATEMENT	182	.....1.....
INTEGER-VARIABLE	183	.....1.....
REAL-VARIABLE	184	.....1.....
NUMERICAL-NON-COMPLEX-EXPRESSION	185	.....1.....1.....1.....1.....
COMPLEX-VARIABLE	186	.....1.....
NUMERICAL-EXPRESSION	187	.....1.....1.....1.....1.....1.....
LOGICAL-VARIABLE	188	.....1.....
BIT-VARIABLE	189	.....1.....
STRING-VARIABLE	190	.....1.....
REFERENCE-VARIABLE	191	.....1.....
PROCEDURE-IDENTIFIER	192	.....1.....
ACTUAL-PARAMETER-LIST	193	.....1.....1.....1.....1.....1.....1.....
ACTUAL-PARAMETER	194	.....1.....1.....1.....1.....1.....1.....
EXPRESSION	195	.....1.....1.....1.....1.....1.....1.....
INTEGER-ARRAY-DESIGNATOR	196	.....1.....
SHORT-REAL-ARRAY-DESIGNATOR	197	.....1.....
LONG-REAL-ARRAY-DESIGNATOR	198	.....1.....
SHORT-COMPLEX-ARRAY-DESIGNATOR	199	.....1.....
LONG-COMPLEX-ARRAY-DESIGNATOR	200	.....1.....
LOGICAL-ARRAY-DESIGNATOR	201	.....1.....
BIT-ARRAY-DESIGNATOR	202	.....1.....
STRING-ARRAY-DESIGNATOR	203	.....1.....
REFERENCE-ARRAY-DESIGNATOR	204	.....1.....
INTEGER-FUNCTION-IDENTIFIER	205	.....1.....
SHORT-REAL-FUNCTION-IDENTIFIER	206	.....1.....
LONG-REAL-FUNCTION-IDENTIFIER	207	.....1.....
SHORT-COMPLEX-FUNCTION-IDENTIFIER	208	.....1.....
LONG-COMPLEX-FUNCTION-IDENTIFIER	209	.....1.....
LOGICAL-FUNCTION-IDENTIFIER	210	.....1.....
BIT-FUNCTION-IDENTIFIER	211	.....1.....
STRING-FUNCTION-IDENTIFIER	212	.....1.....
REFERENCE-FUNCTION-IDENTIFIER	213	.....1.....
LABEL-IDENTIFIER	214	.....1.....
DISCRIMINATOR	215	.....1.....
REFERENCE-VARIABLE-IDENTIFIER	216	.....1.....
CASE-CLAUSE	217	.....1.....
STATEMENT-LIST	218	.....1.....1.....1.....1.....1.....1.....
CONTROL-IDENTIFIER	219	.....1.....
INTEGER-OR-LONG-NUMERICAL-EXPRESSION	220	.....1.....1.....1.....1.....1.....1.....
SIMPLE-INTEGER-EXPRESSION	221	.....1.....
INTEGER-EXPRESSION-LIST	222	.....1.....1.....1.....1.....1.....1.....
SIMPLE-SHORT-REAL-EXPRESSION	223	.....1.....
SHORT-REAL-EXPRESSION-LIST	224	.....1.....1.....1.....1.....1.....1.....
SIMPLE-INTEGER-OR-LONG-REAL-EXPRESSION	225	.....1.....1.....1.....1.....1.....1.....
SIMPLE-LONG-REAL-EXPRESSION	226	.....1.....
LONG-REAL-EXPRESSION-LIST	227	.....1.....1.....1.....1.....1.....1.....
SIMPLE-SHORT-COMPLEX-EXPRESSION	228	.....1.....
SHORT-COMPLEX-EXPRESSION-LIST	229	.....1.....1.....1.....1.....1.....1.....
SIMPLE-NUMERICAL-NON-SHORT-COMPLEX-EXPRESSION	230	.....1.....1.....1.....1.....1.....1.....
SIMPLE-LONG-COMPLEX-EXPRESSION	231	.....1.....
LONG-COMPLEX-EXPRESSION-LIST	232	.....1.....1.....1.....1.....1.....1.....
SIMPLE-LOGICAL-EXPRESSION	233	.....1.....
LOGICAL-EXPRESSION-LIST	234	.....1.....1.....1.....1.....1.....1.....
SIMPLE-BIT-EXPRESSION	235	.....1.....
BIT-EXPRESSION-LIST	236	.....1.....1.....1.....1.....1.....1.....
SIMPLE-STRING-EXPRESSION	237	.....1.....
STRING-EXPRESSION-LIST	238	.....1.....1.....1.....1.....1.....1.....
SIMPLE-REFERENCE-EXPRESSION	239	.....1.....
REFERENCE-EXPRESSION-LIST	240	.....1.....1.....1.....1.....1.....1.....
NUMERICAL-NON-SHORT-COMPLEX-EXPRESSION-LIST	241	.....1.....1.....1.....1.....1.....1.....
INTEGER-OR-LONG-REAL-EXPRESSION-LIST	242	.....1.....1.....1.....1.....1.....1.....













RELATION  $\Rightarrow^*$  ENTRE LES OPERANDES (S)

	23333333333333333333333333333333
	9C0000000011111111111122
	9C123456789012345678901
PROGRAM	099 .....
BLOCK	090 .....
BLOCK-INTERIOR	091 .....
STATEMENT	092 .....
BLOCK-INTERIOR-OR-DECLARATION-PART	093 .....
INCOMPLETE-BLOCK-INTERIOR	094 .....
IDENTIFIER	095 .....
DECLARATION-PART	096 .....
NON-PROCEDURE-DECLARATION	097 .....
PROPER-INCOMPLETE-DECLARATION-PART	098 .....
INTEGER-INCOMPLETE-DECLARATION-PART	099 .....
INTEGER-PROCEDURE-BODY	100 .....
SHORT-REAL-INCOMPLETE-DECLARATION-PART	101 .....
SHORT-REAL-PROCEDURE-BODY	102 .....1.
LONG-REAL-INCOMPLETE-DECLARATION-PART	103 .....
LONG-REAL-PROCEDURE-BODY	104 .....
SHORT-COMPLEX-INCOMPLETE-DECLARATION-PART	105 .....
SHORT-COMPLEX-PROCEDURE-BODY	106 .....1.
LONG-COMPLEX-INCOMPLETE-DECLARATION-PART	107 .....
LONG-COMPLEX-PROCEDURE-BODY	108 .....1.
LOGICAL-INCOMPLETE-DECLARATION-PART	109 .....
LOGICAL-PROCEDURE-BODY	110 .....1.
BIT-INCOMPLETE-DECLARATION-PART	111 .....
BIT-PROCEDURE-BODY	112 .....1.
STRING-INCOMPLETE-DECLARATION-PART	113 .....
STRING-PROCEDURE-BODY	114 .....1.
REFERENCE-INCOMPLETE-DECLARATION-PART	115 .....
REFERENCE-PROCEDURE-BODY	116 .....
SIMPLE-VARIABLE-DECLARATION	117 .....
ARRAY-DECLARATION	118 .....
RECORD-CLASS-DECLARATION	119 .....
SIMPLE-TYPE-FOLLOWED-BY-IDENTIFIER	120 .....
INTEGER-NUMBER	121 .....
RECORD-CLASS-IDENTIFIER	122 .....
INCOMPLETE-ARRAY-DECLARATION	123 .....
BOUND-PAIR	124 .....
SIMPLE-TYPE	125 .....
INTEGER-EXPRESSION	126 .....
FIELD-LIST	127 .....
PROPER-PROCEDURE-HEADING	128 .....
INTEGER-PROCEDURE-HEADING	129 .....
SHORT-REAL-PROCEDURE-HEADING	130 .....
LONG-REAL-PROCEDURE-HEADING	131 .....
SHORT-COMPLEX-PROCEDURE-HEADING	132 .....
LONG-COMPLEX-PROCEDURE-HEADING	133 .....
LOGICAL-PROCEDURE-HEADING	134 .....
BIT-PROCEDURE-HEADING	135 .....
STRING-PROCEDURE-HEADING	136 .....
REFERENCE-PROCEDURE-HEADING	137 .....
PROCEDURE-HEADING	138 .....
SIMPLE-INTEGER-TYPE	139 .....
SIMPLE-SHORT-REAL-TYPE	140 .....
SIMPLE-LONG-REAL-TYPE	141 .....
SIMPLE-SHORT-COMPLEX-TYPE	142 .....
SIMPLE-LONG-COMPLEX-TYPE	143 .....1.
SIMPLE-LOGICAL-TYPE	144 .....
SIMPLE-BIT-TYPE	145 .....
SIMPLE-STRING-TYPE	146 .....
SIMPLE-REFERENCE-TYPE	147 .....
FORMAL-PARAMETER-LIST	148 .....
INTEGER-PROCEDURE-BODY-INTERIOR	149 .....
SHORT-REAL-EXPRESSION	150 .....1.
SHORT-REAL-PROCEDURE-BODY-INTERIOR	151 .....1.
LONG-REAL-EXPRESSION	152 .....
LONG-REAL-PROCEDURE-BODY-INTERIOR	153 .....
SHORT-COMPLEX-EXPRESSION	154 .....1.
SHORT-COMPLEX-PROCEDURE-BODY-INTERIOR	155 .....1.
LONG-COMPLEX-EXPRESSION	156 .....1.
LONG-COMPLEX-PROCEDURE-BODY-INTERIOR	157 .....1.
LOGICAL-EXPRESSION	158 .....1.
LOGICAL-PROCEDURE-BODY-INTERIOR	159 .....1.
BIT-EXPRESSION	160 .....1.
BIT-PROCEDURE-BODY-INTERIOR	161 .....1.
STRING-EXPRESSION	162 .....1.
STRING-PROCEDURE-BODY-INTERIOR	163 .....1.

RELATION  $\Rightarrow^*$  ENTRE LES OPERANDES (SUITE)

REFERENCE-EXPRESSION	164	.....
REFERENCE-PROCEDURE-BODY-INTERIOR	165	.....
FORMAL-PARAMETER-SEGMENT	166	.....
TYPE	167	.....
SIMPLE-STATEMENT	168	.....
IF-STATEMENT	169	.....
CASE-STATEMENT	170	.....
ITERATIVE-STATEMENT	171	.....
NUMERICAL-ASSIGNMENT-STATEMENT	172	.....
LOGICAL-ASSIGNMENT-STATEMENT	173	.....
BIT-ASSIGNMENT-STATEMENT	174	.....
STRING-ASSIGNMENT-STATEMENT	175	.....
REFERENCE-ASSIGNMENT-STATEMENT	176	.....
PROCEDURE-STATEMENT	177	.....
GO-TO-STATEMENT	178	.....
COMPLEX-ASSIGNMENT-STATEMENT	179	.....
NUMERICAL-NON-COMPLEX-ASSIGNMENT-STATEMENT	180	.....
INTEGER-ASSIGNMENT-STATEMENT	181	.....
REAL-ASSIGNMENT-STATEMENT	182	.....
INTEGER-VARIABLE	183	.....
REAL-VARIABLE	184	.....
NUMERICAL-NON-COMPLEX-EXPRESSION	185	.....1
COMPLEX-VARIABLE	186	..1.1.....
NUMERICAL-EXPRESSION	187	..1.1.....1
LOGICAL-VARIABLE	188	.....1.....
BIT-VARIABLE	189	.....1.....
STRING-VARIABLE	190	.....1.....
REFERENCE-VARIABLE	191	.....
PROCEDURE-IDENTIFIER	192	.....
ACTUAL-PARAMETER-LIST	193	..1.1.1.1.1.1111111111
ACTUAL-PARAMETER-EXPRESSION	194	..1.1.1.1.1.1111111111
INTEGER-ARRAY-DESIGNATOR	195	.....1.....
SHORT-REAL-ARRAY-DESIGNATOR	196	.....1.....
LONG-REAL-ARRAY-DESIGNATOR	197	.....1.....
SHORT-COMPLEX-ARRAY-DESIGNATOR	198	.....1.....
LONG-COMPLEX-ARRAY-DESIGNATOR	199	.....1.....
LOGICAL-ARRAY-DESIGNATOR	200	.....1.....
BIT-ARRAY-DESIGNATOR	201	.....1.....
STRING-ARRAY-DESIGNATOR	202	.....1.....
REFERENCE-ARRAY-DESIGNATOR	203	.....1.....
INTEGER-FUNCTION-IDENTIFIER	204	.....1.....
SHORT-REAL-FUNCTION-IDENTIFIER	205	.....1.....
LONG-REAL-FUNCTION-IDENTIFIER	206	.....1.....
SHORT-COMPLEX-FUNCTION-IDENTIFIER	207	.....1.....
LONG-COMPLEX-FUNCTION-IDENTIFIER	208	.....1.....
LOGICAL-FUNCTION-IDENTIFIER	209	.....1.....
BIT-FUNCTION-IDENTIFIER	210	.....1.....
STRING-FUNCTION-IDENTIFIER	211	.....1.....
REFERENCE-FUNCTION-IDENTIFIER	212	.....1.....
LABEL-IDENTIFIER	213	.....1.....
DISCRIMINATOR	214	.....1.....
REFERENCE-VARIABLE-IDENTIFIER	215	.....1.....
CASE-CLAUSE	216	.....1.....
STATEMENT-LIST	217	.....1.....
CONTROL-IDENTIFIER	218	.....1.....
INTEGER-OR-LONG-NUMERICAL-EXPRESSION	219	.....1.....
SIMPLE-INTEGER-EXPRESSION	220	.....1.....
INTEGER-EXPRESSION-LIST	221	.....1.....
SIMPLE-SHORT-REAL-EXPRESSION	222	.....1.....
SHORT-REAL-EXPRESSION-LIST	223	.....1.....
SIMPLE-INTEGER-OR-LONG-REAL-EXPRESSION	224	.....1.....
SIMPLE-LONG-REAL-EXPRESSION	225	.....1.....
LONG-REAL-EXPRESSION-LIST	226	.....1.....
SIMPLE-SHORT-COMPLEX-EXPRESSION	227	.....1.....
SHORT-COMPLEX-EXPRESSION-LIST	228	.....1.....
SIMPLE-NUMERICAL-NON-SHORT-COMPLEX-EXPRESSION	229	.....1.....
SIMPLE-LONG-COMPLEX-EXPRESSION	230	.....1.....
LONG-COMPLEX-EXPRESSION-LIST	231	.....1.....
SIMPLE-LOGICAL-EXPRESSION	232	.....1.....
LOGICAL-EXPRESSION-LIST	233	.....1.....
SIMPLE-BIT-EXPRESSION	234	.....1.....
BIT-EXPRESSION-LIST	235	.....1.....
SIMPLE-STRING-EXPRESSION	236	.....1.....
STRING-EXPRESSION-LIST	237	.....1.....
SIMPLE-REFERENCE-EXPRESSION	238	.....1.....
REFERENCE-EXPRESSION-LIST	239	.....1.....
NUMERICAL-NON-SHORT-COMPLEX-EXPRESSION-LIST	240	.....1.....
INTEGER-OR-LONG-REAL-EXPRESSION-LIST	241	.....1.....
	242	.....

RELATION  $\Rightarrow^*$  ENTRE LES OPERANDES (SUITE)

SIMPLE-NUMERICAL-EXPRESSION	244	.1.1.....1.
SIMPLE-NUMERICAL-NON-COMPLEX-EXPRESSION	244	.....1.
INTEGER-TERM	245	.....1.
SHORT-REAL-TERM	246	.....1.
NUMERICAL-NON-COMPLEX-TERM	247	.....1.
LONG-REAL-TERM	248	.....1.
SHORT-COMPLEX-TERM	249	.1.....1.
NUMERICAL-NON-SHORT-COMPLEX-TERM	250	.....1.
LONG-COMPLEX-TERM	251	.....1.
INTEGER-OR-LONG-NUMERICAL-TERM	252	.....1.
REAL-TERM	253	.....1.
COMPLEX-TERM	254	.1.1.....1.
INTEGER-FACTOR	255	.....1.
SHORT-REAL-FACTOR	256	.....1.
INTEGER-OR-LONG-REAL-FACTOR	257	.....1.
LONG-REAL-FACTOR	258	.....1.
REAL-FACTOR	259	.....1.
SHORT-COMPLEX-FACTOR	260	.1.....1.
NUMERICAL-FACTOR	261	.1.1.....1.
LONG-COMPLEX-FACTOR	262	.....1.
COMPLEX-FACTOR	263	.1.1.....1.
INTEGER-SECONDARY	264	.....1.
SHORT-REAL-SECONDARY	265	.....1.
LONG-REAL-SECONDARY	266	.....1.
SHORT-COMPLEX-SECONDARY	267	.1.....1.
LONG-COMPLEX-SECONDARY	268	.....1.
INTEGER-PRIMARY	269	.....1.
SHORT-REAL-PRIMARY	270	.....1.
SHORT-REAL-NUMBER	271	.....1.
SHORT-COMPLEX-PRIMARY	272	.1.....1.
LONG-REAL-PRIMARY	273	.....1.
LONG-REAL-NUMBER	274	.....1.
LONG-COMPLEX-PRIMARY	275	.1.....1.
SHORT-COMPLEX-NUMBER	276	.....1.
LONG-COMPLEX-NUMBER	277	.....1.
SHORT-REAL-VARIABLE	278	.....1.
LONG-REAL-VARIABLE	279	.....1.
SHORT-COMPLEX-VARIABLE	280	.1.....1.
LONG-COMPLEX-VARIABLE	281	.....1.
LOGICAL-TERM	282	.....1.
RELATION	283	.....1.
LOGICAL-FACTOR	284	.....1.
LOGICAL-SECONDARY	285	.....1.
LOGICAL-PRIMARY	286	.....1.
BIT-TERM	287	.....1.
BIT-FACTOR	288	.....1.
BIT-SECONDARY	289	.....1.
BIT-PRIMARY	290	.....1.
BIT-SEQUENCE	291	.....1.
STRING-PRIMARY	292	.....1.
EXPRESSION-LIST	293	.1.1.1.1.1.....1.
INTEGER-VARIABLE-IDENTIFIER	294	.....1.
INTEGER-FIELD-IDENTIFIER	295	.....1.
SHORT-REAL-VARIABLE-IDENTIFIER	296	.....1.
SHORT-REAL-FIELD-IDENTIFIER	297	.....1.
LONG-REAL-VARIABLE-IDENTIFIER	298	.....1.
LONG-REAL-FIELD-IDENTIFIER	299	.1.....1.
SHORT-COMPLEX-VARIABLE-IDENTIFIER	300	.1.....1.
SHORT-COMPLEX-FIELD-IDENTIFIER	301	.....1.
LONG-COMPLEX-VARIABLE-IDENTIFIER	302	.....1.
LONG-COMPLEX-FIELD-IDENTIFIER	303	.....1.
LOGICAL-VARIABLE-IDENTIFIER	304	.....1.
LOGICAL-FIELD-IDENTIFIER	305	.....1.
BIT-VARIABLE-IDENTIFIER	306	.....1.
BIT-FIELD-IDENTIFIER	307	.....1.
STRING-VARIABLE-IDENTIFIER	308	.....1.
STRING-FIELD-IDENTIFIER	309	.....1.
REFERENCE-FIELD-IDENTIFIER	310	.....1.
INTEGER-ARRAY-IDENTIFIER	311	.....1.
SHORT-REAL-ARRAY-IDENTIFIER	312	.....1.
LONG-REAL-ARRAY-IDENTIFIER	313	.....1.
SHORT-COMPLEX-ARRAY-IDENTIFIER	314	.....1.
LONG-COMPLEX-ARRAY-IDENTIFIER	315	.....1.
LOGICAL-ARRAY-IDENTIFIER	316	.....1.
BIT-ARRAY-IDENTIFIER	317	.....1.
STRING-ARRAY-IDENTIFIER	318	.....1.
REFERENCE-ARRAY-IDENTIFIER	319	.....1.
UNSCALED-REAL	320	.....1.
INTEGER-OR-SHORT-REAL-NUMBER	321	.....11

#### 5.4 Précédence terminale et compilateur de conversation Fortran [\*]

Il s'agissait d'écrire un compilateur de conversation Fortran c'est-à-dire un compilateur qui au fur et à mesure que l'on tape son programme sur la machine à écrire d'une console, en signale les erreurs et permet de les corriger. Afin d'éviter la réécriture complète d'un compilateur Fortran qui existait déjà sur la machine envisagée (UNIVAC 1108) l'option suivante a été prise :

- Le programme à compiler est d'abord traité par un analyseur dont le rôle consiste uniquement à détecter les erreurs. En cas d'erreur, après correction de la ligne erronée, l'analyseur reprend le programme à son début.

- Quand la totalité du programme a été jugée correcte par l'analyseur il est alors transmis au compilateur Fortran non conversationnel.

L'analyseur lui-même est composé de deux parties :

- Une partie édition dont le rôle est de coder la chaîne à analyser et de détecter certaines erreurs particulières telles que les erreurs de déclaration.

- Une partie analyse proprement dite qui consiste en l'analyseur de précédence terminale décrit au paragraphe 4.6.

Voici successivement la grammaire Fortran utilisée, ses relations de précédence terminale et la relation  $\Rightarrow^*$  entre ses opérandes (c'est-à-dire ses symboles non terminaux).

---

[\*] Les réalisations décrites ici ont été principalement l'oeuvre de H. Beauchataud J.Fama et J.P. Tamiatto du Centre de Calcul Scientifique de l'Armement

## REGLES

```

PROGRAMME-DELIMITE ::= ::= PROGRAMME ::=
PROGRAMME ::= LISTE-D'ORDRES RC END
                OU SOUS-PROGRAMME RC END
SOUS-PROGRAMME ::= ::= SOUS-PROGRAMME-INCOMPLET RC RETURN
                OU SOUS-PROGRAMME RC ORDRE
                OU SOUS-PROGRAMME RC RETURN
SOUS-PROGRAMME-INCOMPLET ::= ::= ORDRE-DE-SOUS-PROGRAMME
                OU SOUS-PROGRAMME-INCOMPLET RC ORDRE
ORDRE-DE-SOUS-PROGRAMME ::= ::= ORDRE-DE-SOUS-PROGRAMME-SANS-ETIQUETTE
                OU NOMBRE-DE-MOINS-DE-5-CHIFFRES ORDRE-DE-SOUS-PROGRAMME-SANS-ETIQUETTE
ORDRE-DE-SOUS-PROGRAMME-SANS-ETIQUETTE ::= ::= ORDRE-FUNCTION
                OU ORDRE-SUBROUTINE
LISTE-D'ORDRES ::= ::= ORDRE
                OU LISTE-D'ORDRES RC ORDRE
ORDRE ::= ::= ORDRE-EXECUTABLE
                OU ORDRE-NON-EXECUTABLE
                OU COMMENTAIRE
ORDRE-EXECUTABLE ::= ::= ORDRE-EXECUTABLE-SANS-ETIQUETTE
                OU NOMBRE-DE-MOINS-DE-5-CHIFFRES ORDRE-EXECUTABLE-SANS-ETIQUETTE
ORDRE-NON-EXECUTABLE ::= ::= ORDRE-DIMENSION
                OU ORDRE-EXTERNAL
                OU ETIQUETTE CONTINUE
                OU NOMBRE-DE-MOINS-DE-5-CHIFFRES ORDRE-FORMAT
ORDRE-EXECUTABLE-SANS-ETIQUETTE ::= ::= ORDRE-DE-CALCUL
                OU ORDRE-DE-TRANSFERT
                OU ORDRE-D'ENTREE-SORTIE
ORDRE-DE-CALCUL ::= ::= ORDRE-ARITHMETIQUE
                OU ORDRE-CALL
                OU ORDRE-DO
ORDRE-DE-TRANSFERT ::= ::= ORDRE-GOTO-SIMPLE
                OU ORDRE-GOTO-CALCULE
                OU ORDRE-IF
                OU STOP
ORDRE-D'ENTREE-SORTIE ::= ::= ORDRE-PRINT
                OU ORDRE-READ
ORDRE-FUNCTION ::= ::= FUNCTION IDENTIFICATEUR-SIMPLE ( LISTE-D'IDENTIFICATEURS-SIMPLES
LISTE-D'IDENTIFICATEURS-SIMPLES ::= ::= IDENTIFICATEUR-SIMPLE
                OU LISTE-D'IDENTIFICATEURS-SIMPLES , IDENTIFICATEUR-SIMPLE
ORDRE-SUBROUTINE ::= ::= SUBROUTINE IDENTIFICATEUR-SIMPLE
                OU SUBROUTINE IDENTIFICATEUR-SIMPLE ( LISTE-D'IDENTIFICATEURS-SIMPLES
IDENTIFICATEUR-SIMPLE ::= ::= IDENTIFICATEUR-SIMPLE-REEL
                OU IDENTIFICATEUR-SIMPLE-ENTIER
ORDRE-DIMENSION ::= ::= DIMENSION LISTE-POUR-DIMENSION
LISTE-POUR-DIMENSION ::= ::= VARIABLE-DIMENSION
                OU LISTE-POUR-DIMENSION , VARIABLE-DIMENSION
VARIABLE-DIMENSION ::= ::= IDENTIFICATEUR-SIMPLE ( DECLARATION-DE-DIMENSION )
DECLARATION-DE-DIMENSION ::= ::= CONSTANTE-ENTIERE-SANS-SIGNE
                OU DEUX-DIMENSIONS
                OU DEUX-DIMENSIONS , CONSTANTE-ENTIERE-SANS-SIGNE
DEUX-DIMENSIONS ::= ::= CONSTANTE-ENTIERE-SANS-SIGNE , CONSTANTE-ENTIERE-SANS-SIGNE
ORDRE-DO ::= ::= DO NOMBRE-DE-MOINS-DE-5-CHIFFRES LISTE-POUR-DO
LISTE-POUR-DO ::= ::= IDENTIFICATEUR-SIMPLE-ENTIER-AUXILIAIRE = SUITE-D'INDICES-POUR-DO
SUITE-D'INDICES-POUR-DO ::= ::= DEUX-INDICES-POUR-DO
                OU DEUX-INDICES-POUR-DO , INDICE-POUR-DO
DEUX-INDICES-POUR-DO ::= ::= INDICE-POUR-DO , INDICE-POUR-DO
INDICE-POUR-DO ::= ::= CONSTANTE-ENTIERE-SANS-SIGNE
                OU IDENTIFICATEUR-SIMPLE-ENTIER
ORDRE-FORMAT ::= ::= FORMAT ( LISTE-SPECIFICATION-FORMAT )
ORDRE-CALL ::= ::= CALL IDENTIFICATEUR-SIMPLE
                OU CALL IDENTIFICATEUR-SIMPLE ( LISTE-D'ARGUMENTS )
ORDRE-IF ::= ::= IF CORPS-DE-IF
CORPS-DE-IF ::= ::= CORPS-DE-IF-INCOMPLET , ETIQUETTE
CORPS-DE-IF-INCOMPLET ::= ::= CORPS-DE-IF-TRES-INCOMPLET , ETIQUETTE
CORPS-DE-IF-TRES-INCOMPLET ::= ::= EXPRESSION-ARITHMETIQUE-PARENTHESEE NOMBRE-DE-MOINS-DE-5-CHIFFRES
EXPRESSION-ARITHMETIQUE-PARENTHESEE ::= ::= ( EXPRESSION-ARITHMETIQUE )
ORDRE-GOTO-SIMPLE ::= ::= GOTO ETIQUETTE
ORDRE-GOTO-CALCULE ::= ::= GOTO LISTE-D'AIGUILLAGES , IDENTIFICATEUR-SIMPLE-ENTIER-AUXILIAIRE
LISTE-D'AIGUILLAGES ::= ::= ( LISTE-D'ETIQUETTES )
LISTE-D'ETIQUETTES ::= ::= ETIQUETTE
                OU LISTE-D'ETIQUETTES , ETIQUETTE
ORDRE-EXTERNAL ::= ::= EXTERNAL LISTE-DE-VARIABLES
ORDRE-PRINT ::= ::= PRINT LISTE-D'ENTREE-SORTIE
ORDRE-READ ::= ::= READ LISTE-D'ENTREE-SORTIE
LISTE-D'ENTREE-SORTIE ::= ::= ETIQUETTE , ARGUMENT-D'ENTREE-SORTIE
                OU LISTE-D'ENTREE-SORTIE , ARGUMENT-D'ENTREE-SORTIE
ARGUMENT-D'ENTREE-SORTIE ::= ::= VARIABLE
                OU GROUPE-D'ARGUMENTS-D'E/S-PARENTHESSES
GROUPE-D'ARGUMENTS-D'E/S-PARENTHESSES ::= ::= ( GROUPE-D'ARGUMENTS-D'E/S )
GROUPE-D'ARGUMENTS-D'E/S ::= ::= LISTE-DE-VARIABLES , LISTE-POUR-DO
                OU GROUPE-D'ARGUMENTS-D'E/S-PARENTHESSES , LISTE-POUR-DO
LISTE-DE-VARIABLES ::= ::= VARIABLE
                OU LISTE-DE-VARIABLES , VARIABLE
ORDRE-ARITHMETIQUE ::= ::= VARIABLE = EXPRESSION-ARITHMETIQUE
VARIABLE ::= ::= VARIABLE-REELLE
                OU VARIABLE-ENTIERE
FONCTION-REELLE ::= ::= IDENTIFICATEUR-SIMPLE-REEL-AUXILIAIRE ( LISTE-D'ARGUMENTS )
FONCTION-ENTIERE ::= ::= IDENTIFICATEUR-SIMPLE-ENTIER-AUXILIAIRE ( LISTE-D'ARGUMENTS )
LISTE-D'ARGUMENTS ::= ::= ARGUMENT
                OU LISTE-D'ARGUMENTS , ARGUMENT
ARGUMENT ::= ::= EXPRESSION-ARITHMETIQUE
EXPRESSION-ARITHMETIQUE ::= ::= EXPRESSION-ENTIERE
                OU EXPRESSION-REELLE
EXPRESSION-REELLE ::= ::= TERME-REEL
                OU SIGNE TERME-REEL
                OU EXPRESSION-REELLE SIGNE TERME-REEL
EXPRESSION-ENTIERE ::= ::= TERME-ENTIER
                OU SIGNE TERME-ENTIER
                OU EXPRESSION-ENTIERE SIGNE TERME-ENTIER
TERME-REEL ::= ::= FACTEUR-REEL
                OU TERME-REEL * FACTEUR-REEL
                OU TERME-REEL / FACTEUR-REEL

```





### 5.5 Précédence terminale et compilateur de conversation pour un sous-ensemble d'Algol

Nous décrivons ici un compilateur de conversation pour un sous-ensemble d'Algol. Ce compilateur est composé de trois parties :

- un éditeur dont le rôle est de reconnaître les symboles de base, de supprimer les déclarations et de constituer le dictionnaire des identificateurs utilisés ;
- un analyseur de précédence terminale (celui décrit au paragraphe 4.6) ;
- un générateur qui à chaque réduction d'une sous-chaîne par l'analyseur génère une partie du programme objet sous forme de macro-instructions.

Ce compilateur est principalement l'oeuvre de G. Tassart ; les macro-instructions ont été définies en langage d'assemblage MAP (IBM 7044) par A. Auroux. Nous donnons successivement la grammaire du sous-ensemble d'Algol, ses relations de précédence terminale, la relation  $\Rightarrow^*$  entre ses symboles non terminaux et un exemple commenté de compilation d'un programme.



PL n° 43 EXEMPLE D'ANALYSE ET DE COMPLATION D'UN PROGRAMME ALGOL

ANALYSE DU PROGRAMME

DEBUT  
RESER X1,1  
RESER X2,1

CONER1 =3  
AFFE X2

E1 ANATO X2  
AFFE X1

CONER1 =3  
FDIV2 X1

FADD3 X1

CONER1 =2  
FDIV4

'DEBUT' 'REFL' X1,X2 :: X2:=3 ::

E1: X1:=X2 :: X2:=(X1+3/X1)/2 ::

\*\*\*\*\*LA SEQUENCE  
=3 NOMBRE-ENTIER  
EST REMPLACEE PAR  
PRIMAIRE-ARITHMETIQUE  
\*\*\*\*\*LA SEQUENCE  
X2 VARIABLE-REELLE  
:=  
=3 00006 PRIMAIRE-ARITHMETIQUE  
EST REMPLACEE PAR  
INSTRUCTION-NON-ETIQUETEE

\*\*\*\*\*LA SEQUENCE  
X2 VARIABLE-REELLE  
EST REMPLACEE PAR  
PRIMAIRE-ARITHMETIQUE  
\*\*\*\*\*LA SEQUENCE  
X1 VARIABLE-REELLE  
:=  
X2 00004 PRIMAIRE-ARITHMETIQUE  
EST REMPLACEE PAR  
INSTRUCTION-NON-ETIQUETEE

\*\*\*\*\*LA SEQUENCE  
E1 ETIQUETTE  
:  
INSTRUCTION-NON-ETIQUETEE  
EST REMPLACEE PAR  
INSTRUCTION

\*\*\*\*\*LA SEQUENCE  
INSTRUCTION-NON-ETIQUETEE  
'POINT-VIRGULE'  
INSTRUCTION  
EST REMPLACEE PAR  
SUITE-D-INSTRUCTIONS

\*\*\*\*\*LA SEQUENCE  
X1 VARIABLE-REELLE  
EST REMPLACEE PAR  
PRIMAIRE-ARITHMETIQUE

\*\*\*\*\*LA SEQUENCE  
=3 NOMBRE-ENTIER  
EST REMPLACEE PAR  
PRIMAIRE-ARITHMETIQUE

\*\*\*\*\*LA SEQUENCE  
X1 VARIABLE-REELLE  
EST REMPLACEE PAR  
PRIMAIRE-ARITHMETIQUE

\*\*\*\*\*LA SEQUENCE  
=3 00006 PRIMAIRE-ARITHMETIQUE  
/  
X1 00004 PRIMAIRE-ARITHMETIQUE  
EST REMPLACEE PAR  
TERME

\*\*\*\*\*LA SEQUENCE  
X1 00004 PRIMAIRE-ARITHMETIQUE  
+  
00000 00004 TERME  
EST REMPLACEE PAR  
EXPRESSION-ARITHMETIQUE-SIMPLE

\*\*\*\*\*LA SEQUENCE  
(  
00000 00004 EXPRESSION-ARITHMETIQUE-SIMPLE  
)  
EST REMPLACEE PAR  
PRIMAIRE-ARITHMETIQUE

'ALLERA' 'SI' X1-X2 'INFER' 0,001 'ALORS' E2 'SINON' E1 ::

\*\*\*\*\*LA SEQUENCE  
=2 NOMBRE-ENTIER  
EST REMPLACEE PAR  
PRIMAIRE-ARITHMETIQUE

\*\*\*\*\*LA SEQUENCE  
00000 00004 PRIMAIRE-ARITHMETIQUE  
/  
=2 00006 PRIMAIRE-ARITHMETIQUE  
EST REMPLACEE PAR  
TERME

AFFE X2

FSUB1 X1,X2

INFEZ =0.001  
SCD 0  
TRA E2

SIC 0,1  
TRA E1

E2: 'FIN'

```

*****LA SEQUENCE
X2 VARIABLE-REELLE
:=
000000 000004 TERME
EST REMPLACEE PAR
INSTRUCTION-NON-ETIQUETEE

*****LA SEQUENCE
SUITE-D'INSTRUCTIONS
'POINT-VIRGULE'
INSTRUCTION-NON-ETIQUETEE
EST REMPLACEE PAR
SUITE-D'INSTRUCTIONS

*****LA SEQUENCE
X1 VARIABLE-REELLE
EST REMPLACEE PAR
PRIMAIRE-ARITHMETIQUE

*****LA SEQUENCE
X2 VARIABLE-REELLE
EST REMPLACEE PAR
PRIMAIRE-ARITHMETIQUE

*****LA SEQUENCE
X1 000004 PRIMAIRE-ARITHMETIQUE
-
X2 000004 PRIMAIRE-ARITHMETIQUE
EST REMPLACEE PAR
EXPRESSION-ARITHMETIQUE-SIMPLE

*****LA SEQUENCE
=0.001 NOMBRE-REEL
EST REMPLACEE PAR
PRIMAIRE-ARITHMETIQUE

*****LA SEQUENCE
000000 000004 EXPRESSION-ARITHMETIQUE-SIMPLE
'INFIEUR'
=0.001 000004 PRIMAIRE-ARITHMETIQUE
EST REMPLACEE PAR
PRIMAIRE-BOOLEEN

*****LA SEQUENCE
E2 ETIQUETTE
EST REMPLACEE PAR
EXPRESSION-DE-DESIGNATION-SIMPLE

*****LA SEQUENCE
E1 ETIQUETTE
EST REMPLACEE PAR
EXPRESSION-DE-DESIGNATION-SIMPLE

*****LA SEQUENCE
'SI'
000000 PRIMAIRE-BOOLEEN
'ALORS'
EXPRESSION-DE-DESIGNATION-SIMPLE
'SINON'
EXPRESSION-DE-DESIGNATION-SIMPLE
EST REMPLACEE PAR
EXPRESSION-DE-DESIGNATION

*****LA SEQUENCE
'ALLERA'
EXPRESSION-DE-DESIGNATION
EST REMPLACEE PAR
INSTRUCTION-NON-ETIQUETEE

*****LA SEQUENCE
SUITE-D'INSTRUCTIONS
'POINT-VIRGULE'
INSTRUCTION-NON-ETIQUETEE
EST REMPLACEE PAR
SUITE-D'INSTRUCTIONS

*****LA SEQUENCE
E2 ETIQUETTE
:
EST REMPLACEE PAR
INSTRUCTION

*****LA SEQUENCE
SUITE-D'INSTRUCTIONS
'POINT-VIRGULE'
INSTRUCTION
EST REMPLACEE PAR
SUITE-D'INSTRUCTIONS

*****LA SEQUENCE
'DEBUT'
SUITE-D'INSTRUCTIONS
'FIN'
EST REMPLACEE PAR
PROGRAMME
    
```

E2 FIN

Commentaire concernant les deux planches précédentes :

Le programme compilé calcule  $\sqrt{3}$  par une méthode itérative ; le voici :

```

début réel X1,X2 ; X2 := 3 ;
E1 : X1 := X2 ; X2 := (X1+3/X1)/2 ;
allera si X1-X2 < 0.001 alors E2 sinon E1 ;
E2 : fin

```

sur les planches il apparait souligné. A droite sont imprimés les résultats de l'analyse et à gauche les macro-instructions générées dont voici la liste avec quelques commentaires :

	DEBUT		début du programme ;
	RESER	X1,1	réserver une mémoire pour le réel X1 ;
	RESER	X2,1	réserver une mémoire pour le réel X2 ;
	CONER	1 =3	convertir l'entier 3 en réel et l'empiler ;
	AFFE	X2	affecter le sommet de la pile à X2 ;
E1	ANATO	X2	empiler X2 ;
	AFFE	X1	affecter le sommet de la pile à X1 ;
	CONER	1 =3	convertir l'entier 3 en réel et l'empiler ;
	FDIVZ	X1	diviser le sommet de la par X1 et empiler le résultats ;
	FADD	3 X1	ajouter X1 au sommet de la pile et empiler le résultats ;
	CONER	1 =2	convertir l'entier 2 en réel et l'empiler ;
	FDIV	4	diviser l'avant-sommet de la pile par son sommet et empiler le résultat ;
	AFFE	X2	affecter le sommet de la pile à X2 ;
	FSUB	1 X1,X2	soustraire X2 à X1 et empiler le résultat ;
	INFE	2 =0.001	si le sommet de la pile est inférieur à 0.001 alors le remplacer par 1 sinon par 0 ;
	SCD	0	si le sommet de la pile est égal à 0 alors sauter à l'instruction qui suit l'instruction SIC portant le numéro 0 sinon exécuter l'instruction qui suit ;
	TRA	E2	aller à E2 ;
	SIC	0,1	cette instruction porte le numéro 0, son effet est de sauter à l'instruction FEC portant le numéro 1 ;
	TRA	E1	allera à E1 ;
	FEC	1	cette instruction porte le numéro 1, son action est sans effet ;
E2	FIN		fin du programme.

## CONCLUSION

Après cet exposé qui s'est voulu méthodique, nous aimerions terminer sur quelques remarques dictées par l'expérience.

Si nous avons décrit en premier lieu la méthode d'analyse utilisant des relations de précédence sur la totalité du vocabulaire, ceci est uniquement dû à sa simplicité. En fait cette méthode fut notre dernière découverte. Elle s'est montrée très efficace et le fait qu'elle permette d'analyser d'une façon déterministe des textes, non analysables, par un automate à une pile déterministe, en est sûrement la principale raison. On peut cependant lui faire un reproche, celui de devoir introduire des règles superflues dans la grammaire, afin d'obtenir des relations de précédence incompatibles deux à deux. Les règles IDENTIFICATEUR ::= IDENTIFICATEUR-BIS, TERME ::= TERME-BIS etc. de la grammaire Algol de précédence totale du paragraphe 5.1 en sont des exemples. Nous n'avons pas présenté d'application de cet analyseur sous sa forme non-déterministe c'est à dire sous sa forme la plus générale. Il semble cependant qu'il se prêterait très bien à l'analyse de langages naturels, les relations de précédence permettant d'éviter de nombreuses impasses dans le processus de reconnaissance. Le fait que les grammaires utilisées pour ces langages soient souvent ambiguës n'est pas gênant puisque cet analyseur permet de trouver toutes les structures d'un texte donné.

En ce qui concerne la deuxième méthode d'analyse qui utilise des relations de précédence définies sur une partie plus ou moins arbitraire du vocabulaire, on peut se demander pour quelle raison nous avons imposé un certain ordre dans l'application des règles. Sans cette contrainte le problème de trouver de "bonnes" relations de précédence serait devenu trop complexe et il est douteux que l'on eût pu les exprimer à partir de relations plus simples et des opérations définies sur les relations binaires. Il va de soi que cette restriction peut rendre plus difficile la construction d'une grammaire à relations de précédence incompatibles deux à deux, mais cet inconvénient est compensé par la

liberté du choix des opérateurs, liberté qui permet de mieux ajuster son modèle de description syntaxique. En général on est d'ailleurs amené à choisir comme opérateurs les symboles qui effectivement jouent un rôle d'opérateur au sens habituel du terme. Les métavariabes OPERATEUR-DE-RELATION, OPERATEUR-ADDITIF de la grammaire du paragraphe 5.2 en sont des exemples typiques. Enfin rappelons une caractéristique bien agréable de cette méthode : celle de ne pas avoir à procéder par essais successifs pour traiter certaines ambiguïtés comme celles dues à l'impossibilité d'exprimer le lien entre l'endroit où une variable est utilisée et l'endroit où elle est déclarée.

## REFERENCES

- BC M. Brasseur et J. Cohen, Algorithmes d'analyse syntaxique pour langages "context-free", Chiffres, vol 8, n° 2 et 3, 1965.
- BGL L. Bolliet, N. Gastinel, et P.J. Laurent, Un nouveau langage scientifique ALGOL manuel pratique, Herrmann, Paris, 1964.
- Bol L. Bolliet, Notation et processus de traduction des langages symboliques, thèse, Université de Grenoble, Juin 1967.
- Bou J.C. Boussard, Etude et réalisation d'un compilateur ALGOL 60 sur calculatrice du type IBM 7090/94 et 7040/44, Thèse, Université de Grenoble, Juin 1964.
- Ch N. Chomsky, On certain formal properties of grammars, Inf. and Control, vol 2, Décembre 1959.
- Co-1 A. Colmerauer, Notions d'opérateurs dans une grammaire "context-free", RIRO, n° 2, 1967.
- Co-2 A. Colmerauer, Total precedence relations, a paraître dans J.ACM.
- Fl-1 R.W. Floyd, Syntactic analysis and operator precedence, J.ACM, vol 10, Juillet 1963.
- Fl-2 R.W. Floyd, Bounded context syntactic analysis, Comm. ACM, vol 7, Février 1964.
- Fl-3 R.W. Floyd, Non-deterministic algorithms, Publication interne, Carnegie Institute of Technology, Pittsburgh, Novembre 1966.
- GG S. Ginsburg et S. Greibach, Deterministic context-free languages, Inf. and Control, n° 9, 1966.

- GL M. Gross et A. Lentin, Notions sur les grammaires formelles, Gauthiers-Villars, 1967
- GP T.H. Griffiths et S.R. Petrick, On the relative effecicencies of context-free grammar recogniezers, Comm. ACM, Mai 1965
- Ir-1 F.T. Irons, A syntax directed compiler for Algol 60, Comm. ACM, vol 4, Janvier 1961.
- Ir-2 F.T. Irons, An error-correcting parse algorithm, Comm. ACM, vol 6, Novembre 1963.
- Kn D.E. Knuth, On the translation of languages from left to right, Inf. And Control, n° 8, 1965.
- Na P. Naur, Report on the algorithmic language Algol 60, Comm. ACM, vol 3 Mai 1960 ;  
Revised report on the algorithmic language Algol 60, Comm. ACM, vol 6, Janvier 1963.
- Pa-1 C. Pair, Arbres piles et compilation, RFTI n° 3, 1964.
- Pa-2 C. Pair, Etude de la notion de pile application à l'analyse syntaxique, Thèse, Université de Nancy, Décembre 1965.
- Wa S. Warshall, A theorem on boolean matrices, J. ACM, vol 9, Janvier 1962.
- WH N. Wirth et C.A.R. Hoare, A contribution to the developpement of Algol, Comm. ACM, vol 9, Juin 1966.
- WW N. Wirth et H. Weber, Euler : a generalization of Algol and its formal definition, Comm. ACM, Janvier 1966.

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*



