Détermination de certaines caractéristiques des grammaires et des langages "context-free"

François Martin

T H E S E

présentée à la Faculté des Sciences
de l'Université de Grenoble
pour obtenir le grade de
Docteur de troisième cycle "Mathématiques Appliquées"

par

François MARTIN

~~~~~~~~~~

DETERMINATION DE CERTAINES CARACTERISTIQUES DES GRAMMAIRES ET LANGAGES "CONTEXT-FREE".

~~~~~~~~~~~~

Thèse soutenue le 8 mai 1969,
devant la commission d'examen :

MM.	B. VAUQUOIS	Président
	L. BOLLIET	Examinateurs
	J.C. BOUSSARD	Examinateurs

FACULTE DES SCIENCE

UNIVERSITE DE GRENOBLE

L I S T E des P R O F E S S E U R S

-:-:-:-:-:-:-:-:-:-:-:-:-:-

DOYEN HONORAIRE : M. MORET

DOYEN : M. BONNIER

PROFESSEURS TITULAIRES :

MM.	NEEL Louis	Chaire de Physique expérimentale
	HEILMANN René	Chaire de Chimie
	KRAVTCHENKO Julien	Chaire de Mécanique Rationnelle
	CHABAUTY Claude	Chaire de calcul différentiel et Intégral
	BENOIT Jean	Chaire de Radioélectricité
	CHENE Marcel	Chaire de Chimie Papetière
	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
	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
Mlle	LUTZ Elisabeth	Mathématiques
	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
	DESSEAUX Georges	Chaire de Physiologie Animale
	PILLET E.	Chaire de Physique Industrielle et Electrotechniqu
	YOCCOZ Jean	Chaire de Physique Nucléaire Théorique
	DEBELMAS Jacques	Chaire de Géologie Générale
	GERBER R.	Mathématiques
	PAUTHENET R.	Electrotechnique
	VAUQUOIS B.	Chaire de calcul électronique
	BARJON R.	Physique Nucléaire
	BARBIER Jean-Claude	Chaire de Physique
	SILBER R.	Mécanique des Fluides
	BUYLE-BODIN Maurice	Chaire d'Electronique
	DREYFUS B.	Thermodynamique

PROFESSEURS TITULAIRES (suite)

MM.	KLEIN J.	Mathématiques
	VAILLANT F.	Zoologie et Hydrobiologie
	ARNAUD Paul	Chaire de Chimie
	SENGEL P.	Chaire de Zoologie
	BARNOUD F.	Chaire de Biozynthèse de la Cellulose
	BRISSONNEAU P.	Physique
	GAGNAIRE	Chaire de Chimie Physique
Mme	KOFLER L.	Botanique
	DEGRANGE Charles	Zoologie
	PEBAY-PEROULA J.C.	Physique
	RASSAT A.	Chaire de Chimie Systématique
	DUCROS P.	Vhaire de Cristallographie Physique
	DODU Jacques	Chaire de Mécanique Appliquée I.U.T.
	ANGLES D'AURIAC P.	Mécanique des Fluides
	LACAZE A.	Thermodynamique

PROFESSEURS SANS CHAIRE

MM.	GIDON P.	Géologie et Mnéralogie
	GIRAUD P.	Géologie
	PERRET R.	Servomécanisme
Mme	BARBIER M.J.	Electrochimie
Mme	SOUTIF J.	Physique
	COHEN J.	Electrotechnique
	DEPASSEL R.	Mécanique des Fluides
	GASTINEL N.	Mathématiques Appliquées
	GLENAT R.	Chimie
	BARRA J.R.	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

PROFESSEURS ASSOCIES

MM.	NAPP-ZINN	Botanique
	RODRIGUES Alexandre	Mathématiques Pures
	STANDING Kenneth	Physique Nucléaire

MAITRES DE CONFERENCES

MM.	LANCIA Roland	Physique Atomique
Mme	KAHANE J..	Physique
	DEPORTES C.	Chimie
Mme	BOUCHE L.	Mathématiques
	SARROT-REYNAUD	Géologie Propédeutique

MAITRES DE CONFERENCES (suite)

Mme	BONNIER M.J.	Chimie
MM.	KAHANE A.	Physique Générale
	DOLIQUE J.M.	Electronique
	BRIERE G.	Physique M.P.C.
	DESRE G.	Chimie S.P.C.N.
	LAJZROWICZ J.	Physique M.P.C.
	VALENTIN P.	Physique M.P.C.
	BERTRANDIAS J.P.	Mathématiques Appliquées T.M.P.
	LAURENT P.J.	Mathématiques Appliquées T.M.P.
	CAUBET J.P.	Mathématiques Pures
	PAYAN J.J.	Mathématiques
Mme	BERTRANDIAS F.	Mathématiques Pures M.P.C.
	LONGEGUEUE J.P.	Physique
	NIVAT M.	Mathématiques Appliquées
	SOHM J.C.	Electrochimie
	ZADWORNY F.	Electronique
	DURAND F.	Chimie Physique
	CARLER 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
	BOUVARD Maurice	Hydrologie
	RICHARD Lucien	Botanique
	PELMONT Jean	Physiologie Animale
	BLOCH D.	Electrotechnique I.P.
	BOUSSARD J.Claude	Mathématiques Appliquées I.P.
	MOREAU René	Hydraulique I.P.
	BRUGEL L.	Energétique I.U.T.
	SIBILLE R.	Construction Mécanique I.U.T.
	ARMAND Yves	Chimie I.U.T.
	BOLLIET Louis	Informatique I.U.T.
	KUHN Gérard	Energétique I.U.T.
	GERMAIN J.P.	Construction Mécanique I.U.T.
	CONTE René	Thermodynamique
	JOLY Jean René	Mathématiques Pures
Mme	PIERY Yvette	Biologie Animale
	BENZAKEN Claude	Mathématiques Appliquées

MAITRE DE CONFERENCES ASSOCIES

MM.	SAWCZUK A.	Mécanique des Fluides
	CHEEKE J.	Thermodynamique
	YAMADA O.	Physique du Solide
	NATR Lubomir	B.M.P.V.
	NAYLOR Arch	Physique Industrielle
	SILBER Léo	Radioélectricité
	NOZAKI Akihiro	Mathématiques Appliquées
	RUTLEDGE Joseph	Mathématiques Appliquées
	DONOHO Paul	Physique Générale
	EGGER Kurt	B.M.P.V.

Je tiens à remercier ici :

Monsieur le Professeur B. VAUQUOIS, directeur du Centre d'Etudes pour la Traduction Automatique, qui a bien voulu me faire l'honneur de présider ce Jury ;

Monsieur L. BOLLIET, maître de conférences, qui a accepté de faire partie du Jury, après m'avoir sans cesse encouragé dans mon travail ;

Monsieur J.C. BOUSSARD, maître de conférences, qui a accepté de faire partie du Jury, après avoir supervisé la fin de ce travail.

Le Fonds National (Suisse) de la Recherche Scientifique, pour la bourse de "jeune chercheur" qui m'a permis de faire cette thèse ;

Monsieur le Professeur J.B. GRIZE, des Universités de Neuchâtel et autres lieux, qui s'est intéressé à ce travail en tant que "parrain" désigné par le Fonds National ;

Monsieur le Professeur P. BANDERET, directeur du Centre de Calcul Electronique de l'Université de Neuchâtel, qui a tout fait pour faciliter la rédaction de cette thèse ;

Messieurs J. COHEN, qui m'a donné l'idée directrice de ce travail, et qui en a supervisé le début ;

et A. COLMERAUER, à qui je dois beaucoup d'idées et de conseils ;

mes amis de l'Institut de Mathématiques Appliquées de Grenoble, et notamment LAURENT TRILLING, PHILIPPE JORRAND, NGUYEN-HUU-DUNG, NGUYEN-DINH-XUAN, JACQUES COURTIN, GILLES DAGAND et ANDRE ARNOLD, qui se sont à divers titres intéressés à mon travail.

les responsables du système DIAMAG I, en particulier Messieurs J. BELLINO et J.P. VERJUS, qui ont passé des heures à rétablir un système que j'avais "coincé";

Monsieur O. LECARME, à qui je dois l'essentiel de mes connaissances en programmation ;

mes collègues du Centre de Calcul de Neuchâtel, qui ont pallié mon absence virtuelle durant le temps de ma rédaction ;

Madame Y. JAUSLIN, qui a dactylographié ce texte avec patience et dévouement ;

tous ceux enfin qui ont contribué à la réalisation matérielle de cet ouvrage.

∿∿∿∿∿∿∿∿∿∿∿∿∿∿∿∿∿∿∿∿∿∿

TABLE DES MATIERES

à G . . .

INTRODUCTION

Le premier problème abordé, dans le cadre de ce travail, était le suivant : étant donnée une grammaire "context-free", déterminer toutes les paires de symboles pouvant être adjacents dans une phrase (terminale ou non) dérivable à partir de l'axiome de la grammaire. Cette question s'inscrivait dans le contexte des recherches de J. COHEN sur l'analyse syntaxique et la compilation des langages de programmation définis par une grammaire C.F. ([12]).

Ayant résolu ce problème, nous nous sommes intéressés aux questions qui s'étaient posées en cours de route. Par exemple, l'algorithme utilisé nécessitait comme donnée une grammaire sans "symboles vides"; après avoir modifié dans ce sens, "à la main", la grammaire d'ALGOL, étudiée comme premier exemple, il est apparu que cette transformation était programmable. De même pour la suppression des deux catégories de symboles inutiles, que nous avons appelés "parasites" et "inaccessibles".

Quelle forme fallait-il adopter pour représenter la grammaire? Les nombreux programmes d'analyse syntaxique existant ou en travail à Grenoble ([10], [11], [12], [13], [14], [15], [18], [19], [20]) avaient chacun leurs conventions particulières. Pour la représentation interne de la grammaire, les conventions adoptées par A. COLMERAUER ont paru le plus générales, et ont été adoptées ici (ce qui a permis l'utilisation de plusieurs procédures écrites par ce dernier).

Pour la représentation externe (sur cartes perforées), la plupart des programmes d'analyse exigeaient que la grammaire soit donnée sous forme symbolique, à l'image de la représentation interne ; et que la phrase du langage à analyser soit elle aussi sous forme d'une suite de codes symboliques.

Il nous a paru intéressant d'établir un programme destiné à mettre sous forme symbolique une grammaire écrite "en clair"; ainsi qu'un programme "éditeur", traduisant une phrase du langage en une suite de codes. Pour la grammaire "en clair", nous avons admis qu'il suffisait de donner l'ensemble des règles et l'axiome; le programme se chargeant de déterminer le vocabulaire selon un critère simple: les symboles non-terminaux sont ceux qui se trouvent en partie gauche d'une règle au moins, les terminaux ceux qui ne figurent qu'en partie droite, mais dans une règle au moins.

Ainsi se trouva réalisée la chaîne de programmes constituant ce travail, et effectuant les opérations suivantes: entrée d'une grammaire C.F., détermination des vocabulaires terminal et non-terminal, transformation de la grammaire par élimination des symboles "parasites", "inaccessibles" et "vides", calcul de certaines relations, édition d'une phrase de langage. Le maillon suivant, qui devrait être un programme d'analyse syntaxique utilisant les relations calculées, n'existe pas dans ce travail. En revanche, les programmes d'analyse cités plus haut s'enchaînent aisément ici.

Les problèmes traités dans ce travail étaient suffisamment simples pour ne pas nécessiter de nombreuses lectures préalables. A part quelques astuces techniques, comme l'algorithme de WARSHALL pour la fermeture transitive d'une matrice booléenne ([9]), les méthodes à utiliser et la façon adéquate d'enchaîner les algorithmes étaient intuitivement évidents. Après coup, il fallut néanmoins les justifier.

Or, certaines des propriétés entrevues intuitivement étaient plus ou moins liées au fait que nous avions considéré, au départ, la grammaire comme définie par ses règles et son axiome seulement; en d'autres termes, que les vocabulaires

étaient considérés comme contenus implicitement dans les règles. Formalisée, cette idée conduisait à la notion de vocabulaires associés à un ensemble de règles; les règles étant construites avec des symboles appartenant à un ensemble assez vague (pratiquement : l'ensemble des suites de caractères pouvant être perforés sur carte).

Nous avons choisi de reconstruire sur cette notion toute la partie théorique de ce travail, plutôt que de nous référer à la littérature, les définitions peu orthodoxes que nous posons au départ permettant de déduire exactement les propriétés des algorithmes programmés ici.

I. DEFINITIONS et PROPRIETES

Dans tout ce chapitre, on considère un même ensemble E, fini ou infini, dont les éléments x, y, ..., sont appelés symboles.

1. Chaînes. Règles. Vocabulaire.

Rappels.

a) Une chaîne Σ sur E est une suite finie d'occurrences d'éléments de E.

$$\Sigma = x_1 x_2 \ldots x_n$$

On désigne par Λ la chaîne vide.

$$\Lambda =$$

b) Etant données deux chaînes $\Phi = x_1 \ldots x_n$ et $\Psi = y_1 \ldots y_m$, on désigne par $\Phi\ \Psi$ la chaîne $x_1 \ldots x_n\ y_1 \ldots y_m$, obtenue par concaténation des chaînes Φ et Ψ.

c) L'ensemble des chaînes sur E, noté E^*, est le monoïde libre sur E. Si A est une partie quelconque de E, on désigne de même par A^* l'ensemble des chaînes sur A.

Remarquons que l'ensemble des chaînes construites sur la partie vide de E se réduit à la seule chaîne vide. Autrement dit :

$$\emptyset^* = \{ \Lambda \} \tag{1.1}$$

On appelle règle un couple $T = (x,\Sigma)$, où x est un symbole et Σ une chaîne. On écrira

$$T : x ::= \Sigma$$

ou, plus simplement

$$x ::= \Sigma$$

x est la partie gauche de la règle, Σ la partie droite.

Soit R un ensemble fini de règles. On appelle vocabulaire associé à R, et on désigne par V[R], l'ensemble des symboles qui apparaissent dans une règle au moins de R, soit comme partie gauche, soit dans la partie droite. On appelle vocabulaire non-terminal associé à R, et on désigne par $V_N[R]$, l'ensemble des symboles qui sont partie gauche d'une règle au moins de R.

On appelle vocabulaire terminal associé à R, et on désigne par $V_T[R]$, l'ensemble $V[R] - V_N[R]$.

Remarquons que V[R], $V_T[R]$, $V_N[R]$ sont finis, et que $V_T[R]$ et $V_N[R]$ forment une partition de V[R].

Dans la suite, lorsqu'il n'y aura pas d'équivoque sur l'ensemble R considéré, on écrira simplement V, V_T, V_N pour V[R], $V_T[R]$, $V_N[R]$. De même on appellera symboles les seuls éléments de V, et chaînes les seules chaînes sur V , autrement dit les éléments de V*. Les éléments de V_T seront appelés symboles terminaux, ceux de V_N symboles non-terminaux, et ceux de V_T^* chaînes terminales. La chaîne vide Λ sera considérée comme terminale.

2. Dérivations.

Etant donné un ensemble fini de règles R, on définit entre les chaînes une relation binaire, notée $\longrightarrow$ ou $\xrightarrow[R]{}$.

Si	$\Sigma, \Phi \in V^*$
on dira que	$\Sigma \xrightarrow[R]{} \Phi$
s'il existe	$x \in V_N, \Psi, \chi, \Omega \in V^*$
tels que	$\Sigma = \Psi \; x \; \Omega,$
	$\Phi = \Psi \; \chi \; \Omega,$
et que	$x ::= \chi$
soit une règle de R.	

Une dérivation de longueur n est une suite finie de chaînes $\Sigma_0, \Sigma_1, \ldots, \Sigma_n$, telle que pour tout i, $0 \leq i \leq n-1$, on ait $\Sigma_i \longrightarrow \Sigma_{i+1}$. Plus précisément, on définira une dérivation D

de longueur n comme une suite finie de couples (Σ_i, u_i), $0 \leq i \leq n$, où les Σ_i sont des chaînes, et où, pour tout i, $0 \leq i \leq n-1$, u_i est une occurrence de symbole non-terminal dans Σ_i (dite <u>occurrence associée</u> à Σ_i) ;

telle que $\forall i,\ 0 \leq i \leq n-1$:

$$\Sigma_i = \Psi_i\ u_i\ \Omega_i\ ,$$
$$\Sigma_{i+1} = \Psi_i\ \chi_i\ \Omega_i\ ,$$

et que $u_i ::= \chi_i$ soit une règle.

On écrira

$$\Sigma_o \longrightarrow \Sigma_1 \longrightarrow \ldots \longrightarrow \Sigma_{n-1} \longrightarrow \Sigma_n$$

On dit aussi que D est une Σ_o-dérivation de Σ_n dans R. Les chaînes $\Sigma_1, \ldots, \Sigma_{n-1}$ sont dites <u>intérieures</u> à la dérivation. Si Σ_n est une chaîne terminale, on dit que D est une dérivation terminale. Enfin, si n=0, la dérivation se réduit à une seule chaîne Σ_o.

On définit encore une relation binaire entre les chaînes, notée $\Longrightarrow$ ou $\underset{R}{\Longrightarrow}$. Si $\Sigma, \Phi \in V^*$, on dira que $\Sigma \underset{R}{\Longrightarrow} \Phi$ s'il existe dans R une Σ-dérivation de Φ. Notons que si la dérivation est de longueur 0, cela revient à dire que $\Sigma = \Phi$.

<u>Lemme 2.1</u> Soit $\Phi \longrightarrow \ldots \longrightarrow \Psi$ une dérivation de longueur n. Si $\Phi = \Phi_1\ \Phi_2\ \ldots\ \Phi_k$, on peut associer, de façon canonique, à chaque Φ_i une sous-chaîne Ψ_i de Ψ, telle que

$$\Psi = \Psi_1\ \Psi_2\ \ldots\ \Psi_k\ ;$$

que, pour tout i, $\Phi_i \Longrightarrow \Psi_i$;

et que la dérivation

$$\Phi \longrightarrow \Psi_1\ \Phi_2 \ldots\ \Phi_k \longrightarrow \Psi_1\ \Psi_2\ \ldots\ \Phi_k \longrightarrow \ldots \longrightarrow \Psi$$

soit de longueur n.

<u>Démonstration</u>. Elle se fait par récurrence sur n.

a) Si $n = 0$, $\Phi = \Psi$ et la propriété est évidente.

b) Supposons la propriété vraie pour $n=m$.

Soit $D : \Phi \longrightarrow \ldots \longrightarrow \Sigma \longrightarrow \Psi$
une dérivation de longueur m + 1 . La dérivation

$$\Phi \longrightarrow \ldots \longrightarrow \Sigma$$

est de longueur m. D'après l'hypothèse de récurrence, à chaque Φ_i est associée Σ_i, telle que

$$\Sigma = \Sigma_1 \ldots \Sigma_j \ldots \Sigma_k$$

$$\Phi_i \Longrightarrow \Sigma_i, \quad \forall\, i$$

et que la dérivation

$$D' : \Phi = \Phi_1 \ldots \Phi_j \ldots \Phi_k \longrightarrow \Sigma_1 \ldots \Phi_j \ldots \Phi_k \longrightarrow \ldots$$
$$\longrightarrow \Sigma_1 \ldots \Sigma_j \ldots \Phi_k \longrightarrow \ldots$$
$$\longrightarrow \Sigma_1 \ldots \Sigma_j \ldots \Sigma_k = \Sigma$$

soit une longueur m.

Soit u l'occurrence associée à Σ dans la dérivation D, et soit Σ_j la sous-chaîne de Σ contenant u. Il existe une règle

$$u ::= \chi ,$$

et on a

$$\Sigma_j = \theta\ u\ \omega ,$$
$$\Sigma = \Sigma_1 \ldots \theta\ u\ \omega \ldots \Sigma_k ,$$
$$\Psi = \Sigma_1 \ldots \theta\ \chi\ \omega \ldots \Sigma_k .$$

$$\Phi_j \longrightarrow \ldots \longrightarrow \Sigma_j \longrightarrow \theta\ \chi\ \omega$$

est une dérivation, donc $\Phi_j \Longrightarrow \theta\ \chi\ \omega$. En ajoutant, dans la dérivation D', la chaîne $\Sigma_1 \ldots \theta\ \chi\ \omega \ldots \Phi_k$ après la chaîne $\Sigma_1 \ldots \Sigma_j \ldots \Phi_k$, et en remplaçant, dans les chaînes suivantes, Σ_j par $\theta\ \chi\ \omega$, on obtient une dérivation de longueur m + 1 :

$$\Phi = \Phi_1 \ldots \Phi_j \ldots \Phi_k \longrightarrow \Sigma_1 \ldots \Phi_j \ldots \Phi_k \longrightarrow \ldots$$
$$\longrightarrow \Sigma_1 \ldots \Sigma_j \ldots \Phi_k \longrightarrow \Sigma_1 \ldots \theta\ \chi\ \omega \ldots \Phi_k \longrightarrow \ldots$$
$$\longrightarrow \Sigma_1 \ldots \theta\ \chi\ \omega \ldots \Sigma_k = \Sigma$$

La propriété est ainsi démontrée, avec $\Psi_i = \Sigma_i$, pour $i \neq j$, et $\Psi_j = \theta\ \chi\ \omega$.

CQFD.

3. Quelques relations et ensembles.

R étant un ensemble fini de règles, on définit, entre les symboles de V[R], trois relations binaires, notées respectivement σ_R, γ_R, δ_R, ou simplement σ, γ, δ.

x et y étant des éléments de V, on dira que y est un successeur de x, et on écrira $x\sigma_R y$, s'il existe une x-dérivation d'une chaîne contenant y.

(3.1) $x \sigma_R y$

s'il existe $\Sigma, \Phi \in V^*$

tels que $x \underset{R}{\Longrightarrow} \Sigma y \Phi$

Si $\Sigma = \Lambda$, y est un successeur gauche de x, et on écrit

$x \gamma_R y$.

Si $\Phi = \Lambda$, y est un successeur droit de x, et on écrit

$x \delta_R y$.

Donc :

(3.2) $x \gamma_R y$

s'il existe $\Phi \in V^*$

tel que $x \underset{R}{\Longrightarrow} y \Phi$

(3.3) $x \delta_R y$

s'il existe $\Sigma \in V^*$

tel que $x \underset{R}{\Longrightarrow} \Sigma y$

Soit A une partie quelconque de V. Appelons $\tilde{A}$ l'ensemble des symboles x tels qu'il existe une x-dérivation d'une chaîne de A^*.

(3.4) $$\tilde{A} = \{ x \in V \mid \exists \Sigma \in A^* , x \Longrightarrow \Sigma \}$$

Posons $A_0 = A$, et pour tout $i \geq 0$, appelons A_{i+1} la réunion de A_i et de l'ensemble des symboles y qui sont partie gauche d'une règle, dont la partie droite appartient à A_i^*.

(3.5)
$$A_0 = A$$
$$A_{i+1} = A_i \cup \{ y \in V_N \mid \exists T \in R, T : y ::= \Sigma, \Sigma \in A_i^* \}$$

Proposition 3.6

Il existe un entier N , majoré par le nombre d'éléments de V , tels que

$$A_N = \bigcup_{i=0}^{\infty} A_i = \tilde{A}$$

Démonstration.

1° La suite A_0 , A_1 ,... étant croissante et majorée par l'ensemble V , il existe des ensembles de la suite qui sont égaux. Or , on voit d'après (3.5) que si $A_n = A_{n-1}$, alors $A_{n+1} = A_n$. Il existe donc un N tel que, pour tout $m > N$, $A_m = A_N$. D'où $A_N = \bigcup A_i$.

2° Montrons , par récurrence sur i , que $A_i \subset \tilde{A}$.

a) Pour $i = 0$, on a $A_0 = A \subset \tilde{A}$, d'après (3.4) et la relation $x \Longrightarrow x$.

b) Supposons que $A_i \subset \tilde{A}$, et soit $x \in A_{i+1}$. Ou bien x appartient à A_i , donc à $\tilde{A}$. Ou bien il existe une règle $x ::= y_1 \ldots y_n$, avec $y_j \in A_i$. D'après l'hypothèse de récurrence, pour tout j , $y_j \in \tilde{A}$, ce qui signifie que $y_j \Longrightarrow \Sigma_j \in A^*$. Il existe donc une dérivation

$$x \longrightarrow y_1 \ldots y_n \longrightarrow \ldots \longrightarrow \Sigma_1 \ldots \Sigma_n$$

avec $\Sigma_1 \ldots \Sigma_n \in A^*$, ce qui implique que $x \in \tilde{A}$.

On a ainsi montré que $\bigcup A_i \subset \tilde{A}$.

3° Si $x \in \bar{A}$, il existe une x-dérivation de $\Sigma \in A^*$.
On montrera , par récurrence sur la longueur de la dérivation , que $x \in \bigcup A_i$.
a) Si la dérivation est une longueur 0 , on a $x = \Sigma$, donc $x \in A = A_0$, donc $x \in \bigcup A_i$.
b) Supposons que $y \in \bigcup A_i$ s'il existe une y-dérivation de $\phi \in A^*$, de longueur $\leq n$. Soit

$$x \longrightarrow y_1 \ldots y_k \longrightarrow \ldots \longrightarrow \Sigma \in A^*$$

une dérivation de longueur $n+1$. D'après le lemme 2.1 , $\Sigma = \Sigma_1 \ldots \Sigma_k$, et , pour tout j , il existe une y_j-dérivation de $\Sigma_j \in A^*$, de longueur $\leq n$. D'après l'hypothèse de récurrence , pour tout j ,

$$y_j \in \bigcup A_i = A_N ,$$

donc $y_1 \ldots y_k \in A_N^*$,
d'où $x \in A_{N+1} = A_N = \bigcup A_i$.

CQFD.

4. Grammaire.

Soit R un ensemble fini de règles , et S un élément de $V_N[R]$. On vérifie aisément que le quadruplet $(V_T[R] , V_N[R] , S, R)$ est une grammaire de Chomsky de type 2 , c'est-à-dire "context-free".

Celle-ci étant uniquement déterminée par R et S , on conviendra ici d'appeler grammaire le couple G=(R , S) , où R est un ensemble fini de règles, et S un symbole de $V_N[R]$, appelé axiome de la grammaire .

Rappelons qu'on appelle langage défini par la grammaire G l'ensemble L[G] des chaînes terminales ϕ , telles qu'il existe une S-dérivation de ϕ dans R.

(4.1) $$L[G] = \{ \phi \in V_T^* \ [R] \mid S \Longrightarrow \phi \}$$

Rappelons encore que deux grammaires sont dites <u>équivalentes</u> si elles définissent le même langage.

Enfin, deux symboles x et y, terminaux ou non-terminaux, seront dit <u>compatibles</u> s'il existe une S-dérivation d'une chaîne contenant la chaîne x y. Cette nouvelle relation binaire entre les symboles sera notée ϵ_G, ou simplement ϵ.

(4.2)
Si $x, y \in V$
on dira que $x \ \epsilon_G \ y$
s'il existe $\Sigma, \Phi \in V^*$
tels que $S \underset{R}{\Longrightarrow} \Sigma x y \Phi$

5. <u>Symboles parasites. Symboles inaccessibles.</u>

Soit une grammaire G = (R, S). On appelle <u>symbole parasite</u> tout symbole x tel qu'il n'existe pas de x-dérivation terminale. Un tel symbole est toujours non-terminal, car pour tout y terminal, on a $y \Longrightarrow y$.

Soit P[G] l'ensemble des symboles parasites.

(5.1) $$P[G] = V - \{ x \in V \mid \exists \ \phi \in V_T^* , \ x \Longrightarrow \phi \}$$
d'où $P[G] = V - \tilde{V}_T$
en vertu de (3.4).

<u>Lemme 5.2</u> Si tous les symboles appartenant à la partie droite d'une règle sont non parasites, alors le symbole en partie gauche est non parasite.

<u>Démonstration</u>. Soit la règle $x ::= y_1 \ldots y_k$.
Si les y_i sont non parasites, il existe des chaînes terminales ϕ_i telles que $y_i \Longrightarrow \phi_i$. Alors $x \Longrightarrow \phi_1 \ldots \phi_k$ qui est une chaîne terminale.

CQFD.

On appelle symbole inaccessible tout symbole y qui n'est pas un successeur de l'axiome S.

Soit I[G] l'ensemble des symboles inaccessibles.

(5.3) $$I[G] = V - \{ y \in V \mid S \sigma y \}$$

Lemme 5.4 Si le symbole en partie gauche d'une règle n'est pas inaccessible, alors aucun symbole de la partie droite n'est inaccessible.

Démonstration. Soit la règle $x ::= y_1 \ldots y_k$.

Si $S \sigma x$, il existe Σ, Φ tels que $S \Longrightarrow \Sigma x \Phi$, donc $S \Longrightarrow \Sigma y_1 \ldots y_k \Phi$, d'où $S \sigma y_i$, pour tout i.

CQFD.

Si $P[G] = \emptyset$, on dira que la grammaire G est sans parasites. Si $I[G] = \emptyset$, on dira que G est sans inaccessibles.

6. Symboles vides.

Soit G = (R,S) une grammaire sans parasites, c'est-à-dire telle que pour tout symbole x, il existe une x-dérivation terminale.

On appelle symbole vide tout symbole y tel qu'il existe une y-dérivation de la chaîne vide Λ.

Soit W[G] l'ensemble des symboles vides.

(6.1) $$W[G] = \{ y \in V \mid y \Longrightarrow \Lambda \}$$

d'où $$W[G] = \emptyset$$

en vertu de (1.1) et (3.4) .

G étant sans parasites , il existe une partition de W[G] en deux ensembles $W_1[G]$ et $W_2[G]$. $W_1[G]$ est l'ensemble des symboles z , appelés symboles semi-vides , tels qu'il existe une z-dérivation de Λ et une z-dérivation d'une chaîne terminale ϕ non vide. $W_2[G]$ est l'ensemble des symboles u, appelés symboles essentiellement vides , tels que toute u-dérivation terminale est une dérivation de Λ .

(6.2)
$$W_1[G] = \{ z \in W[G] \mid \exists \phi \in V_T^* , \phi \neq \Lambda , z \Longrightarrow \phi \}$$
$$W_2[G] = W[G] - W_1[G]$$

Lemme 6.3 Si le symbole en partie gauche d'une règle est essentiellement vide , alors tous les symboles de la partie droite sont essentiellement vides. Si toutes les règles , ayant pour partie gauche un même symbole x , ont une partie droite vide ou formée de symboles essentiellement vides , alors x est essentiellement vide .

Démonstration. 1° Soit la règle $x ::= y_1 \ldots y_k$. Considérons un y_j quelconque , $1 \leq j \leq k$, et une y_j-dérivation terminale quelconque, $y_j \longrightarrow \ldots \longrightarrow \phi_j$. La grammaire étant sans parasite , il existe , pour tout $i \neq j$, une y_i-dérivation d'une chaîne terminale ϕ_i. On a donc une dérivation terminale

$$x \longrightarrow y_1 \ldots y_j \ldots y_k \longrightarrow \ldots \longrightarrow \phi_1 \ldots \phi_j \ldots \phi_k .$$

x étant essentiellement vide , $\phi_1 \ldots \phi_j \ldots \phi_k = \Lambda$ donc ϕ_j est vide . Ce qui implique que y_j est essentiellement vide.

2° Soit x tel que toute règle dont il est la partie gauche ait une partie droite vide ou formée de symboles essentiellement vides. Considérons une x-dérivation terminale quelconque.
Elle est de la forme $x \longrightarrow \Lambda$
ou $x \longrightarrow y_1 \ldots y_k \longrightarrow \ldots \longrightarrow \phi$

D'après le lemme 2.1 , $\phi = \phi_1 \ldots \phi_k$, avec $y_i \Longrightarrow \phi_i$

Or, comme les y_i sont essentiellement vides , on a $\phi_i = \Lambda$ pour tout i , donc $\phi = \Lambda$. Ce qui implique que x est essentiellement vide.

CQFD .

Proposition 6.4 Il existe une x-dérivation d'une chaîne terminale non vide si et seulement si le symbole x admet un successeur terminal.

Démonstration. 1° S'il existe une chaîne terminale $\phi \neq \Lambda$, telle que $x \Longrightarrow \phi$, alors ϕ contient au moins un symbole $y \in V_T$, et $x \sigma y$.

2° Soit $y \in V_T$, tel que $x \sigma y$. Il existe des chaînes Φ et Ψ vides ou non, telles que $x \Longrightarrow \Phi\ y\ \Psi$. La grammaire étant sans parasites, pour tout symbole z_i, appartenant à Φ ou à Ψ, il existe une z_i-dérivation terminale. Donc, dans tous les cas, $\Phi \Longrightarrow \phi$ et $\Psi \Longrightarrow \psi$, où ϕ et ψ sont des chaînes terminales. On a donc une dérivation

$$x \longrightarrow \ldots \longrightarrow \Phi\ y\ \Psi \longrightarrow \ldots \longrightarrow \phi\ y\ \psi$$

et $\phi\ y\ \psi$ est une chaîne terminale non vide.

CQFD.

Corollaire 6.5 Un symbole vide x est semi-vide si et seulement si il admet un successeur terminal.

7. Calcul de certaines relations.

Soit une grammaire G = (R,S). Définissons encore trois relations binaires entre les symboles, notées respectivement σ_o, γ_o, δ_o.

x et y étant des symboles, on dira que y est un successeur immédiat de x, et on écrira $x\ \sigma_o\ y$, s'il existe une règle $x ::= \Sigma\ y\ \Phi$. On dira que y est un successeur gauche immédiat de x, et on écrira $x\ \gamma_o\ y$, s'il existe une règle $x ::= y\ \Phi$. On dira que y est un successeur droit immédiat de x, et on écrira $x\ \delta_o\ y$, s'il existe une règle $x ::= \Sigma\ y$.

Proposition 7.1 La relation σ est la fermeture transitive de la relation σ_o . En d'autres termes,

$x \sigma y$

est équivalent à: $x = y$

ou : il existe une suite

$x = x_1 , x_2 , \ldots , x_n = y ,$

telle que $x_i \sigma_o x_{i+1} , 1 \leq i \leq n-1.$

Démonstration. Désignons par σ' la fermeture transitive de la relation σ_o.

1° Si $x \sigma' y$, et si $x \neq y$, il existe une suite de règles

$T_i : x_i ::= \Sigma_i x_{i+1} \Phi_i ,$

avec $x = x_1$ et $y = x_n$,

et une dérivation

$x = x_1 \longrightarrow \Sigma_1 x_2 \Phi_1 \longrightarrow \ldots \longrightarrow \Sigma_1 \ldots \Sigma_{n-1} y \Phi_{n-1} \ldots \Phi_1 ,$

ce qui implique que $x \sigma y$.

2° Si $x \sigma y$, il existe une x-dérivation de $\Sigma y \Phi$, $\Sigma, \Phi \in V^*$. On montre, par récurrence sur la longueur de la dérivation, que ceci implique $x \sigma' y$.

a) Si la dérivation est de longueur 0 , alors $x = \Sigma y \Phi = y$.

b) Supposons la propriété vraie pour une dérivation de longueur m. Soit

$x \longrightarrow \Psi_1 \longrightarrow \ldots \longrightarrow \Psi_m \longrightarrow \Sigma y \Phi$

une dérivation de longueur m+1 . Si la chaîne Ψ_m contient y, alors $x \sigma' y$ d'après l'hypothèse de récurrence. Sinon , si u est l'occurrenceassociée à Ψ_m dans la dérivation , $x \sigma' u$, d'après l'hypothèse de récurrence ; or, on doit avoir une règle $u ::= \Sigma_o y \Phi_o$, donc $u \sigma_o y$, d'où $x \sigma' y$.

CQFD.

Proposition 7.2 Si G est une grammaire sans vide , la relation γ est la fermeture transitive de la relation γ_o .

Démonstration. Désignons par γ' la fermeture transitive de la relation γ_o .

1° Si $x \gamma' y$, et si $x \neq y$, il existe une suite de règles

$$T_i : X_i ::= X_{i+1} \Phi_i ,$$

avec $x = X_1$ et $y = X_n$,

et une dérivation

$$x = X_1 \longrightarrow X_2 \Phi_1 \longrightarrow \ldots \longrightarrow y \Phi_{n-1} \ldots \Phi_1 ,$$

ce qui implique que $x \gamma y$.

2° Si $x \gamma y$, il existe une x-dérivation de $y \Phi$, $\Phi \in V^*$. On montre , par récurrence sur la longueur de la dérivation, que ceci implique $x \gamma' y$.

a) Si la dérivation est de longueur 0 , alors

$$x = y \Phi = y .$$

b) Supposons la propriété vraie pour une dérivation de longueur m . Soit

$$x \longrightarrow \Psi_1 \longrightarrow \ldots \longrightarrow z \Psi_m \longrightarrow y \Phi$$

une dérivation de longueur m + 1 . D'après l'hypothèse de récurrence , $x \gamma' z$. Si $z = y$, la propriété est démontrée. Sinon , z doit être l'occurrence associée à $z \Psi_m$ dans la dérivation ; on doit avoir une règle $z ::= \alpha$, avec $\alpha \neq \Lambda$ puisque la grammaire est sans vide ; et $\alpha \Psi_m = y \Phi$, d'où $\alpha = y \alpha_o$, ce qui implique $z \gamma_o y$. D'où enfin $x \gamma' y$.

CQFD.

Proposition 7.3 Si g est une grammaire sans vides , la relation δ est la fermeture transitive de la relation δ_o.

La démonstration est analogue à celle de la proposition 7.2 .

Proposition 7.4 Si G est une grammaire sans vides et sans inaccessibles ,

$$x \; \varepsilon \; y$$

est équivalent à :

il existe $z, u, v \in V$; $\Phi , \Psi \in V^*$;

tels que $z ::= \Phi \; u \; v \; \Psi$ soit une règle

et que $u \; \delta \; x$,

$v \; \gamma \; y$.

Démonstration. Désignons par ε' la relation définie ci-dessus.

1° Soit $x \; \varepsilon' \; y$. La grammaire étant sans inacessibles , il existe $\Phi_o , \Psi_o \in V^*$ tels que

$$S \Longrightarrow \Phi_o \; z \; \Psi_o .$$

Comme $u \; \delta \; x$ et $v \; \gamma \; y$, il existe $\Phi_1 , \Psi_1 \in V^*$ tels que

$$u \Longrightarrow \Phi_1 \; x ,$$

$$v \Longrightarrow y \; \Psi_1 .$$

Il existe donc une dérivation

$$S \longrightarrow \ldots \longrightarrow \Phi_o \; z \; \Psi_o \longrightarrow \Phi_o \; \Phi \; u \; v \; \Psi \; \Psi_o \longrightarrow \ldots$$

$$\longrightarrow \Phi_o \; \Phi \; \Phi_1 \; x \; y \; \Psi_1 \; \Psi \; \Psi_o ,$$

ce qui implique que $x \; \varepsilon \; y$.

2° Si $x \; \varepsilon \; y$, il existe une S-dérivation de $\Sigma \; x \; y \; \Phi$, de longueur ≥ 1 . Montrons par récurrence sur la longueur de cette dérivation , que ceci implique $x \; \varepsilon' \; y$.

a) Si la dérivation est de longueur 1 , on a la règle $S ::= \Sigma \; x \; y \; \Phi$. Ce qui implique que $x \; \varepsilon' \; y$, puisque $x \; \delta \; x$ et $y \; \gamma \; y$.

b) Supposons la propriété vraie pour une dérivation de longueur n . Soit

$$S \longrightarrow \Psi_1 \longrightarrow \ldots \longrightarrow \Psi_n \longrightarrow \Sigma \; x \; y \; \Phi$$

une dérivation de longueur $n+1$.

- Si ψ_n contient la chaîne x y ,

$$\psi_n = \lceil\ x\ y\ \Omega\ ,$$

alors x ε' y , d'après l'hypothèse de récurrence.

- Si ψ_n ne contient pas x y , soit z l'occurrence associée à ψ_n dans la dérivation , et z ::= x la règle concernée.
La grammaire étant sans vides , on a x $\neq \Lambda$.
Il y a donc 3 possibilités :

- Ou bien, x contient la chaîne x y .

z ::= $\lceil$ x y Ω est une règle

et comme x δ x , y γ y ,

on a x ε' y .

- Ou bien $\psi_n = \lceil\ x\ z\ \Omega$,

et $x = y\ \Omega_o$.

D'après l'hypothèse de récurrence ,

x ε' z ,

donc il existe z_o , u_o , $v_o \in V$; Σ_o , $\Phi_o \in V^*$;

tels que $z_o ::= \Sigma_o\ u_o\ v_o\ \Phi_o$ soit une règle ,

et que $u_o\ \delta\ x$,

$v_o \gamma\ z$.

On a d'autre part la règle

z ::= y Ω_e,

donc z γ_o y ,

d'où , d'après la proposition 7.2 ,

$\psi_o\ \gamma\ y$.

Ce qui permet d'affirmer que

x ε' y .

- Ou bien $\psi_n = \lceil\ z\ y\ \Omega$,

et $x = \lceil_o\ x$.

On montre alors , de la même manière , que x ε' y .

CQFD.

6. Construction d'une grammaire sans parasites.

Soit $G = (R,S)$ une grammaire définissant un langage non vide.

$$L[G] \neq \emptyset$$

$P[G]$ étant l'ensemble des symboles parasites de V, désignons par R' l'ensemble des règles de R qui ne contiennent pas de symboles de $P[G]$. Les vocabulaires associés à R', $V[R']$, $V_T[R']$, $V_N[R']$, seront appelés respectivement V', V_T', V_N'.

Lemme 6.1 Soit $D: x \longrightarrow \ldots \longrightarrow \phi$, où $x \in V_N$, une dérivation de longueur $n \geq 1$ dans R. Si tous les symboles de ϕ sont non parasites, alors D est aussi une dérivation dans R'.

Démonstration. Il suffit de montrer que toutes les chaînes de D sont formées de symboles de V', et que toutes les règles utilisées appartiennent à R'. D'après le lemme 5.2, on montre par récurrence sur n que les règles ne contiennent pas de symboles parasites, donc sont aussi des règles de R'. Et comme tous les symboles contenus dans les chaînes de D figurent dans une de ces règles, ils appartiennent à V'.

CQFD.

Tout symbole de V_N' est partie gauche d'une règle de R', qui est aussi une règle de R; c'est donc un symbole de V_N.

$$(a) \quad V_N' \subset V_N \cap V'$$

D'autre part, la définition de R' implique qu'il n'y a aucun symbole de $P[G]$ dans V'.

$$(b) \quad V' \subset V - P[G]$$

Soit enfin x un symbole de $V_N - P[G]$, donc un symbole non-terminal non parasite. Il existe dans R une dérivation terminale D : $x \longrightarrow \ldots \longrightarrow \phi$ de longueur ≥ 1. Les symboles de ϕ sont non parasites, donc, d'après le lemme 8.1, D est une dérivation dans R'. On en déduit que $x \in V'_N$.

$$(c) \quad V_N - P[G] \subset V'_N$$

De (a), (b) et (c), on tire :

$$V'_N \subset V_N \cap V' \subset V_N \cap (V - P[G]) = V_N - P[G] \subset V'_N$$

d'où finalement :

$$(8.2) \qquad V'_N = V_N \cap V' = V_N - P[G]$$

$$\text{et} \qquad V'_T = V_T \cap V'$$

Le langage L[G] étant supposé non vide, l'axiome S de G n'est pas parasite, car il existe au moins une S-dérivation terminale. D'après (8.2), on voit que $S \in V'_N$. Il est donc possible de prendre S comme axiome d'une nouvelle grammaire $G' = (R', S)$, qu'on appelera aussi $\Pi(G)$.

On montrera que $\Pi(G)$ est une grammaire sans parasites équivalente à G.

Proposition 8.3 La grammaire $G' = \Pi(G)$ est équivalente à G.

$$L[\Pi(G)] = L[G]$$

Démonstration. 1° Soit ϕ une chaîne de L[G]. Il existe une dérivation terminale D: $S \longrightarrow \ldots \longrightarrow \phi$, de longueur ≥ 1, dans R. D'après le lemme 8.1 , D est aussi une dérivation dans R'. Les symboles de ϕ appartiennent à $V' \cap V_T$, donc à V'_T , d'après (8.2). D est donc une S-dérivation terminale dans R' , et ϕ appartient au langage L[G'] .

2° Comme $R' \subset R$, $V'_T \subset V_T$ et $V'_N \subset V_N$, toute S-dérivation terminale dans R' est une S-dérivation terminale dans R.

CQFD.

Proposition 8.4 . La grammaire $G' = \Pi(G)$ est sans parasites.

$$P[\Pi(G)] \neq \emptyset$$

Démonstration. Soit x un symbole quelconque de V'_N . D'après (8.2) , $x \in V_N - P[G]$, donc il existe une dérivation terminale $D: x \longrightarrow \ldots \longrightarrow \phi$, de longueur ≥ 1 , dans R. D'après le lemme 8.1 , D est une dérivation dans R'. Les symboles de ϕ appartiennent à $V' \cap V_T$, donc à V_T' , d'après (8.2). D est donc une x-dérivation terminale dans R' .

CQFD.

9. Construction d'une grammaire sans inaccessibles.

Soit une grammaire G = (R,S). I[G] étant l'ensemble des symboles inaccessibles de V , désignons par R' l'ensemble des règles de R qui ne contiennent pas de symboles de I[G]. Les vocabulaires associés à R' seront appelés V' , V'_T et V'_N .

Lemme 9.1 Soit $D: x \longrightarrow \ldots \longrightarrow \phi$, où $x \in V_N$, une dérivation de longueur $n \geq 1$ dans R. Si x n'est pas inaccessible, alors D est aussi une dérivation dans R'.

Démonstration. Il suffit de montrer que toutes les chaînes de D sont formées de symboles de V', et que toutes les règles utilisées appartiennent à R'. D'après le lemme 5.4 , on montre par récurrence sur n que les règles ne contiennent pas de symboles inaccessibles, donc sont aussi des règles de R'. Et comme tous les symboles contenus dans les chaînes de D figurent dans une de ces règles, ils appartiennent à V'.

CQFD.

Tout symbole de V'_N est partie gauche d'une règle de R', qui est aussi une règle de R ; c'est donc un symbole de V_N.

$$(a) \quad V'_N \subset V_N \cap V'$$

D'autre part, la définition de R' implique qu'il n'y a aucun symbole de $I[G]$ dans V'.

$$(b) \quad V' \subset V - I[G]$$

Soit x un symbole de $V_N - I[G]$.
x est partie gauche d'une règle T de R, dont la partie droite ne contient pas de symboles inaccessibles, d'après le lemme 5.4. T est aussi une règle de R' , donc $x \in V'_N$.

$$(c) \quad V_N - I[G] \subset V'_N$$

Soit enfin y un symbole quelconque de $V - I[G]$, différent de l'axiome S de G . Il existe dans R une dérivation
D: $S \longrightarrow \ldots \longrightarrow \Phi\, y\, \Psi$, de longueur ≥ 1.
S n'appartient pas à $I[G]$, donc, d'après le lemme 9.1 , D est une dérivation dans R' .
On en déduit que $y \in V'$.

$$(d) \quad V - I[G] \subset V'$$

De (a), (b), (c) et (d), on tire :

$$V' = V - I[G]$$

$$V'_N \subset V_N \cap V' = V_N \cap (V - I[G]) = V_N - I[G] \subset V'_N$$

d'où finalement

$$(9.2) \qquad V'_N = V_N \cap V' = V_N - I[G]$$

$$V'_T = V_T \cap V' = V_T - I[G]$$

De (9.2) , on déduit que l'axiome S de G appartient à V'_N.

Il est donc possible de le prendre comme axiome d'une nouvelle grammaire $G' = (R' , S)$, qu'on appelera aussi $\mu(G)$.

On montrera que $\mu(G)$ est une grammaire sans inaccessibles équivalente à G.

Proposition 9.3 La grammaire $G' = \mu(G)$ est équivalente à G.

$$L[\mu(G)] = L[G]$$

Démonstration. 1° Soit ϕ une chaîne de $L[G]$. Il existe une dérivation terminale D: $S \longrightarrow \ldots \longrightarrow \phi$, de longueur ≥ 1, dans R. D'après le lemme 9.1 , D est aussi une dérivation dans R' . Les symboles de ϕ appartiennent à $V' \cap V_T$, donc à V'_T , d'après (9.2) . D est donc une S-dérivation terminale dans R', et ϕ appartient au langage $L[G']$.
2° Comme $R' \subset R$, $V'_T \subset V_T$ et $V'_N \subset V_N$, toute S-dérivation terminale dans R' est une S-dérivation terminale dans R.

CQFD.

Proposition 9.4 La relation $\sigma_{R'}$ est la restriction aux éléments de V' de la relation σ_R.

Démonstration. 1° Si $x \sigma_R y$, et si $x \neq y$, il existe dans R une dérivation D: $x \longrightarrow \ldots \longrightarrow \Phi y \Psi$, de longueur ≥ 1. Si x et y appartiennent à V' , alors $x \in V - I[G]$, d'après (9.2) . D'après le lemme 9.1 , D est une dérivation dans R', donc $x \sigma_{R'} y$.
2° Comme $V' \subset V$ et $R' \subset R$, toute dérivation dans R' est une dérivation dans R, donc $x \sigma_{R'} y$ implique $x \sigma_R y$.

CQFD.

Corollaire 9.5 La grammaire $G' = \mu(G)$ est sans inaccessibles.

$$I[\mu(G)] = \emptyset$$

Démonstration. Soit x un symbole quelconque de V'. D'après (9.2), $x \in V - I[G]$, donc $S \sigma_R x$. D'où on déduit que $S \sigma_{R'} x$, en vertu de la proposition 9.4

CQFD.

Proposition 9.6 Si G est sans parasites, $\mu(G)$ est aussi sans parasites.

Démonstration. Soit x un symbole quelconque de V'_N. D'après (9.2), $x \in V_N - I[G]$, donc, si $P[G] = \emptyset$, il existe dans R une dérivation terminale D: $x \longrightarrow \ldots \longrightarrow \phi$, de longueur ≥ 1. D'après le lemme 9.1, D est une dérivation dans R'. Les symboles de ϕ appartiennent à $V' \cap V_T$, donc à V'_T, d'après (9.2). D est donc une x-dérivation terminale dans R'.

CQFD.

10. Construction d'une grammaire sans vides.

Soit G = (R, S) une grammaire sans parasites, définissant un langage qui ne se réduit pas à la chaîne vide.

$$P[G] = \emptyset$$
$$L[G] \neq \{\Lambda\}$$

Soit W[G] l'ensemble des symboles vides de V, $W_1[G]$ l'ensemble des symboles semi-vides, $W_2[G]$ l'ensemble des symboles essentiellement vides.

On définit une fonction f sur R ; à chaque règle $T \in R$,

$$T: x ::= \Sigma ,$$

on fait correspondre la règle

$$f(T) = T' : x ::= \Sigma' ,$$

où Σ' est la chaîne obtenue en supprimant dans Σ toutes les occurrences de symboles de $W_2[G]$.

On définit sur f(R) une fonction multivoque g ; à chaque règle U e f(R) ,

$$U : y ::= \Phi$$

où Φ contient n occurrences de symboles de $W_1[G]$, on fait correspondre l'ensemble

$$g(U) = \{U'_i : y ::= \Phi'_i \quad , \quad 1 \leq i \leq 2^n\} ,$$

où les Φ'_i sont : la chaîne Φ et toutes les chaînes obtenues en supprimant dans Φ un nombre quelconque d'occurrences de symboles de $W_1[G]$.

On désigne enfin par R' l'ensemble des règles de g(f(R)) qui ont une chaîne non vide en partie droite.

Exemple. Supposons que a, b, c e $W_1[G]$, d, e, f e $W_2[G]$. Considérons une partie A de R :

x ::= a g b h d
x ::= b c e f
y ::=
y ::= a b
z ::= d e
z ::= g h i f

On a pour f(A) :

x ::= a g b h
x ::= b c
y ::=
y ::= a b
z ::=
z ::= g h i

On a pour g(f(A)) :

x ::= a g b h
x ::= a g h
x ::= g b h

x ::= g h
x ::= b c
x ::= b
x ::= c
x ::=
y ::=
y ::= a b
y ::= a
y ::= b
z ::=
z ::= g h i

Enfin, $g(f(A)) \cap R'$ se réduit à :

x ::= a g b h
x ::= a g h
x ::= g b h
x ::= g h
x ::= b c
x ::= b
x ::= c
y ::= a b
y ::= a
y ::= b
z ::= g h i

Appelons V', V'_T, V'_N, respectivement, les vocabulaires $V[R']$, $V_T[R']$, $V_N[R']$, associés à R'.

Soit x un symbole de V'_N. x est partie gauche d'une règle T' de R', et T' appartient à g(f(T)), où T est une règle de R ayant x comme partie gauche.
Donc $x \in V_N$.

$$(a) \quad V'_N \subset V_N \cap V'$$

Par construction, aucune règle de R' n'a de symboles essentiellement vides en partie droite. Supposons qu'une règle T' de R' ait pour partie gauche un symbole x e $W_2[G]$. On doit avoir T' e g(f(T)) , où T est une règle de R ayant x comme partie gauche. Or, d'après le lemme 6.3 , tous les symboles de la partie droite de T sont essentiellement vides, donc on a :

$$f(T) : x ::= \Lambda$$
$$g(f(T)) = \{ x ::= \Lambda \}$$

et enfin $g(f(T)) \cap R' = \emptyset$

ce qui contredit l'existence de la règle T'. Autrement dit, aucune règle de R' n'a pour partie gauche un symbole essentiellement vide.

$$(b) \quad V' \subset V - W_2[G]$$

Soit y un symbole de V_T. y appartient à la partie droite d'une règle T de R ,

$$T : x ::= \Phi\ y\ \Psi$$

On a $f(T) : x ::= \Phi'\ y\ \Psi'$

et $f(T)$ e $g(f(T)) \cap R'$,

donc y e V'.

$$(c) \quad V_T \subset V'$$

Soit enfin z un symbole de $V_N - W_2[G]$.

D'après le lemme 6.3 , il existe dans R une règle T au moins,

$$T: z ::= \Phi\ u\ \Psi$$

où u n'appartient pas à $W_2[G]$.

On a $f(T) : z ::= \Phi'\ u\ \Psi'$

et $f(T)$ e $g(f(T)) \cap R'$,

donc z e V'_N .

$$(d) \quad V_N - W_2[G] \subset V'_N$$

De (a) , (b) , (c) et (d) on tire :

$$V'_N \subset V_N \cap V' \subset V_N \cap (V - W_2[G]) = V_N - W_2[G] \subset V'_N$$

d'où finalement :

(10.1)
$$V'_N = V_N \cap V' = V_N - W_2[G]$$
$$V'_T = V_T \cap V' = V'_T$$

Le langage $L[G]$ ne se réduisant pas à la seule chaîne vide, l'axiome S de G n'est pas essentiellement vide. D'après (10.1) , on voit que $S \in V'_N$. Il est donc possible de prendre S comme axiome d'une nouvelle grammaire $G' = (R', S)$ qu'on appellera aussi $\lambda[G]$.

On montrera que $\lambda[G]$ est une grammaire sans vides et sans parasites, équivalente à G, à la chaîne vide près.

Proposition 10.2 La grammaire $G' = \lambda[G]$ est sans vides.

$$W[G] = \emptyset$$

Démonstration. x étant un symbole quelconque de V'_N , il n'existe pas de x-dérivation de la chaîne vide , puisque aucune règle de R' n'a la chaîne vide en partie droite.

CQFD.

Considérons une dérivation terminale D dans R.

$$D: \Sigma_1 \longrightarrow \Sigma_2 \longrightarrow \ldots \longrightarrow \Sigma_n \longrightarrow \phi$$

Dans chaque chaîne Σ_i , en vertu du lemme 2.1 , on peut associer à chaque occurrence x de symbole une sous-chaîne ψ de ϕ , telle que $x \Longrightarrow \psi$; on notera x_ψ l'occurrence considérée.

Quelle que soit la dérivation terminale envisagée , il est clair qu'un symbole $x \in W_2[G]$ a toutes ses occurrences de la forme x_Λ ; et que toute occurrence y_Λ est une occurrence d'un symbole $y \in W[G]$.

On dira qu'une dérivation

$$\Sigma_o \longrightarrow \Sigma_1 \longrightarrow \ldots \longrightarrow \Sigma_n$$

est <u>de type vide</u> si on peut obtenir Σ_n en supprimant des occurrences de symboles dans Σ_1. On dira qu'une dérivation terminale est <u>standard</u> si toute chaîne contenant une occurrence x_Λ est intérieure à une dérivation de type vide.

<u>Lemme 10.3</u> Soit une dérivation terminale dans R ,

$$D: \quad \Sigma \longrightarrow \Sigma_1 \longrightarrow \ldots \longrightarrow \phi \ ,$$

où Σ ne contient pas d'occurrence du type x_Λ . Alors D est équivalente à une dérivation standard ; en d'autres termes , il existe dans R une dérivation standard

$$D': \quad \Sigma \longrightarrow \ldots \longrightarrow \phi \ .$$

<u>Démonstration</u>. Elle se fait par récurrence sur la longueur de D.

a) Si D est de longueur 0 , $\Sigma = \phi$, il n'y a aucune occurrence x_Λ donc D est standard.

b) Supposons la propriété vraie pour toutes les dérivations de longueur $\leq n$. Soit

$$D: \quad \Sigma \longrightarrow \Sigma_1 \longrightarrow \ldots \longrightarrow \phi$$

une dérivation de longueur $n + 1$. Si Σ_1 ne contient pas d'occurrence x_Λ , on peut appliquer l'hypothèse de récurrence à la dérivation

$$D_1 : \quad \Sigma_1 \longrightarrow \ldots \longrightarrow \phi .$$

Il existe donc une dérivation standard

$$D_1' : \Sigma_1 \longrightarrow \ldots \longrightarrow \phi ,$$

d'où une dérivation standard

$$D' : \Sigma \longrightarrow \Sigma_1 \longrightarrow \ldots \longrightarrow \phi .$$

Si Σ_1 contient une ou des occurrences $x_{i,\Lambda}$, on peut écrire

$$\Sigma_1 = \Phi_1 \, x_{1,\Lambda} \, \Phi_2 \ldots \Phi_k \, x_{k,\Lambda} \, \Phi_{k+1} ,$$

où les Φ_i sont des chaînes, non toutes vides, d'occurrences y_{j,ψ_j} avec $\psi_j \neq \Lambda$.

D'après le lemme 2.1, il existe une dérivation de longueur n

$$D_1 : \Sigma_1 = \Phi_1 \, x_{1,\Lambda} \, \Phi_2 \ldots \Phi_k \, x_{k,\Lambda} \, \Phi_{k+1} \longrightarrow$$

$$\longrightarrow \Phi_1 \, \Phi_2 \ldots \Phi_k \, x_{k,\Lambda} \, \Phi_{k+1} \longrightarrow \ldots$$

$$\longrightarrow \Phi_1 \, \Phi_2 \ldots \Phi_k \, \Phi_{k+1} \longrightarrow \ldots \longrightarrow \phi .$$

La dérivation

$$D_2 : \Phi_1 \, \Phi_2 \ldots \Phi_k \, \Phi_{k+1} \longrightarrow \ldots \longrightarrow \phi$$

est de longueur <n, et sa première chaîne ne contient pas d'occurrence y_Λ. D'après l'hypothèse de récurrence, il existe une dérivation standard

$$D_2' : \Phi_1 \, \Phi_2 \ldots \Phi_k \, \Phi_{k+1} \longrightarrow \ldots \longrightarrow \phi .$$

Considérons alors la dérivation

$$D' : \Sigma \longrightarrow \Sigma_1 = \Phi_1 \, x_{1,\Lambda} \, \Phi_2 \ldots \Phi_k \, x_{k,\Lambda} \, \Phi_{k+1} \longrightarrow$$

$$\longrightarrow \Phi_1 \, \Phi_2 \ldots \Phi_k \, x_{k,\Lambda} \, \Phi_{k+1} \longrightarrow \ldots$$

$$\longrightarrow \Phi_1 \, \Phi_2 \ldots \Phi_k \, \Phi_{k+1} \longrightarrow \ldots \longrightarrow \phi$$

ayant pour queue la dérivation D_2'.

Les chaînes Σ et $\Phi_1 \, \Phi_2 \ldots \Phi_k \, \Phi_{k+1}$ ne contiennent pas d'occurrence y_Λ. Les chaînes Σ_1, $\Phi_1 \, \Phi_2 \ldots \phi_k \, x_{k,\Lambda} \, \Phi_{k+1}$, ..., qui contiennent des occurrences $x_{i,\Lambda}$, sont intérieures à la dérivation de type vide

$$\Sigma \longrightarrow \Sigma_1 \longrightarrow \ldots \longrightarrow \Phi_1 \, \Phi_2 \ldots \Phi_k \, \Phi_{k+1} .$$

Enfin, toute chaîne, placée après $\Phi_1\ \Phi_2\ \ldots\ \Phi_k\ \Phi_{k+1}$, contenant une occurrence y_Λ , est intérieure à une dérivation de type vide, puisque D_2' est standard. On en déduit que la dérivation D' est standard.

CQFD.

Lemme 10.4 S'il existe dans R une x-dérivation d'une chaîne terminale ϕ non vide, alors il existe une x-dérivation de ϕ dans R'. Réciproquement, s'il existe dans R' une x-dérivation d'une chaîne terminale ϕ, alors il existe une x-dérivation de ϕ dans R.

Démonstration. 1° Considérons la dérivation

$$D: \quad x_\phi \longrightarrow \ldots \longrightarrow \phi$$

dans R. Comme $\phi \neq \Lambda$, on peut appliquer le lemme 10.3 . Il existe donc une dérivation standard dans R:

$$D_1 : \quad x_\phi \longrightarrow \ldots \longrightarrow \phi \ .$$

Considérons la suite de chaînes D' obtenue en supprimant dans D_1 les chaînes intérieures aux dérivations de type vide, et montrons que c'est une dérivation dans R' .

Il n'y a pas d'occurrence de symboles essentiellement vides dans D' . En effet, toute occurrence dans D_1 d'un symbole y de $W_2[G]$ est de la forme y_Λ ; comme D_1 est standard, une telle occurrence doit être dans une chaîne intérieure à une dérivation de type vide, et ne figure donc plus dans D' . Ainsi tous les symboles des chaînes de D' appartiennent à $V\text{-}W_2[G]$, donc à V', d'après (10.1). Il reste à montrer que, pour toute paire de chaînes Φ et Σ adjacentes dans D' , on a $\Phi \xrightarrow[R']{} \Sigma$.

a) Si Φ et Σ sont déjà adjacentes dans D_1 , on a $\Phi \xrightarrow[R]{} \Sigma$.

Soit u_ψ l'occurrence associée à Φ , et

$$T: \quad u ::= \theta \ ,$$

la règle de R concernée. θ n'est pas vide, puisque $\psi \neq \Lambda$; et θ ne contient pas de symboles de $W_2[G]$, puisque c'est une partie de Σ. Donc

$$T = f(T) \in g(f(T)) \cap R' ,$$

ce qui implique que $\Phi \xrightarrow[R']{} \Sigma$.

b) Si Φ et Σ ne sont pas adjacents dans D_1 , alors il y a dans D_1 une dérivation de type vide

$$\Phi \longrightarrow \Phi_1 \longrightarrow \ldots \longrightarrow \Sigma ,$$

avec $\Phi = \Sigma' u_\psi \Sigma''$

$\Phi_1 = \Sigma' \Sigma_1 y_{1,\Lambda} \Sigma_2 \ldots \Sigma_k y_{k,\Lambda} \Sigma_{k+1} \Sigma''$,

$\Sigma = \Sigma' \Sigma_1 \Sigma_2 \ldots \Sigma_k \Sigma_{k+1} \Sigma''$,

et la règle $T : u ::= \Sigma_1 y_1 \Sigma_2 \ldots \Sigma_k y_k \Sigma_{k+1}$

dans R.

La chaîne Σ , appartenant à la dérivation D' , ne contient pas de symboles de $W_2[G]$; donc les Σ_i ne contiennent pas de tels symboles. Les symboles y_i appartiennent à $W[G]$. Il existe donc dans $g(f(T))$ une règle

$$T' : u ::= \Sigma_1 \Sigma_2 \ldots \Sigma_k \Sigma_{k+1} .$$

Comme $\psi \neq \Lambda$, les Σ_i ne sont pas tous vides, donc la règle T' appartient à R' . Ce qui permet d'affirmer que $\Phi \xrightarrow[R']{} \Sigma$.

2° Soit D' une x-dérivation de ϕ dans R' . Comme $V' \subset V$, tous les symboles des chaînes de D' appartiennent à V. Il suffit donc de montrer que, pour toute paire de chaînes Φ et Σ adjacentes dans D' , on a $\Phi \xRightarrow[R]{} \Sigma$.

Comme $\Phi \xrightarrow[R']{} \Sigma$, il existe dans R' une règle

$$T' : u ::= \theta ,$$

avec $\Phi = \Phi_1 u \Phi_2$

et $\Sigma = \Phi_1 \theta \Phi_2$.

Or, T' ∈ g(f(T)) , où T est une règle

$$T: \quad u ::= \theta_o$$

de R, θ pouvant être obtenu en supprimant dans θ_o un ensemble (éventuellement vide) d'occurrences de symboles $y_i \in W[G]$.

$$\theta_o = \theta_1 \, y_1 \, \theta_2 \ldots \theta_k \, y_k \, \theta_{k+1}$$

$$\theta = \theta_1 \, \theta_2 \ldots \theta_k \, \theta_{k+1} .$$

Il existe donc, dans R, une dérivation

$$\Phi = \Phi_1 \, u \, \Phi_2 \longrightarrow \Phi_1 \, \theta_o \, \Phi_2 = \Phi_1 \, \theta_1 \, y_1 \, \theta_2 \ldots \theta_k \, y_k \, \theta_{k+1} \, \Phi_2 \longrightarrow \ldots$$

$$\longrightarrow \Phi_1 \, \theta_1 \, \theta_2 \ldots \theta_k \, y_k \, \theta_{k+1} \, \Phi_2 \longrightarrow \ldots$$

$$\longrightarrow \Phi_1 \, \theta_1 \, \theta_2 \ldots \theta_k \, \theta_{k+1} \, \Phi_2 = \Phi_1 \, \theta \, \Phi_2 = \Sigma$$

Ce qui démontre que $\Phi \underset{R}{\Longrightarrow} \Sigma$.

CQFD.

Proposition 10.5 $\quad L[\lambda(G)] = L[G] - \{\Lambda\}$.

Démonstration. 1° Si $\phi \in L[G]$, et $\phi \neq \Lambda$, alors $S \underset{R}{\Longrightarrow} \phi$, d'où $S \underset{R'}{\Longrightarrow} \phi$ d'après le lemme 10.4, ce qui signifie que $\phi \in L[G']$.

2° Si $\phi \in L[G']$, alors $S \underset{R'}{\Longrightarrow} \phi$, ce qui implique $\phi \neq \Lambda$, d'après la proposition 10.2 , et $S \underset{R}{\Longrightarrow} \phi$, d'après le lemme 10,4 . Donc $\phi \in L[G] - \{\Lambda\}$.

CQFD.

Proposition 10.6 La grammaire $G' = \lambda(G)$ est sans parasites.

$$P[\Lambda(G)] = \emptyset$$

Démonstration. Soit x un symbole quelconque de V'_N . D'après (10.1) , $x \in V_N - W_2[G]$, donc il existe dans R une x-dérivation d'une chaîne terminale $\phi \neq \Lambda$. D'après le lemme 10.4 , il existe une x-dérivation de ϕ dans R' , et c'est une dérivation terminale puisque $V'_T = V_T$, d'après (10.1) .

CQFD.

Proposition 10.7 La relation $\sigma_{R'}$ est la restriction aux éléments de V' de la relation σ_R .

Démonstration. 1° Si $x\ \sigma_R\ y$, et si $x \neq y$, il existe dans R, d'après la proposition 7.1 , des règles T_o , ... , T_n , telles que

$$T_i \ : x_i ::= \Phi_i\ x_{i+1}\ \Psi_i$$

avec $x_o = x$ et $x_{n+1} = y$. Si x et y appartiennent à V' , alors $y \in V - W_2[G]$, d'après (10.1), donc tous les x_i appartiennent à $V - W_2[G]$, d'après le lemme 6.3. Pour tout i , on a

$$f(T_i) \ : \ x_i ::= \Phi'_i\ x_{i+1}\ \Psi'_i$$

et $$f(T_i) \in g(f(T_i)) \cap R' .$$

Ce qui implique que $x\ \sigma_{R'}\ y$, d'après la proposition 7.1 .

2° Si $x\ \sigma_{R'}\ y$, et si $x \neq y$, il existe dans R' , d'après la proposition 7.1 , des règles

$$T'_i : x_i ::= \Phi'_i\ x_{i+1}\ \Psi'_i$$

avec $x_o = x$ et $x_{n+1} = y$. Pour tout i , il existe dans R une règle

$$T_i : x_i ::= \Phi_i\ x_{i+1}\ \Psi_i$$

telle que $T'_i \in g(f(T_i))$. On en déduit que $x\ \sigma_R\ y$, d'après la proposition 7.1 .

CQFD.

Corollaire 10.8 Si G est sans inaccessibles, $\lambda(G)$ est aussi sans inaccessibles.

Démonstration. Soit x un symbole quelconque de V' . $x \in V$, et , si $I[G] = \emptyset$, on a $S\ \sigma_R\ x$. Ce qui implique que $S\ \sigma_{R'}\ x$, d'après la proposition 10.7 .

CQFD.

11. Problème général.

Soit $G = (R,S)$ une grammaire quelconque. On désire connaître :
une grammaire $G_o = (R_o , S)$, équivalente à G , sans parasites , sans inaccessibles et sans vides ; et les relations σ_{Ro} , γ_{Ro} , δ_{Ro} et ε_{Go} , valables entre les symboles de $V_o = V[R_o]$.

Ce problème peut être résolu par l'algorithme suivant :

a) Partant de l'ensemble $V_T[R]$, on construit successivement , en parcourant les règles de R , les ensembles $V_{T,o} = V_T$, $V_{T,1}$, ..., définis en (3.5) . Dès qu'on obtient $V_{T,N+1} = V_{T,N}$, on a $V_{T,N} = \tilde{V}_T$, d'après la proposition 3.6 .

On connaît alors $P[G] = V - \tilde{V}_T$, d'après (5.1) .

b) Si $S \in P[G]$, on sait alors que $L[G] = \emptyset$, et l'étude de la grammaire G s'arrête ici .

Sinon, en supprimant dans R les règles qui contiennent des symboles de $P[G]$, on obtient la grammaire $G_1 = \Pi(G) = (R_1 , S)$. D'après les propositions 8.3 et 8.4 , G_1 est une grammaire sans parasites équivalente à G .

c) En parcourant les règles de R_1 , on établit la relation σ_o . On calcule ensuite sa fermeture transitive , par l'algorithme de Warshall ([9]) ; ce qui donne la relation σ_{R_1} , d'après la proposition 7.1 .

d) La relation σ_{R_1} permet de déterminer l'ensemble $I[G_1] = V[R_1] - \{ y \in V[R_1] \mid S \, \sigma_{R_1} \, y \}$, d'après (5.3) .

En supprimant dans R_1 les règles qui contiennent des symboles de $I[G_1]$, on obtient la grammaire $G_2 = \mu(G_1) = (R_2, S)$. D'après les propositions 9.3 et 9.6 , et le

corollaire 9.5 , G_2 est une grammaire sans parasites et sans inaccessibles , équivalente à G .

D'après la proposition 9.4 , la relation σ_{R_1} , restreinte aux symboles de $V[R_2]$, donne la relation σ_{R_2} .

e) Partant de l'ensemble vide $\emptyset$, on construit successivement , en parcourant les règles de R_2 , les ensembles $A_o = \emptyset$, A_1 ,...., définis en (3.5) . Dès que l'on obtient $A_{N+1} = A_N$, on a $A_N = \tilde{\emptyset}$, d'après la proposition 3.6 .

On connaît alors $W[G_2] = \tilde{\emptyset}$, d'après (6.1) .

D'après le corollaire 6.5 , la relation σ_{R_2} permet de déterminer les ensembles

$$W_1[G_2] = \{ z \in W[G_2] \mid \exists y \in V_T[G_2] , \ z \ \sigma_{R_2} \ y \}$$

et $W_2[G_2] = W[G_2] - W_1[G_2]$.

f) Si $S \in W_2[G_2]$, on sait alors que $L[G_2] = \{\Lambda\}$.

La grammaire G est équivalente à la grammaire G_o ayant pour seule règle : $S ::= \Lambda$, et son étude s'arrête ici.

Sinon , on construit la grammaire $G_3 = \lambda(G_2) = (R_3, S)$. D'après les propositions 10.2 et 10.6 , et le corollaire 10.8, G_3 est une grammaire sans parasites , sans inaccessibles et sans vides .

g) D'après la proposition 10.7 , la relation σ_{R_2} , restreinte aux éléments de $V[R_3]$, donne la relation σ_{R_3} .

En parcourant les règles de R_3 , on établit les relations γ_o et δ_o. On calcule ensuite leurs fermetures transitives , par l'algorithme de Warshall ; ce qui donne les relations γ_{R_3} et δ_{R_3} , d'après les propositions 7.2 et 7.3 .

h) Pour calculer la relation ε_{G_3} , on cherche dans R_3 tous les couples de symboles qui sont adjacents dans la partie droite d'une règle ; pour chacun de ces couples u, v, et pour tout x tel que $u \delta_{R_3} x$, et tout y tel que $v \gamma_{R_3} y$, on pose $x \varepsilon_{G_3} y$, en vertu de la proposition 7.4 .

i) D'après la proposition 10.5 , $L[G_3] = L[G_2] - \{\Lambda\}$.

Si S n'appartient pas à $W_1[G_2]$, on sait alors que la chaîne vide Λ n'appartient pas à $L[G_2]$, donc $L[G_3] = L[G_2]$. La grammaire G_3 est équivalente à G , c'est la grammaire G_o cherchée.

Si $S \in W_1[G_2]$, alors $\Lambda \in L[G_2]$. Il est impossible dans ce cas de trouver une grammaire sans vides équivalente à G . La grammaire G_o , obtenue en ajoutant à G_3 la règle : $S ::= \Lambda$, est en revanche sans parasites et sans inaccessibles , et possède une seule règle ayant la chaîne vide en partie droite. En outre , l'adjonction de cette règle ne modifie ni les vocabulaires associés, ni les relations σ, γ, δ, ε; σ_{R_o}, γ_{R_o}, δ_{R_o} et ε_{G_o} , sont donc données par σ_{R_3}, γ_{R_3} , δ_{R_3} et ε_{G_3} , respectivement.

II. REALISATION PRATIQUE

1. Aperçu global

Etant données :

a) Une grammaire écrite sous une forme voisine de celle de Backus ;

b) Une chaîne de caractères quelconque ;

les programmes ALGOL décrits dans ce chapitre effectuent les opérations suivantes :

1) Lecture et enregistrement des règles de la grammaire ; constitution du dictionnaire des symboles terminaux et non-terminaux.

2) Construction d'une grammaire équivalente, sans parasites, sans inaccessibles et sans vides.

3) Calcul des relations σ, γ, δ et ε .

4) Edition de la chaîne de caractères ; c'est-à-dire : interprétation de la chaîne de caractères comme une suite de symboles appartenant au vocabulaire terminal associé à la grammaire.

5) Enregistrement :

des règles de la grammaire;
du dictionnaire des symboles;
des tableaux booléens représentant les relations;
de la chaîne éditée;
sur un support extérieur (unité de disques), d'où ils pourront être repris par un programme d'analyse syntaxique.

L'ensemble E des symboles, au sens du chapitre précédent, est l'ensemble des suites finies de caractères différents du blanc.

Sur cartes perforées, les règles de la grammaire donnée se suivent, sans séparateurs autres que des blancs. Dans chaque règle, les symboles (et les métasymboles ::= et | de la forme de Backus) sont séparés par des blancs. Aucune restriction n'est faite quand à la forme des symboles (terminaux ou non-terminaux).

2. Représentation en machine.

Dans les programmes, la grammaire et le vocabulaire sont représentés avec les conventions adoptées par A. COLMERAUER ([13],[14]).

Notons d'abord que la grammaire est une suite de règles C.F. : un symbole en partie gauche, une chaîne en partie droite. Ainsi, une règle avec plusieurs chaînes en partie droite, séparée par le | de Backus, est mise en mémoire sous la forme de plusieurs règles C.F. ayant la même partie gauche.

La grammaire est représentée par un tableau entier GRAMMAIRE à un indice, dans lequel les règles se suivent sans séparateur. Chaque symbole y est représenté par un code numérique, et le métasymbole ::= remplacé par un entier indiquant la longueur de la chaîne (le nombre de symboles) en partie droite.

Ainsi, tout algorithme parcourant les règles de la grammaire devra utiliser la boucle : pour I := 2,
I + GRAMMAIRE[I] + 2 tant que GRAMMAIRE [I - 1]≠ 0 faire ...

Ceci afin d'avoir, pour chaque règle, la partie gauche en GRAMMAIRE [I-1] , et la partie droite de GRAMMAIRE [I+1] à GRAMMAIRE [I+GRAMMAIRE[I]].

Le dictionnaire des symboles est représenté par deux tableaux entiers à un indice, DICO et CHAINE. A chaque

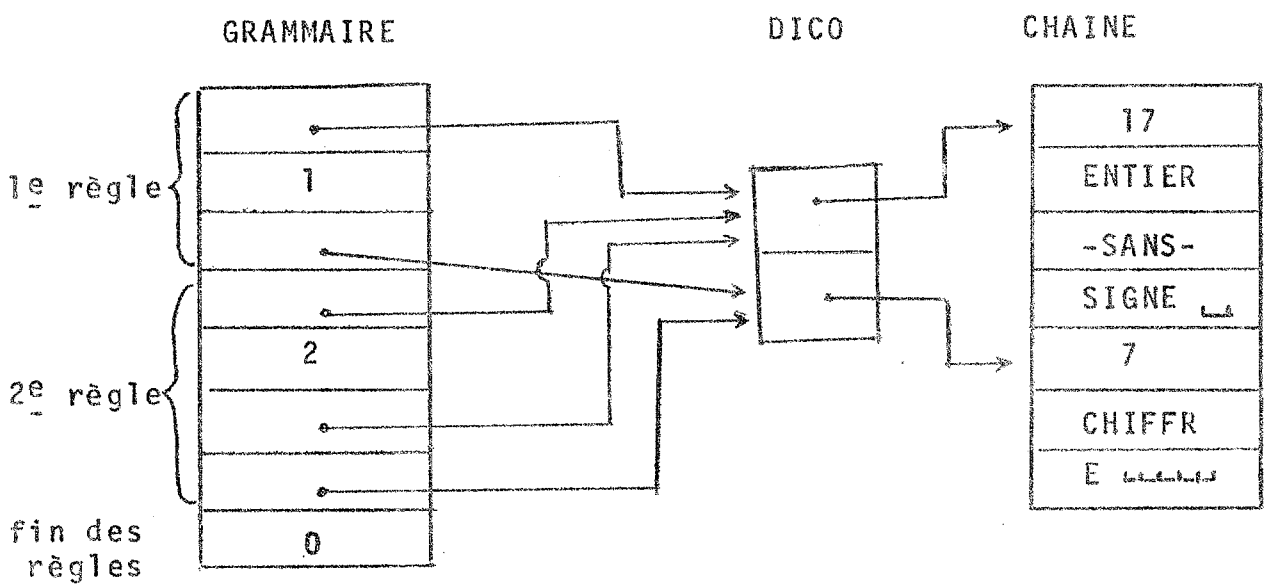
symbole correspond un élément du tableau DICO, et un nombre quelconque d'éléments du tableau CHAINE.

Le code numérique J d'un symbole est en fait un pointeur (indice) vers l'élément correspondant de DICO. DICO[J] est lui-même un pointeur vers le premier des éléments correspondants de CHAINE. Celui-ci, CHAINE[DICO[J]] , est un entier indiquant le nombre de caractères du symbole. Les éléments suivants contiennent les codes-machine de caractères, rangés de manière compacte (pour IBM 7044 : 6 caractères BCD par mot).

Exemple. La grammaire suivante, réduite à une seule règle sous forme de Backus (donnant deux règles C.F.) :

```
ENTIER-SANS-SIGNE  ::=  CHIFFRE
                     |  ENTIER-SANS-SIGNE   CHIFFRE
```

serait représentée par les tableaux suivants (les pointeurs étant indiqués ici par des flèches) :



Remarques. 1° Seul le tableau GRAMMAIRE intervient dans les divers algorithmes ; DICO et CHAINE ne servent qu'aux entrées/sorties.

2° L'usage d'une table d'adresses (le tableau DICO) permet seul d'utiliser les N premiers entiers comme codes de symboles; il permet aussi de redistribuer facilement ces codes.

3° Dans certains programmes, DICO et CHAINE sont rangés bout à bout dans un même tableau (appelé DICO).

3. Procédures d'entrée/sortie.

Le langage ALGOL n'étant a priori pas conçu pour les manipulations de symboles, les opérations d'entrée/sortie nécessitent des procédures spéciales. En dépit de leur grande généralité, les procédures INSYMBOL et OUTSYMBOL, recommandées par l'IFIP ([6]), et implémentées sur IBM 7044 par J. COHEN et NGUYEN-HUU-DUNG ([16],[17]), ont paru inadéquates, car elles affectent un code numérique à chaque caractère, alors qu'ici l'unité d'information, appelé symbole, est une suite finie de caractères. La préférence a donc été donnée à des procédures manipulant : d'une part, des mots contenant plusieurs codes-machine de caractères; d'autre part, des codes numériques désignant des symboles; et transformant directement les uns en les autres, au moyen du dictionnaire décrit au paragraphe précédent.

Les procédures d'écriture utilisées ici, de conception assez générale, sont dues à A COLMERAUER. Les procédures de lecture ont été étudiées spécialement pour les opérations où elles interviennent: la lecture des règles de la grammaire et l'édition de la chaîne de caractères. Elles comprennent des procédures écrites en code (LIREMOT, AVANCER, RECULER), qui

lisent des caractères et les transfèrent dans un tableau servant de buffer; et des procédures entières, écrites en ALGOL (LIRE ET CODER, EDITEUR), donnant pour valeur le code numérique du symbole lu, obtenu par comparaison du buffer avec le dictionnai

a) Lecture de la grammaire.

Les symboles à lire peuvent être n'importe quels éléments de l'ensemble E, donc n'importe quelle suite finie de caractères. D'où la nécessité, sur cartes perforées, de considérer les blancs comme séparateurs. La procédure LIRE MOT a donc pour tâche de lire une suite de caractère comprise entre deux blancs séparateurs. Un appel à LIRE MOT (BUFFER,NMOTS, NCARAC) lit des caractères sur cartes; ignore les blancs du début; transfère les caractères non blancs, sous forme compacte, dans le tableau BUFFER; s'arrête à la rencontre du prochain blanc; affecte à la variable NCARAC le nombre de caractères transférés, à NMOTS le nombre de mots utilisés dans BUFFER.

La procédure entière LIRE ET CODER, listée ci-après, a pour tâche, non seulement de déterminer le code d'un symbole lu, mais de mettre à jour le dictionnaire des symboles de la grammaire. L'instruction N:= LIRE ET CODER appelle d'abord LIRE MOT, cherche le symbole lu dans le dictionnaire (tableaux DICO et CHAIN l'y ajoute éventuellement, et affecte à N le code du symbole.

b) Editeur.

Au moment d'éditer une chaîne de caractères, le vocabulaire associé à la grammaire est connu, le dictionnaire ne doit pas être mis à jour, et les symboles à lire doivent appartenir au vocabulaire terminal. Ce qui permet une plus grande souplesse dans la perforation de la chaîne d'entrée: les blancs y sont en principe non significatifs, et c'est par essais successifs

que les procédures cherchent dans la chaîne d'entrée un symbole appartenant au vocabulaire terminal.

La procédure AVANCER est une généralisation de LIRE MOT . Au moment de l'appel, le tableau CHAINE peut déjà contenir NCARAC caractères , occupant NMOT mots du tableau . Un appel à AVANCER(CHAINE, NMOT, NCARAC, LMAX, MAXATTEINT) lit des caractères sur cartes (ou dans un buffer d'entrée) en ignorant les blancs du début; les transfère à la suite dans le tableau CHAINE, en mettant à jour les valeurs de NMOT et NCARAC; s'arrête: soit à la rencontre du prochain blanc, en affectant la valeur faux à la variable booléenne MAXATTEINT; soit lorsque NCARAC atteint la valeur LMAX, en affectant à MAXATTEINT la valeur vrai.

Cette procédure est complétée par la procédure RECULER, qui supprime des caractères dans le tableau CHAINE et les replace dans le buffer d'entrée, où ils seront repris lors du prochain appel à AVANCER. Un appel à RECULER(CHAINE, NMOT, NCARAC, N, IMPOSSIBLE) transfère les N derniers caractères du tableau CHAINE vers le buffer d'entrée, et met à jour les valeurs de NMOT et NCARAC. La variable booléenne IMPOSSIBLE prend la valeur vrai si le tableau CHAINE est vide, ou si le buffer d'entrée est plein, rendant le recul impossible.

La procédure entière EDITEUR, listée ci-après, cherche dans la chaîne d'entrée une suite de caractères figurant dans le dictionnaire des symboles terminaux (tableau DICO). Elle utilise les procédures AVANCER et RECULER, selon une politique qui sera discutée au paragraphe 6. L'instruction N:=EDITEUR affecte à N le code du symbole ainsi déterminé.

. Lecture des règles.

Les règles à lire sont formées de symboles, de deux méta-symboles correspondant aux ::= et | de Backus, et d'un délimiteur, servant notamment à indiquer la fin des données;

```
1635        'BOOLEEN  ' 'PROCEDURE' EN TABLE ::
1639        'DEBUT' 'POUR' CODE:= 0 'PAS' 1 'JUSQUA' LONGDICO 'FAIRE'
1649           'DEBUT' L:=ABS(DICO.(CODE).) ::
1660           'SI' CHAINE.(L).=NCARAC 'ALORS'
1668              'DEBUT' 'POUR' J:=1 'PAS' 1 'JUSQUA' NMOTS 'FAIRE'
1678              'SI' CHAINE.(L+J). 'NONEG' BUFFER.(J). 'ALORS' 'ALLERA' A ::
1694              ENTABLE:= 'VRAI' ::
1698              'ALLERA' B ::
1701           A: 'FIN'
1704        'FIN' ::
1706        ENTABLE:= 'FAUX' ::
1710     B: 'FIN' ENTABLE ::
1714     'ENTIER ''PROCEDURE' LIRE ET CODER ::
1716        'DEBUT' LIREMOT (BUFFER, NMOTS, NCARAC) ::
1728        'SI' ENTABLE        'ALORS' LIREETCODER:=CODE 'SINON'
1735        'DEBUT' CHAINE.(LONGCHAINE).:=NCARAC ::
1743        'POUR' J:=1 'PAS' 1 'JUSQUA' NMOTS 'FAIRE'
1752        CHAINE.(LONGCHAINE+J).:=BUFFER.(J). ::
1764           'SI' NCARAC 'SUPER' NCARMAX 'ALORS' NCARMAX := NCARAC ::
1773           DICO.(LONGDICO+2).:= LONGCHAINE :=LONGCHAINE + NMOTS + 1 ::
1788           LIRE ET CODER:=LONGDICO := LONGDICO +1
1794           'FIN'
1796        'FIN' LIRE ET CODER ::
```

Procédure LIRE ET CODER

```
2612        'ENTIER''PROCEDURE' EDITEUR ::
2616        'DEBUT' 'ENTIER' CODE, NMOT, NCARAC, MIN, MAX ::
2628           'BOOLEEN' IMPOS, MAXATTEINT ::
2633           'ENTIER''TABLEAU' CHAINE.(1 : 50). ::
2642           'BOOLEEN''PROCEDURE' ENTABLE ::
2646           'DEBUT' 'ENTIER' J, L ::
2652              CODE := TABLE.(NCARAC). ::
2659              'POUR' CODE := CODE+1 'TANT QUE' DICO.(DICO.(CODE).).=NCARAC
2674              'FAIRE' 'DEBUT' L := DICO.(CODE). ::
2684                          'POUR' J := 1 'PAS' 1 'JUSQUA' NMOT 'FAIRE'
2693                    'SI' DICO.(L+J). 'NONEG' CHAINE.(J). 'ALORS'
2706                    'ALLER A' A ::
2709                    EN TABLE := 'VRAI' ::
2713                    'ALLER A' B ::
2716              A : 'FIN' ::
2720               EN TABLE := 'FAUX' ::
2724        B : 'FIN' ::
2728           NCARAC := 0 ::
2732        A : MIN := NCARAC + 1 ::
2740           AVANCER (CHAINE, NMOT, NCARAC, LMAX, MAXATTEINT) ::
2753           MAX := NCARAC ::
2757        Q : 'SI' ENTABLE 'ALORS'
2762           'DEBUT' EDITEUR := CODE ::
2767              'ALLER A' F
2768           'FIN' ::
2771           'SI' NCARAC 'SUPER' MIN 'ALORS'
2776           'DEBUT' RECULER (CHAINE, NMOT, NCARAC, 1, IMPOS) ::
2790              'SI' 'NON' IMPOS 'ALORS' 'ALLER A' Q
2795           'FIN' 'SINON'
2798           'SI' 'NON' MAX ATTEINT 'ALORS'
2802           'DEBUT' AVANCER (CHAINE, NMOT, NCARAC, MAX, IMPOS) ::
2816              'ALLER A' A
2817           'FIN' ::
2820           EDITEUR := 0 ::
2824     F : 'FIN' ::
```

Procédure EDITEUR

symboles, métasymboles et délimiteur sont représentés sur cartes par des suites quelconques de caractères.

Les données doivent contenir, dans l'ordre :

- le délimiteur (pour définition) ;
- le métasymbole ::= (pour définition) ;
- le métasymbole | (pour définition);
- la liste des opérateurs non-terminaux (cette liste peut être vide; voir plus loin) ;
- le délimiteur (peut être omis si la liste des opérateurs est vide) ;
- la liste des terminaux à code imposé (cette liste peut-être vide; voir plus loin) ;
- la liste des règles de la grammaire ;
- le délimiteur .

Dans une première partie, le programme construit la représentation symbolique de la grammaire. Il lit des symboles par la procédure LIRE ET CODER , considérant comme symboles du vocabulaire tous ceux qui diffèrent des métasymboles et du délimiteur.

Chaque symbole du vocabulaire est alors placé dans le tableau GRAMMAIRE . Chaque métasymbole indique une nouvelle règle, et provoque la mise à jour, dans le tableau GRAMMAIRE , des compteurs indiquant la longueur des parties droites des règles ; de plus, le symbole en partie gauche de la règle est noté comme non-terminal (provisoirement, par un signe - dans le tableau DICO) .

Dans une deuxième partie, le programme procède à une redistribution des codes des symboles, selon les principes suivants :

a) Les symboles terminaux sont séparés des non-terminaux. Si la liste des "opérateurs" est vide, les codes allant de 1 à NBTERMINAUX désignent les symboles terminaux, les codes de NBTERMINAUX + 1 à NBSYMBOLES désignent les non-terminaux.

L'axiome a le code NBTERMINAUX + 1 .

b) Sinon, les codes de 1 à NBTERMINAUX désignent les terminaux, les codes de NBTERMINAUX + 1 à NBOPERATEURS désignent les opérateurs non-terminaux, les codes de NBOPERATEURS + 1 à NBSYMBOLES désignent les autres symboles non-terminaux.

c) Si la liste des "codes imposés" est vide, les symboles terminaux sont classés par longueur croissante. Ceci facilite les recherches dans le dictionnaire, qui se font alors au moyen du tableau TABLE (notamment dans le programme éditeur). TABLE[N] est le code précédant celui du premier symbole de longueur N .

d) Sinon, les terminaux à "code imposé" reçoivent pour code les premiers entiers, affectés selon leur ordre d'apparition dans la liste.

Remarques :

1° Les possibilités de déterminer à l'avance le code de certains symboles terminaux (liste de codes imposés), et de distinguer un certain sous-ensemble parmi les symboles non-terminaux (liste d'opérateurs), ont été utilisées, notamment, dans les travaux décrits en [13] et [14].

2° De toute manière, les terminaux et les non-terminaux sont déterminés par les positions qu'ils occupent dans les règles. La présence de terminaux dans la liste des opérateurs, ou de non-terminaux dans la liste des codes imposés, donnerait des résultats indéterminés.

. Transformation de la grammaire et calcul des relations.

L'algorithme décrit au chapitre I, paragraphe 11, pour résoudre ce problème, contient essentiellement les constituants élémentaires suivants : calcul de relations, construction d'un ensemble Ã, transformation de la grammaire.

```
      'DEBUT' 'ENTIER ' 'TABLEAU ' DICO.(0 :1000).,
.0    CHAINE.(1:5000).,GRAMMAIRE.(1:7000). ::
24    'ENTIER ' LONGDICO, LONGGRAMMAIRE, LONGCHAINE,  NBOPERATEURS, NCARMAX,
35    LMAX, I,J,K,L ::
45    'BOOLEEN' CODE IMPOSE ::
48    'PROCEDURE' LIREMOT (BUFFER, NMOTS, NCARAC) ::
58    'ENTIER''TABLEAU' BUFFER :: 'ENTIER ' NMOTS, NCARAC ::                'CODE'
614      CODERCHAINE:
616      'DEBUT' 'ENTIER' 'TABLEAU' BUFFER.(1: 16). ::
626         'ENTIER ' PARTIE GAUCHE,NMOTS,NCARAC,CODE ::
798         LONGDICO := -1 ::
803         NBOPERATEURS := NCARMAX := 0 ::
809         DICO.(0). := LONGCHAINE := 1 ::
818         I := LONGGRAMMAIRE := 2 ::
824         'POUR' K := 1,2,3, K 'TANT QUE' PARTIE GAUCHE 'NONEG' 1 'FAIRE'
839         'DEBUT' CODE IMPOSE := PARTIE GAUCHE 'SUPER' 2 'ET' GRAMMAIRE.(I).
850            'SUPER' 2 ::
853                 GRAMMAIRE.(I). := PARTIE GAUCHE ::
860            PARTIE GAUCHE := LIRE ET CODER ::
864            'SI' PARTIE GAUCHE = 0 'ALORS' NBOPERATEURS := LONGDICO
871         'FIN' ::
874         'ALLER A' DEGRA ::
877         MOT SUIVANT: GRAMMAIRE.(LONGGRAMMAIRE).:=LIREETCODER ::
886         'SI' GRAMMAIRE.(LONGGRAMMAIRE).= 1 'ALORS'
894   DEGRA:'DEBUT' PARTIE GAUCHE:=GRAMMAIRE.(LONGGRAMMAIRE-1). ::
906         DICO.(PARTIE GAUCHE).:=-ABS(DICO.(PARTIEGAUCHE).)   ::
920         GRAMMAIRE.(I).:=GRAMMAIRE.(I).-1 ::
932         I:=LONGGRAMMAIRE ::
936         GRAMMAIRE.(I).:=0 ::
943         LONGGRAMMAIRE:=I+1 ::
949         'ALLERA' MOT SUIVANT
950         'FIN' ::
953         'SI' GRAMMAIRE.(LONGGRAMMAIRE).= 2 'ALORS'
961         'DEBUT' GRAMMAIRE.(LONGGRAMMAIRE).:=PARTIE GAUCHE ::
969         I:=LONGGRAMMAIRE+1 ::
975         GRAMMAIRE.(I).:=0 ::
982         LONGGRAMMAIRE:=I+1 ::
988         'ALLERA' MOT SUIVANT
989         'FIN' ::
992         'SI' GRAMMAIRE.(LONGGRAMMAIRE). 'NONEG' 0  'ALORS'
000          'DEBUT'
001         GRAMMAIRE.(I).:=GRAMMAIRE.(I).+1 ::
013         LONGGRAMMAIRE:=LONGGRAMMAIRE+1 ::
019         'ALLERA' MOT SUIVANT
020      'FIN' ::
023      LMAX := 0 ::
027      'POUR' J := 3 'PAS' 1 'JUSQUA' LONGDICO 'FAIRE'
036      'DEBUT' L := DICO.(J). ::
044         'SI' L 'SUPER' 0 'ALORS'
049         'DEBUT' K := CHAINE.(L). ::
057            'SI' K 'SUPER' LMAX 'ALORS' LMAX := K ::
066            DICO.(J). := -L ::
074            CHAINE.(L). := -K
080         'FIN'
082      'FIN'
083      'FIN' CODERCHAINE ::
```

LECTURE DES RÈGLES

```
2085      CHANGER CODE  :
2087      'DEBUT' 'ENTIER' NBTERMINAUX, NBSYMBOLES, FLECHE, OU, L1, L2 ::
2101         'ENTIER''TABLEAU' DICO2.(1 : LONGDICO)., TABLE.(1 : LMAX). ::
2228      I := 0 ::
2232      'SI' CODE IMPOSE 'ALORS'
2235      'DEBUT' 'POUR' J := 3 'PAS' 1 'JUSQUA' LONGDICO 'FAIRE'
2245         'DEBUT' L := -DICO.(J). ::
2254         'SI' L 'SUPER' 0 'ALORS'
2259         'DEBUT' 'SI' CHAINE.(L). 'INFER' 0 'ALORS'
2268            'DEBUT' I := I + 1 ::
2275               DICO2.(I). := L ::
2282               DICO.(J). := I ::
2289               CHAINE.(L). := -CHAINE.(L).
2299            'FIN'
2300         'FIN'
2301         'FIN'
2302      'FIN' 'SINON'
2304      'DEBUT'
2305      'POUR' K := 1 'PAS' 1 'JUSQUA' LMAX 'FAIRE'
2314      'DEBUT' TABLE.(K). := I ::
2322         'POUR' J := 3 'PAS' 1 'JUSQUA' LONGDICO 'FAIRE'
2331         'DEBUT' L :=-DICO.(J). ::
2340            'SI' L 'SUPER' 0 'ALORS'
2345            'DEBUT'
2346            'SI' CHAINE.(L). = -K 'ALORS'
2355            'DEBUT' I := I+1 ::
2362               DICO2.(I). := L ::
2369               DICO.(J). := I ::
2376               CHAINE.(L). := K
2381            'FIN'
2383            'FIN'
2384         'FIN'
2385      'FIN'
2386      'FIN' ::
2388       NBTERMINAUX := I ::
2392      'POUR' J:= 3 'PAS' 1 'JUSQUA' LONGDICO 'FAIRE'
2401      'SI' DICO .(J). 'INFER' 0 'ALORS'
2409         'DEBUT' I:=I+1 ::
2416         DICO2.(I). := -DICO.(J). ::
2427         DICO.(J). := I ::
2434         'SI' J = NBOPERATEURS 'ALORS' NBOPERATEURS := I-NBTERMINAUX
2443         'FIN' ::
2446       NBSYMBOLES := I ::
2450      FLECHE := LONGDICO-1 ::
2458      OU := LONGDICO ::
2460      DICO2.(FLECHE). := DICO.(1). ::
2470      DICO2.(OU). := DICO.(2). ::
2480      'POUR' I := 2, I+GRAMMAIRE.(I).+2 'TANT QUE' GRAMMAIRE.(I-1).
2500      'NONEG' 0 'FAIRE'
2503         'POUR' J:=I-1,I+1 'PAS' 1 'JUSQUA' I+GRAMMAIRE.(I). 'FAIRE'
2523         GRAMMAIRE.(J).:= DICO.(GRAMMAIRE.(J).). ::
2905      'FIN' CHANGERCODE
2906   'FIN' DE PROGRAMME ::
2908   FALGOL
```

REDISTRIBUTION DES CODES

1° Calcul de relations.

Les relations entre symboles sont représentées par des tableaux booléens à deux indices, rangés ici de manière compacte (35 élément par mot-machine) ; les tableaux SUCCESSEUR, GAUCHE, DROITE, COMPATIBLE représentent respectivement les relations σ, γ, δ, ε.

Les relations élémentaires σ_o, γ_o, δ_o s'établissent simplement par balayage du tableau représentant la grammaire. Leurs fermetures transitives σ, γ, δ (pas (c) et (g) de l'algorithme) se calculent par l'algorithme de WARSHALL ([9]) .

La relation ε s'établit par balayage de la grammaire, et à l'aide de γ et δ (pas (h) de l'algorithme).

2° Construction d'un ensemble $\tilde{A}$.

Les ensembles $\tilde{V}_T$ (pas (a) de l'algorithme) et $W[G] = \emptyset$ (pas (e)) sont construits par la procédure COOPTATION (GRAMMAIRE, HAPPYFEW), listée ci-après, qui opère par balayages successifs du tableau GRAMMAIRE.

La propriété d'appartenance à un certain ensemble est représentée par le tableau booléen à un indice HAPPYFEW. HAPPYFEW[I] est vrai si le symbole de code I appartient à l'ensemble considéré.

Initialisé de manière à caractériser l'ensemble A, le tableau HAPPY FEW, après exécution de la procédure, caractérisera $\tilde{A}$.

3° Transformation de la grammaire.

Programmées en ALGOL, les transformations de la grammaire sont des opérations assez lourdes (translation des éléments du tableau GRAMMAIRE). Elles sont donc ici réduites au minimum.

La suppression des symboles parasites (pas (b) de l'algorithme) doit être effectuée séparément, car la relation σ, nécessaire pour déterminer les autres transformations, doit se calculer sur la grammaire sans parasites.

En revanche, la suppression des symboles inaccessibles et vides (pas (d) et (f)) peut être effectuée en une seule fois. En effet, la relation σ calculée pour la grammaire sans parasites reste valable après les autres transformations; et l'ensemble des symboles vides peut déjà être déterminé avant la suppression des inaccessibles.

Le "problème général" du chapitre précédent est donc traité pratiquement dans l'ordre suivant :

- détermination des symboles parasites ;
- construction d'une grammaire sans parasites ;
- calcul de la relation σ ;
- détermination des symboles inaccessibles, semi-vides et essentiellement vides ;
- construction d'une grammaire sans inaccessibles et sans vides ;
- calcul des relations γ, δ et ε .

La grammaire G sans parasites étant représentée par le tableau OLDGRAM, les tableaux booléens VIDE et PEUT'ETRE VIDE caractérisant les ensembles $W_2[G]$ et $W_1[G]$, le programme listé ci-après construit le tableau NEWGRAM, représentant la grammaire $\lambda(u(G))$.

Editeur.

Le problème général de l'éditeur est le suivant : diviser une chaîne de caractères $X_1\ X_2\ \ldots\ X_n$ en une suite de sous-chaînes $u_1, u_2, \ldots, u_k$, telle que chaque $u_i = X_{i,1} \ldots X_{i,n_i}$ représente un symbole terminal.

a) Procédure simplifiée.

La procédure EDITEUR, listée au paragraphe 3, procède de gauche à droite, et traîte un seul symbole.

```
1269  'PROCEDURE' COOPTATION       (HAPPY FEW, GRAMMAIRE) ::
1277  'BOOLEEN''TABLEAU' HAPPY FEW ::
1281  'ENTIER''TABLEAU' GRAMMAIRE ::
1285  'DEBUT' 'ENTIER' I, J, K, L ::
1295     'BOOLEEN' DU NOUVEAU ::
1298  BALAI : DU NOUVEAU := 'FAUX' ::
1304     'POUR' I := 2, I+GRAMMAIRE.(I).+2 'TANT QUE'
1318     GRAMMAIRE.(I-1). 'NONNEG' 0 'FAIRE'
1327     'DEBUT' K := GRAMMAIRE.(I-1). ::
1337        'SI' 'NON' HAPPY FEW .(K). 'ALORS'
1344        'DEBUT' L := GRAMMAIRE.(I). ::
1352           'POUR' J := 1 'PAS' 1 'A' L 'FAIRE'
1361           'SI' 'NON' HAPPY FEW .(GRAMMAIRE.(I+J).). 'ALORS' 'ALLER A'
1374           MAUVAIS ::
1376           HAPPY FEW .(K). := DU NOUVEAU := 'VRAI' ::
1385  MAUVAIS : 'FIN'
1388     'FIN' ::
1390     'SI' DU NOUVEAU 'ALORS' 'ALLER A' BALAI
1394  'FIN' ::
```

Procédure COOPTATION

```
2597     LGRAM:=0 ::
2601     'POUR' I:=2,I+OLDGRAM.(I).+2 'TANTQUE' OLDGRAM.(I-1). 'NONNEG' 0
2622     'FAIRE'
2624  'DEBUT' P := OLDGRAM.(I-1). ::
2634  'SI' VAL (SUCCESSEUR, TERMINAUX+1, F) 'ALORS'
2646     'DEBUT' WORKSPACE.(1, -1). := P ::
2657             WORKSPACE.(1,0). := OLDGRAM.(I). ::
2669        'POUR' J :=    1   'PAS' 1 'JUSQUA' OLDGRAM.(I). 'FAIRE'
2681        'DEBUT' P := OLDGRAM.(I+J). ::
2691           WORKSPACE.(1,J). := 'SI' VIDE.(P). 'ALORS' 0 'SINON'
2706           'SI' PEUT ETRE VIDE .(P). 'ALORS' -P 'SINON' P
2719        'FIN' ::
2718        L:=1 ::
2722        'POUR' N:=1 'PAS' 1 'JUSQUA' L 'FAIRE'
2731        'DEBUT' K:=LGRAM+2 ::
2738           'POUR' M:=-1 'PAS' 1 'JUSQUA' WORKSPACE.(N,0). 'FAIRE'
2753           'DEBUT' P := WORKSPACE.(N, M). ::
2763           'SI' P = 0 'ALORS'
2768           NEWGRAM.(K).:=NEWGRAM.(K).-1
2778           'SINON'
2780           'DEBUT' LGRAM:=LGRAM+1 ::
2787              'SI' P 'SUPER' 0 'ALORS'
2792              NEWGRAM.(LGRAM).:= P
2797              'SINON'
2799              'DEBUT' NEWGRAM.(LGRAM).:=WORKSPACE.(N,M).:= -P ::
2815                 L:=L+1 ::
2821                 'POUR' J:=-1 'PAS' 1 'JUSQUA' M-1,M+1 'PAS' 1
2837                 'JUSQUA' WORKSPACE.(N,0).
2845                 'FAIRE' WORKSPACE.(L,J).:=WORKSPACE.(N,J). ::
2860                 WORKSPACE.(L,M).:=0
2867              'FIN'
2869           'FIN'
2870           'FIN' ::
2872           'SI' LGRAM 'INFEG' K 'ALORS' LGRAM:=K-2
2881        'FIN'
2883     'FIN'
2884     'FIN' ::
2886     LGRAM:=LGRAM+1 ::
2892     NEWGRAM.(LGRAM).:=0 ::
```

CONSTRUCTION D'UN GRAMMAIRE SANS INACCESSIBLES ET SANS VIDES

La suite de symboles u_1 , ..., u_j étant représentée par la chaîne de caractères $X_1 \ldots X_m$, la procédure a pour tâche, au $(j+1)^{e}$ appel, de déterminer un symbole terminal

$$u_{j+1} = X_{m+1} \ldots X_{m+p} .$$

LMAX étant la longueur du plus long symbole terminal, les sous-chaînes $v_i = X_{m+1} \ldots X_{m+i}$, $1 \leq i \leq LMAX$, doivent être comparées avec le dictionnaire. Les symboles terminaux y étant classés par longueur, il suffit de chercher la chaîne de caractères v_i dans la région du dictionnaire contenant les terminaux de longueur i ; de sorte que le dictionnaire complet n'est balayé qu'une fois au plus lors d'un appel de la procédure.

La recherche s'arrête à la première coïncidence d'un v_i avec un symbole terminal, et la procédure prend pour valeur le code de ce symbole; elle prend pour valeur 0 si aucun v_i ne se trouve dans le dictionnaire.

Ainsi, au cas où plusieurs v_i figurent dans le dictionnaire, seul le premier rencontré est pris en considération. D'où l'importance de l'ordre dans lequel les v_i sont examinés.

La politique adoptée ici part de l'hypothèse que les blancs de la chaîne d'entrée, qui ne sont pas nécessairement séparateurs, le sont souvent, et notamment en cas d'ambiguïté. (C'est généralement ainsi qu'on perfore un programme ALGOL, par exemple) . Donc le premier v_i examiné sera celui que délimite le prochain blanc. Ensuite seront examinés les v_i plus courts, puis celui que délimite le blanc suivant, et ainsi de suite.

Si la chaîne d'entrée contient des blancs après les caractères X_{m+p}, X_{m+q}, X_{m+r}, on examine donc successivement v_p, v_{p-1},..., v_1, v_q, v_{q-1},..., v_{p+1}, v_r, v_{r-1},... .

On passe de l'une de ces chaînes de caractères à la suivante au moyen des procédures AVANCER et RECULER. NCARAC indique la longueur i de la chaîne v_i examinée ; MIN prend les valeurs

successives 1, p+1, q+1, ... indiquant jusqu'à quelle valeur de NCARAC on peut reculer ; MAX prend les valeurs successives p, q, r, ... indiquant à partir de quelle valeur de NCARAC il faudra avancer plus loin. La variable booléenne MAXATTEINT devient vraie lorsque NCARAC atteint la valeur LMAX, indiquant qu'il est inutile d'avancer plus loin.

b) Programme général.

La procédure décrite ci-dessus s'arrêtant à la première impossibilité rencontrée, toute erreur sera détectée et localisée à temps. Cependant, il se peut, malgré la politique choisie, que le premier v_i reconnu comme symbole terminal, ne soit pas le bon; en d'autres termes, qu'il soit impossible d'éditer la chaîne de caractères restante, bien que la chaîne complète soit correcte. La procédure ne pourra alors que constater cette impossibilité.

Il nous a donc paru intéressant d'établir un programme plus général (listé ci-après), qui traite la chaîne d'entrée en entier, range les codes des symboles trouvés dans un tableau CHAINE CODEE, et permette le retour en arrière en cas d'erreur.

De plus, une condition nécessaire pour que la chaîne éditée soit une phrase du langage est que le premier et le dernier symboles soient respectivement des successeurs gauche et droit de l'axiome, et que toutes les paires de symboles adjacents soient compatibles. Cette condition peut être testée par l'éditeur.

On considère qu'on a deux piles. L'une contient les caractères de la chaîne d'entrée, qui entrent et sortent par la gauche; c'est le buffer d'entrée géré par les procédures AVANCER et RECULER. L'autre contient les codes des symboles de la chaîne éditée, qui entrent et sortent par la droite; c'est le tableau CHAINE CODEE. Le tableau CHAINE sert d'intermédiaire; on passe du buffer d'entrée à CHAINE, et vice-versa, au moyen d'AVANCER et RECULER; on passe de CHAINE à CHAINE CODEE, et vice-versa,

par une recherche dans le dictionnaire au moyen de la procédure booléenne EN TABLE, listée au paragraphe 3.

Le tableau CHAINE CODEE est vide au début du programme; le buffer d'entrée doit être vide à la fin. L'avance, c'est-à-dire le passage du buffer d'entrée à CHAINE CODEE, se fait de la même façon que dans la procédure simplifiée; cependant, si le symbole trouvé n'est pas compatible avec le précédent (pour le premier symbole de la chaîne: s'il n'est pas successeur gauche de l'axiome), il est ignoré et la recherche se poursuit. Si le symbole trouvé vérifie la relation voulue, on empile son code dans CHAINECODEE, et les valeurs de MIN, MAX et MAX ATTEINT dans les tableaux PILE 1, PILE 2 et PILE BOOLE, respectivement.

Le recul a lieu dans deux situations: s'il s'avère impossible de déterminer le symbole suivant; et si le dernier symbole de la chaîne n'est pas successeur droit de l'axiome. Le code du dernier symbole est alors retiré de la pile, la chaîne de caractère correspondante est replacée dans CHAINE, les valeurs empilées de MIN, MAX et MAX ATTEINT sont restaurées, et la recherche dans le dictionnaire reprend au point où elle était à la rencontre du symbole fautif.

Ce programme, très général, doit être capable d'éditer n'importe quelle phrase du langage; encore faut-il pour cela que le buffer d'entrée soit assez grand pour permettre le recul dans tous les cas. Cette condition étant supposée réalisée, considérons une chaîne de caractères représentant une phrase du langage. Si les blancs y sont disposés de manière à délimiter exactement les symboles, l'édition sera rapide. Si les blancs sont disposés de façon quelconque, l'édition sera beaucoup plus lente. Si enfin la chaîne contient une erreur, même minime, celle-ci ne sera pas localisée, et le programme gaspillera du temps à essayer toutes les combinaisons avant de déclarer forfait.

```
3871          I := 1 ::
3875          NCARAC := 0 ::
3879    EN AVANT : MIN := NCARAC+1 ::
3887          AVANCER (LMAX, MAX ATTEINT) ::
3894          'SI' NCARAC = 0 'ALORS'
3899          'ALLER A' 'SI' VAL (DROITE, AXIOME, CHAINE CODEE.(I-1).)
3914          'ALORS' C EST BON 'SINON' CN RECULE ::
3919          MAX := NCARAC ::
3923    Y EST IL : 'SI' EN TABLE 'ALORS'
3928          'DEBUT' 'SI' 'SI' I=1 'ALORS' VAL (GAUCHE, AXIOME, CODE)
3943             'SINON' VAL (COMPATIBLE, CHAINE CODEE.(I-1)., CODE) 'ALORS'
3958             'DEBUT' CHAINE CODEE.(I). := CODE ::
3966                PILE 1.(I). := MIN ::
3973                PILE 2.(I). := MAX ::
3980             PILE BOOLE .(I). := MAX ATTEINT ::
3987                I := I+1 ::
3993                NCARAC := 0 ::
3997                'ALLER A' EN AVANT
3998             'FIN'
4000          'FIN' ::
4002    ON CHANGE : 'SI' NCARAC 'SUPER' MIN 'ALORS'
4009          'DEBUT' RECULER (1, IMPOS) ::
4017             'ALLER A' 'SI' IMPOS 'ALORS' ERREUR 'SINON' Y EST IL
4023          'FIN' ::
4026          'SI' MAX ATTEINT 'ALORS'
4029          'DEBUT' 'SI' I = 1 'ALORS'
4035             'DEBUT' RCH :: SORTEXTE (''ERREUR EDITION'') ::
4047                SAUTLIGNE :: RCH ::
4051                'ALLER A' ERREUR
4052             'FIN' ::
4055             RECULER (NCARAC, IMPOS) ::
4062             'SI' IMPOS 'ALORS' 'ALLER A' ERREUR ::
4068    ON RECULE : I := I - 1 ::
4076             L := DICO.(CHAINE CODEE.(I).). ::
4086             MIN := PILE 1.(I). ::
4093             MAX := PILE2.(I). ::
4100             NCARAC := DICO.(L). ::
4107             NMOT := (NCARAC-1) '/' 6 + 1 ::
4119             MAX ATTEINT := PILE BOOLE .(I). ::
4126             'POUR' J := 1 'PAS' 1 'A' NMOT 'FAIRE'
4135             CHAINE.(J). := DICO.(L+J). ::
4147             'ALLER A' ON CHANGE
4148          'FIN' ::
4151          AVANCER (MAX, MAX ATTEINT) ::
4158          'ALLER A' EN AVANT ::
4161    C EST BON :
```

EDITEUR (Programme général)

c) "L'oeuf de Colomb".

D'un point de vue purement théorique, un problème échappe encore au programme général décrit ci-dessus. Il n'est pas interdit de supposer qu'une chaîne de caractères puisse être interprétée de plusieurs façons comme une suite de symboles terminaux, vérifiant toutes les relations nécessaires; et que cependant une seule de ces suites soit une phrase du langage. En ce cas, seul le programme d'analyse syntaxique s'apercevrait d'une éventuelle "erreur d'édition", et il faudrait lui donner les moyens d'imposer à l'éditeur un retour en arrière.

D'un point de vue pratique, cet éditeur possède déjà la généralité, donc la lenteur, d'un analyseur syntaxique, procédant par essais successifs. Or, la plupart des programmes d'analyse sont plus subtils, et il serait irrationnel de les faire précéder par un tel éditeur; mieux vaut appliquer à l'édition déjà ce qui fait la finesse de l'analyseur.

Ces deux points de vue conduisent à la même constatation : un programme d'édition, s'il se veut absolument général, doit être intégré au programme d'analyse syntaxique.

Cela suppose que la décomposition des symboles en caractères fasse partie intégrante de la grammaire; que les caractères soient les seuls symboles terminaux; et que l'éditeur soit remplacé par une simple procédure de lecture de caractères.

C'est la méthode appliquée par A. COLMERAUER pour la "Recherche d'erreurs syntaxiques en ALGOL" ([13],[14]), avec des règles telles que

'DEBUT' ::= ' D E B U T '

7. Enchaînement des programmes.

En résumé, nous avons décrit dans les paragraphes précédents :

deux programmes de "lecture", au sens large : la lecture des règles, et l'éditeur ;

et deux programmes de "calcul"; en effet, les opérations décrites au paragraphe 5 ont été réparties en deux programmes séparés : d'une part, les transformations de la grammaire et le calcul de σ, d'autre part le calcul de γ, δ et ε.

Ce sont quatre programmes ALGOL indépendants, prévus pour être enchaînés suivant le schéma: lecture des règles - transformation de la grammaire - calcul des relations - éditeur - un programme quelconque d'analyse syntaxique. Les maillons de la chaîne communiquent entre eux au moyen d'une unité de disques ; à la fin de chaque programme, les tableaux représentant la grammaire, le dictionnaire et les relations sont écrits sur disque ; ils y sont relus au début de chaque programme, à l'exception du programme de lecture.

Ces rangements intermédiaires sur disque sont faits à tous les stades sous une forme standard, ce qui permet de sauter un ou plusieurs maillons de la chaîne. Ainsi, on pourra passer directement de la lecture des règles au calcul des relations, si l'on sait que la grammaire donnée est sans parasites, sans inaccessibles et sans vides. De même, on pourra enchaîner un programme d'analyse sans passer par l'éditeur, si la grammaire admet les caractères comme terminaux, ou si l'analyseur fait appel à la procédure simplifiée d'édition.

Nous disposons donc d'une relative souplesse d'utilisation. Celle-ci serait plus grande si les programmes pouvaient s'articuler selon leurs phases élémentaires : lecture des règles, calcul de telle relation, suppression de tel ensemble de symboles , impression de la grammaire ou de tel tableau, édition , ... Chaque phase pourrait être appelée , soit,

sous forme de procédure, par les programmes d'analyse syntaxique, soit par l'utilisateur, dans le cadre d'un système conversationnel où ces programmes seraient disponibles.

Ayant mis nos travaux dans une bibliothèque du système de téléprocessing DIAMAG I, nous avons tenté d'automatiser le rangement intermédiaire des tableaux sur disque, dans l'idée de découper ensuite les programmes en phases élémentaires, pour une utilisation "à la carte"; et d'y adjoindre éventuellement quelques procédures de chaînage standard.

L'état alors embryonnaire du système DIAMAG I, joint à notre inexpérience, ne nous a malheureusement pas permis de mener cette tentative à bien.

8. Exemple.

Le petit exemple suivant a été construit spécialement pour mettre en évidence les différents types de symboles, et tester le programme transformant la grammaire.

```
     AXIOME  ::=  BON-1  BON-2  TERM-1
               /  BON-2  VIDE-1
      BON-1  ::=  PARASITE-1  TERM-2
               /  TERM-1  TERM-2  TERM-3
      BON-2  ::=  PEUT-ETRE-1  BON-3  BON-4
               /  BON-4  TERM-4  PEUT-ETRE-2  VIDE-2
      BON-3  ::=  HORS-1  PARASITE-2
               /  TERM-3  TERM-2
      BON-4  ::=  TERM-1  PARASITE-3
               /  TERM-2  BON-3
 PARASITE-1  ::=  TERM-1  PARASITE-1
 PARASITE-2  ::=  PARASITE-3  BON-4
 PARASITE-3  ::=  BON-1  PARASITE-2  TERM-1  TERM-2
               /  TERM-2  PARASITE-1
PEUT-ETRE-1  ::=  BON-2  BON-4
               /  VIDE-1
PEUT-ETRE-2  ::=
               /  TERM-1  BON-4
      1      ::=  PARASITE-1
               /  VIDE-2
     VIDE-2  ::=  VIDE-1  VIDE-2
               /
     HORS-1  ::=  TERM-1
     HORS-2  ::=  BON-2  TERM-2  TERM-3
```

SYMBOLES TERMINAUX

1 TERM-1
2 TERM-2
3 TERM-3
4 TERM-4

SYMBOLES NON TERMINAUX

5 AXIOME
6 BON-1
7 BON-2
8 VIDE-1
9 PARASITE-1
10 PEUT-ETRE-1
11 BON-3
12 BON-4
13 PEUT-ETRE-2
14 VIDE-2
15 HORS-1
16 PARASITE-2
17 PARASITE-3
18 HORS-2

6044,C192 F. MARTIN

NON-TERMINAUX PARASITES :

PARASITE-1
PARASITE-2
PARASITE-3

6044,C192 F. MARTIN

GRAMMAIRE DEPARASITEE

```
      AXIOME  ::=  BON-1  BON-2  TERM-1
                /  BON-2  VIDE-1
       BON-1  ::=  TERM-1  TERM-2  TERM-3
       BON-2  ::=  PEUT-ETRE-1  BON-3  BON-4
                /  BON-4  TERM-4  PEUT-ETRE-2  VIDE-2
       BON-3  ::=  TERM-3  TERM-2
       BON-4  ::=  TERM-2  BON-3
PEUT-ETRE-1   ::=  BON-2  BON-4
                /  VIDE-1
PEUT-ETRE-2   ::=
                /  TERM-1  BON-4
      VIDE-1  ::=  VIDE-2
      VIDE-2  ::=  VIDE-1  VIDE-2
                /
      HORS-1  ::=  TERM-1
      HORS-2  ::=  BON-2  TERM-2  TERM-3
```

MATRICE DE SUCCESSION

```
                      111111111
             123456789012345678

    AXIOME    5  1111.111.11111....
    BON-1     6  111..............
    BON-2     7  1111..1..11111....
    VIDE-1    8  .......1.....1....
PARASITE-1    9  ..................
PEUT-ETRE-1  10  1111..11.11111....
    BON-3    11  .11..............
    BON-4    12  .11.......1.......
PEUT-ETRE-2  13  111.......11......
    VIDE-2   14  .......1.....1....
    HORS-1   15  1...................
PARASITE-2   16  ..................
PARASITE-3   17  ..................
    HORS-2   18  1111..11.11111....
```

6044,C192 F. MARTIN

SYMBOLES VIDES :

VIDE-1
VIDE-2

SYMBOLES POUVANT ETRE VIDES :

PEUT-ETRE-1
PEUT-ETRE-2

SYMBOLES INATTEIGNABLES :

HORS-1
HORS-2

6044,C192 F. MARTIN

GRAMMAIRE EQUIVALENTE SANS VIDE

```
    AXIOME  ::=  BON-1  BON-2  TERM-1
              /  BON-2
     BON-1  ::=  TERM-1  TERM-2  TERM-3
     BON-2  ::=  PEUT-ETRE-1  BON-3  BON-4
              /  BON-3  BON-4
              /  BON-4  TERM-4  PEUT-ETRE-2
              /  BON-4  TERM-4
     BON-3  ::=  TERM-3  TERM-2
     BON-4  ::=  TERM-2  BON-3
PEUT-ETRE-1 ::=  BON-2  BON-4
PEUT-ETRE-2 ::=  TERM-1  BON-4
```

III. APPLICATION A ALGOL 60.

A titre d'exemple privilégié, nous avons appliqué nos programmes à la grammaire d'ALGOL 60. Comme définition de la grammaire, nous avons pris l'ensemble des règles sous forme de Backus figurant dans le rapport revisé ([5]), tel qu'il apparaît en traduction française dans [7] , avec l'axiome <programme>.

1. Vides.

La grammaire ainsi définie contient une seule règle ayant la partie droite vide :

<vide> ::=

Il en découle cependant l'existence de deux symboles essentiellement vides, <vide> et <instruction vide>, et de quelques symboles semi-vides, <instruction>, <corps de procédure>, La construction de la grammaire sans vides appelle les remarques suivantes :

1° Conformément à la théorie, les symboles semi-vides subsistent dans la grammaire transformée, mais en tant que symboles formels. Considérés comme métavariables recouvrant une certaine "notion", leur signification n'est plus la même. Ainsi, le symbole <instruction> subsiste, mais avec la signification "instruction non vide" .

2° On sait que la définition du langage n'est pas tout entière dans la grammaire. Parmi les phrases terminales syntaxiquement correctes, certaines sont des programmes ALGOL, et d'autres, que nous appellerons "pseudo-programmes", sont déclarées incorrectes par la sémantique. Naturellement, chacune des règles données dans le rapport Algol, si elle peut intervenir dans la dérivation de pseudo-programmes, permet de dériver au moins un programme correct. Ce n'est

pas le cas de certaines règles introduites par la transformation de la grammaire, qui ne peuvent donner lieu qu'à la dérivation de pseudo-programmes.

Par exemple, la règle

<tête de procédure> ::= <identificateur de procédure>
<partie paramètre formel>;<partie valeur> <partie spécification>

où <partie paramètre formel>,<partie valeur> et <partie spécification> sont semi-vides, donne naissance dans la grammaire transformée, à huit règles, dont quatre ne permettent de dériver que des pseudo-programmes :

<tête de procédure>::=<identificateur de procédure>
<partie valeur> <partie spécification>

<tête de procédure>::= <identificateur de procédure><partie valeur

<tête de procédure>::=
<identificateur de procédure><partie spécification>

<tête de procédure> ::=<identificateur de procédure>
<partie paramètre formel><partie valeur>

2. Inaccessibles.

La grammaire d'Algol ne contient pas de symboles parasites; on pourrait s'attendre qu'elle ne contienne pas non plus d'inaccessibles; ce n'est pas le cas. On y trouve, par exemple la règle :

<nombre>::= <nombre sans signe>
| + <nombre sans signe>
| - <nombre sans signe>

Or, si <nombre sans signe> est un successeur de l'axiome <programme> , <nombre> ne l'est pas.

En fait, deux des notions définies au paragraphe 2 du rapport revisé, <nombre> et <symbole de base> , n'interviennent pas dans

la définition du <programme> . Il faudrait, pour avoir une grammaire sans inaccessibles, ne considérer qu'une partie des règles de ce paragraphe; mais leur choix implique déjà une certaine étude de la grammaire, et il est avantageux d'en laisser le soin à l'algorithme d'élimination des inaccessibles.

Partant de l'ensemble des règles du rapport révisé, et en considérant successivement comme axiome les symboles <programme>, <nombre> et <symbole de base> , on obtient, par élimination des inaccessibles, trois grammaires "indépendantes" , chacune définissant la notion correspondante. Les ensembles de règles de ces grammaires ne sont pas inclus l'un dans l'autre, et leur réunion est l'ensemble des règles du rapport.

On trouvera, listés ci-après :

- l'ensemble des règles données, avec nos notations;
- la liste des symboles inaccessibles et vides, correspondant à l'axiome <programme> ;
- la grammaire transformée, correspondant à l'axiome <programme>
- la grammaire transformée, correspondant à l'axiome <nombre> .

Quand à la grammaire avec l'axiome <symbole de base> , elle définit un "langage" qui se réduit au vocabulaire terminal.

GRAMMAIRE ALGOL I

6044,0192,0000 F. MARTIN

```
                    VIDE  ::=
          SYMBOLE-DE-BASE  ::=  LETTRE
                           //   CHIFFRE
                           //   VALEUR-LOGIQUE
                           //   DELIMITEUR
                  LETTRE  ::=  A
                           //   B
                           //   C
                           //   D
                           //   E
                           //   F
                           //   G
                           //   H
                           //   I
                           //   J
                           //   K
                           //   L
                           //   M
                           //   N
                           //   O
                           //   P
                           //   Q
                           //   R
                           //   S
                           //   T
                           //   U
                           //   V
                           //   W
                           //   X
                           //   Y
                           //   Z
                 CHIFFRE  ::=  0
                           //   1
                           //   2
                           //   3
                           //   4
                           //   5
                           //   6
                           //   7
                           //   8
                           //   9
          VALEUR-LOGIQUE  ::=  'TRUE'
                           //   'FALSE'
              DELIMITEUR  ::=  OPERATEUR
                           //   SEPARATEUR
                           //   CROCHET
                           //   DECLARATEUR
                           //   SPECIFIEUR
                OPERATEUR  ::=  OPERATEUR-ARITHMETIQUE
                           //   OPERATEUR-DE-RELATION
                           //   OPERATEUR-LOGIQUE
                           //   OPERATEUR-SEQUENTIEL
OPERATEUR-ARITHMETIQUE  ::=  +
                           //   -
```

GRAMMAIRE ALGOL II

6044,0192,0000 F. MARTIN

```
                                //  *
                                //  /
                                //  '/'
                                //  **
        OPERATEUR-DE-RELATION  ::=  'LESS'
                                //  'NOTGREATER'
                                //  =
                                //  'NOTLESS'
                                //  'GREATER'
                                //  'NOTEQUAL'
            OPERATEUR-LOGIQUE  ::=  'IDENT'
                                //  'IMPL'
                                //  'OR'
                                //  'AND'
                                //  'NOT'
         OPERATEUR-SEQUENTIEL  ::=  'GOTO'
                                //  'IF'
                                //  'THEN'
                                //  'ELSE'
                                //  'FOR'
                                //  'DO'
                    SEPARATEUR  ::=  ,
                                //  .
                                //  '10'
                                //  :
                                //  ::
                                //  :=
                                //  'STEP'
                                //  'UNTIL'
                                //  'WHILE'
                                //  'COMMENT'
                       CROCHET  ::=  (
                                //  )
                                //  .(
                                //  ).
                                //  '('
                                //  ')'
                                //  'BEGIN'
                                //  'END'
                   DECLARATEUR  ::=  'OWN'
                                //  'BOOLEAN'
                                //  'INTEGER'
                                //  'REAL'
                                //  'ARRAY'
                                //  'SWITCH'
                                //  'PROCEDURE'
                   SPECIFIEUR  ::=  'STRING'
                                //  'LABEL'
                                //  'VALUE'
              IDENTIFICATEUR  ::=  LETTRE
                                //  IDENTIFICATEUR  LETTRE
                                //  IDENTIFICATEUR  CHIFFRE
         ENTIER-SANS-SIGNE  ::=  CHIFFRE
                                //  ENTIER-SANS-SIGNE  CHIFFRE
                    ENTIER  ::=  ENTIER-SANS-SIGNE
```

GRAMMAIRE ALGOL IIIa

6044,0192,0000 F. MARTIN

```
                                                    //
                                                    //
                                  PARTIE-DECIMALE  ::=
                               FACTEUR-DE-CADRAGE  ::=
                                  NOMBRE-DECIMAL   ::=
                                                    //
                                                    //
                               NOMBRE-SANS-SIGNE   ::=
                                                    //
                                                    //
                                          NOMBRE   ::=
                                                    //
                                                    //
                                  CHAINE-OUVERTE   ::=
                                                    //
                                                    //
                                          CHAINE   ::=
                                      EXPRESSION   ::=
                                                    //
                                                    //
                        IDENTIFICATEUR-DE-VARIABLE ::=
                                  VARIABLE-SIMPLE  ::=
                              EXPRESSION-EN-INDICE ::=
                                  LISTE-D'INDICES  ::=
                                                    //
                          IDENTIFICATEUR-DE-TABLEAU ::=
                                  VARIABLE-INDICEE  ::=
                                          VARIABLE  ::=
                                                    //
                       IDENTIFICATEUR-DE-PROCEDURE  ::=
                                  PARAMETRE-EFFECTIF ::=
                                                    //
                                                    //
                                                    //
                                                    //
                                  CHAINE-DE-LETTRES ::=
                                                    //
                                DELIMITEUR-DE-PARAMETRES ::=
                                                    //
                          LISTE-DE-PARAMETRES-EFFECTIFS ::=
                                                    //
                                PARTIE-PARAMETRE-EFFECTIF ::=
                                                    //
                                  INDICATEUR-DE-FONCTION ::=
                                      OPERATEUR-ADDITIF  ::=
                                                    //
                                OPERATEUR-MULTIPLICATIF  ::=
                                                    //
                                                    //
                                PRIMAIRE-ARITHMETIQUE    ::=
                                                    //
                                                    //
                                                    //
                                            FACTEUR  ::=
                                                    //
```

```
+  ENTIER-SANS-SIGNE
-  ENTIER-SANS-SIGNE
.  ENTIER-SANS-SIGNE
'10'  ENTIER
ENTIER-SANS-SIGNE
PARTIE-DECIMALE
ENTIER-SANS-SIGNE  PARTIE-DECIMALE
NOMBRE-DECIMAL
FACTEUR-DE-CADRAGE
NOMBRE-DECIMAL  FACTEUR-DE-CADRAGE
NOMBRE-SANS-SIGNE
+  NOMBRE-SANS-SIGNE
-  NOMBRE-SANS-SIGNE
CHAINE-PROPRE
'('  CHAINE-CUVERTE  ')'
CHAINE-OUVERTE  CHAINE-OUVERTE
'('  CHAINE-CUVERTE  ')'
EXPRESSION-ARITHMETIQUE
EXPRESSION-BOOLEENNE
EXPRESSION-DE-DESIGNATION
IDENTIFICATEUR
IDENTIFICATEUR-DE-VARIABLE
EXPRESSION-ARITHMETIQUE
EXPRESSION-EN-INDICE
LISTE-D'INDICES  ,  EXPRESSION-EN-INDICE
IDENTIFICATEUR
IDENTIFICATEUR-DE-TABLEAU  .(  LISTE-D'INDICES  ).
VARIABLE-SIMPLE
VARIABLE-INDICEE
IDENTIFICATEUR
CHAINE
EXPRESSION
IDENTIFICATEUR-DE-TABLEAU
IDENTIFICATEUR-D'AIGUILLAGE
IDENTIFICATEUR-DE-PROCEDURE
LETTRE
CHAINE-DE-LETTRES  LETTRE
,
)  CHAINE-DE-LETTRES  :  (
PARAMETRE-EFFECTIF
LISTE-DE-PARAMETRES-EFFECTIFS  DELIMITEUR-DE-PARAMETRES  PARAMETRE-EFFECT
VIDE
(  LISTE-DE-PARAMETRES-EFFECTIFS  )
IDENTIFICATEUR-DE-PROCEDURE  PARTIE-PARAMETRE-EFFECTIF
+
-
*
/
'/'
NOMBRE-SANS-SIGNE
VARIABLE
INDICATEUR-DE-FONCTION
(  EXPRESSION-ARITHMETIQUE  )
PRIMAIRE-ARITHMETIQUE
FACTEUR  **  PRIMAIRE-ARITHMETIQUE
```

GRAMMAIRE ALGOL IVa

6044,0192,0C00 F. MARTIN

```
                                        TERME  ::=
                                               //
                  EXPRESSION-ARITHMETIQUE-SIMPLE  ::=
                                               //
                                               //
                                PROPOSITION-SI  ::=
                            EXPRESSION-ARITHMETIQUE  ::=
                                               //
                                      RELATION  ::=

                                PRIMAIRE-BOOLEEN  ::=
                                               //
                                               //
                                               //
                                               //
                              SECONDAIRE-BOOLEEN  ::=
                                               //
                                 FACTEUR-BOOLEEN  ::=
                                               //
                                   TERME-BOOLEEN  ::=
                                               //
                                     IMPLICATION  ::=
                                               //
                        EXPRESSION-BOOLEENNE-SIMPLE  ::=
                                               //
                                EXPRESSION-BOOLEENNE  ::=
                                               //
                                       ETIQUETTE  ::=
                                               //
                    IDENTIFICATEUR-D'AIGUILLAGE  ::=
                        INDICATEUR-D'AIGUILLAGE  ::=
                EXPRESSION-DE-DESIGNATION-SIMPLE  ::=
                                               //
                                               //
                          EXPRESSION-DE-DESIGNATION  ::=
                                               //

          INSTRUCTION-NON-ETIQUETEE-DE-BASE  ::=
                                               //
                                               //
                                               //
                                INSTRUCTION-DE-BASE  ::=
                                               //
                        INSTRUCTION-INCONDITIONNELLE  ::=
                                               //
                                               //
                                       INSTRUCTION  ::=
                                               //
                                               //
                        FIN-D'INSTRUCTION-COMPOSEE  ::=
                                               //
                                    EN-TETE-DE-BLOC  ::=
                                               //
                              COMPOSEE-NON-ETIQUETEE  ::=
                                  BLOC-NON-ETIQUETE  ::=
```

```
FACTEUR
TERME  OPERATEUR-MULTIPLICATIF  FACTEUR
TERME
OPERATEUR-ADDITIF  TERME
EXPRESSION-ARITHMETIQUE-SIMPLE  OPERATEUR-ADDITIF  TERME
'IF'  EXPRESSION-BOOLEENNE  'THEN'
EXPRESSION-ARITHMETIQUE-SIMPLE
PROPOSITION-SI  EXPRESSION-ARITHMETIQUE-SIMPLE  'ELSE'  EXPRESSION-ARITHMETI
EXPRESSION-ARITHMETIQUE-SIMPLE  OPERATEUR-DE-RELATION
EXPRESSION-ARITHMETIQUE-SIMPLE
VALEUR-LOGIQUE
VARIABLE
INDICATEUR-DE-FONCTION
RELATION
(  EXPRESSION-BOOLEENNE  )
PRIMAIRE-BOOLEEN
'NOT'  PRIMAIRE-BOOLEEN
SECONDAIRE-BOOLEEN
FACTEUR-BOOLEEN  'AND'  SECONDAIRE-BOOLEEN
FACTEUR-BOOLEEN
TERME-BOOLEEN  'OR'  FACTEUR-BOOLEEN
TERME-BOOLEEN
IMPLICATION  'IMPL'  TERME-BOOLEEN
IMPLICATION
EXPRESSION-BOOLEENNE-SIMPLE  'IDENT'  IMPLICATION
EXPRESSION-BOOLEENNE-SIMPLE
PROPOSITION-SI  EXPRESSION-BOOLEENNE-SIMPLE  'ELSE'  EXPRESSION-BOOLEENNE
IDENTIFICATEUR
ENTIER-SANS-SIGNE
IDENTIFICATEUR
IDENTIFICATEUR-D'AIGUILLAGE  .(  EXPRESSION-EN-INDICE  ).
ETIQUETTE
INDICATEUR-D'AIGUILLAGE
(  EXPRESSION-DE-DESIGNATION  )
EXPRESSION-DE-DESIGNATION-SIMPLE
PROPOSITION-SI  EXPRESSION-DE-DESIGNATION-SIMPLE  'ELSE'
EXPRESSION-DE-DESIGNATION
INSTRUCTION-D'AFFECTATION
INSTRUCTION-ALLER-A
INSTRUCTION-VIDE
INSTRUCTION-PROCEDURE
INSTRUCTION-NON-ETIQUETEE-DE-BASE
ETIQUETTE  :  INSTRUCTION-DE-BASE
INSTRUCTION-DE-BASE
INSTRUCTION-COMPOSEE
BLOC
INSTRUCTION-INCONDITIONNELLE
INSTRUCTION-CONDITIONNELLE
INSTRUCTION-POUR
INSTRUCTION  'END'
INSTRUCTION  ::  FIN-D'INSTRUCTION-COMPOSEE
'BEGIN'  DECLARATION
EN-TETE-DE-BLOC  ::  DECLARATION
'BEGIN'  FIN-D'INSTRUCTION-COMPOSEE
EN-TETE-DE-BLOC  ::  FIN-D'INSTRUCTION-COMPOSEE
```

GRAMMAIRE ALGOL Va

6044,0192,0000 F. MARTIN

INSTRUCTION-COMPOSEE ::=
//
BLOC ::=
//
PROGRAMME ::=
//
PARTIE-GAUCHE ::=
//
LISTE-DE-PARTIES-GAUCHES ::=
//
INSTRUCTION-D'AFFECTATION ::=
//
INSTRUCTION-ALLER-A ::=
INSTRUCTION-VIDE ::=
INSTRUCTION-SI ::=
INSTRUCTION-CONDITIONNELLE ::=
//
//
//
ELEMENT-D'UNE-LISTE-DE-POUR ::=
//

//
LISTE-DE-POUR ::=
//
PROPOSITION-POUR ::=
INSTRUCTION-POUR ::=
//
INSTRUCTION-PROCEDURE ::=
DECLARATION ::=
//
//
//
LISTE-DE-TYPE ::=
//
TYPE ::=
//
//
DECLARATION-DE-TYPE ::=
TYPE-LOCAL-OU-REMANENT ::=
//
BORNE-INFERIEURE ::=
BORNE-SUPERIEURE ::=
PAIRE-DE-BORNES ::=
LISTE-DE-PAIRES-DE-BORNES ::=
//
SECTION-DE-TABLEAU ::=
//
LISTE-DE-TABLEAU ::=
//
DECLARATION-DE-TABLEAU ::=
//
LISTE-D'AIGUILLAGE ::=
//
DECLARATION-D'AIGUILLAGE ::=

```
COMPOSEE-NON-ETIQUETEE
ETIQUETTE  :  INSTRUCTION-COMPOSEE
BLOC-NON-ETIQUETE
ETIQUETTE  :  BLOC
BLOC
INSTRUCTION-COMPOSEE
VARIABLE  :=
IDENTIFICATEUR-DE-PROCEDURE  :=
PARTIE-GAUCHE
LISTE-DE-PARTIES-GAUCHES  PARTIE-GAUCHE
LISTE-DE-PARTIES-GAUCHES  EXPRESSION-ARITHMETIQUE
LISTE-DE-PARTIES-GAUCHES  EXPRESSION-BOOLEENNE
'GOTO'  EXPRESSION-DE-DESIGNATION
VIDE
PROPOSITION-SI  INSTRUCTION-INCONDITIONNELLE
INSTRUCTION-SI
INSTRUCTION-SI  'ELSE'  INSTRUCTION
PROPOSITION-SI  INSTRUCTION-POUR
ETIQUETTE  :  INSTRUCTION-CONDITIONNELLE
EXPRESSION-ARITHMETIQUE
EXPRESSION-ARITHMETIQUE  'STEP'  EXPRESSION-ARITHMETIQUE  'UNTIL'
EXPRESSION-ARITHMETIQUE
EXPRESSION-ARITHMETIQUE  'WHILE'  EXPRESSION-BOOLEENNE
ELEMENT-D'UNE-LISTE-DE-POUR
LISTE-DE-POUR  ,  ELEMENT-D'UNE-LISTE-DE-POUR
'FOR'  VARIABLE  :=  LISTE-DE-POUR  'DO'
PROPOSITION-POUR  INSTRUCTION
ETIQUETTE  :  INSTRUCTION-POUR
IDENTIFICATEUR-DE-PROCEDURE  PARTIE-PARAMETRE-EFFECTIF
DECLARATION-DE-TYPE
DECLARATION-DE-TABLEAU
DECLARATION-D'AIGUILLAGE
DECLARATION-DE-PROCEDURE
VARIABLE-SIMPLE
VARIABLE-SIMPLE  ,  LISTE-DE-TYPE
'REAL'
'INTEGER'
'BOOLEAN'
TYPE-LOCAL-OU-REMANENT  LISTE-DE-TYPE
TYPE
'OWN'  TYPE
EXPRESSION-ARITHMETIQUE
EXPRESSION-ARITHMETIQUE
BORNE-INFERIEURE  :  BORNE-SUPERIEURE
PAIRE-DE-BORNES
LISTE-DE-PAIRES-DE-BORNES  ,  PAIRE-DE-BORNES
IDENTIFICATEUR-DE-TABLEAU  .(  LISTE-DE-PAIRES-DE-BORNES  ).
IDENTIFICATEUR-DE-TABLEAU  ,  SECTION-DE-TABLEAU
SECTION-DE-TABLEAU
LISTE-DE-TABLEAU  ,  SECTION-DE-TABLEAU
'ARRAY'  LISTE-DE-TABLEAU
TYPE-LOCAL-OU-REMANENT  'ARRAY'  LISTE-DE-TABLEAU
EXPRESSION-DE-DESIGNATION
LISTE-D'AIGUILLAGE  ,  EXPRESSION-DE-DESIGNATION
'SWITCH'  IDENTIFICATEUR-D'AIGUILLAGE  :=  LISTE-D'AIGUILLAGE
```

GRAMMAIRE ALGOL VIa

6044,0192,0000 F. MARTIN

PARAMETRE-FORMEL ::=

LISTE-DE-PARAMETRES-FORMELS ::=
//

PARTIE-PARAMETRE-FORMEL ::=
//

LISTE-D'IDENTIFICATEURS ::=
//

PARTIE-VALEUR ::=
//

SPECIFICATEUR ::=
//
//
//
//
//
//
//

PARTIE-SPECIFICATION ::=
//
//

TETE-DE-PROCEDURE ::=

CORPS-DE-PROCEDURE ::=
//

DECLARATION-DE-PROCEDURE ::=
//

GRAMMAIRE ALGOL VIb


```
IDENTIFICATEUR
PARAMETRE-FORMEL
LISTE-DE-PARAMETRES-FORMELS  DELIMITEUR-DE-PARAMETRES  PARAMETRE-FORMEL
VIDE
(  LISTE-DE-PARAMETRES-FORMELS  )
IDENTIFICATEUR
LISTE-D'IDENTIFICATEURS  ,  IDENTIFICATEUR
'VALUE'  LISTE-D'IDENTIFICATEURS  ::
VIDE
'STRING'
TYPE
'ARRAY'
TYPE  'ARRAY'
'LABEL'
'SWITCH'
'PROCEDURE'
TYPE  'PROCEDURE'
VIDE
SPECIFICATEUR  LISTE-D'IDENTIFICATEURS  ::
PARTIE-SPECIFICATION  SPECIFICATEUR  LISTE-D'IDENTIFICATEURS  ::
IDENTIFICATEUR-DE-PROCEDURE  PARTIE-PARAMETRE-FORMEL  ::  PARTIE-VALEUR
PARTIE-SPECIFICATION
INSTRUCTION
CODE
'PROCEDURE'  TETE-DE-PROCEDURE  CORPS-DE-PROCEDURE
TYPE  'PROCEDURE'  TETE-DE-PROCEDURE  CORPS-DE-PROCEDURE
```

6044,0192,0000 F. MARTIN

SYMBOLES VIDES :

VIDE
INSTRUCTION-VIDE

SYMBOLES POUVANT ETRE VIDES :

PARTIE-PARAMETRE-EFFECTIF
INSTRUCTION-NON-ETIQUETEE-DE-BASE
INSTRUCTION-DE-BASE
INSTRUCTION-INCONDITIONNELLE
INSTRUCTION
PARTIE-PARAMETRE-FORMEL
PARTIE-VALEUR
PARTIE-SPECIFICATION
CORPS-DE-PROCEDURE

SYMBOLES INACCESSIBLES :

'COMMENT'
SYMBOLE-DE-BASE
DELIMITEUR
OPERATEUR
SEPARATEUR
CROCHET
DECLARATEUR
SPECIFIEUR
OPERATEUR-ARITHMETIQUE
OPERATEUR-LOGIQUE
OPERATEUR-SEQUENTIEL
NOMBRE

GRAMMAIRE ALGOL TRANSFORMEE I

6044,0192,0000 F. MARTIN

GRAMMAIRE EQUIVALENTE SANS VIDE

```
                    LETTRE  ::=  A
                             //  B
                             //  C
                             //  D
                             //  E
                             //  F
                             //  G
                             //  H
                             //  I
                             //  J
                             //  K
                             //  L
                             //  M
                             //  N
                             //  O
                             //  P
                             //  Q
                             //  R
                             //  S
                             //  T
                             //  U
                             //  V
                             //  W
                             //  X
                             //  Y
                             //  Z
                   CHIFFRE  ::=  0
                             //  1
                             //  2
                             //  3
                             //  4
                             //  5
                             //  6
                             //  7
                             //  8
                             //  9
           VALEUR-LOGIQUE  ::=  'TRUE'
                             //  'FALSE'
OPERATEUR-DE-RELATION  ::=  'LESS'
                             //  'NOTGREATER'
                             //  =
                             //  'NOTLESS'
                             //  'GREATER'
                             //  'NOTEQUAL'
              IDENTIFICATEUR  ::=  LETTRE
                             //  IDENTIFICATEUR  LETTRE
                             //  IDENTIFICATEUR  CHIFFRE
         ENTIER-SANS-SIGNE  ::=  CHIFFRE
                             //  ENTIER-SANS-SIGNE  CHIFFRE
                    ENTIER  ::=  ENTIER-SANS-SIGNE
                             //  +  ENTIER-SANS-SIGNE
```

GRAMMAIRE ALGOL TRANSFORMEE IIa

6044,0192,0000 F. MARTIN

```
                              //
          PARTIE-DECIMALE    ::=
        FACTEUR-DE-CADRAGE   ::=
           NOMBRE-DECIMAL    ::=
                              //
                              //
        NOMBRE-SANS-SIGNE    ::=
                              //
                              //
           CHAINE-OUVERTE    ::=
                              //
                              //
                   CHAINE    ::=
               EXPRESSION    ::=
                              //
                              //
IDENTIFICATEUR-DE-VARIABLE   ::=
          VARIABLE-SIMPLE    ::=
      EXPRESSION-EN-INDICE   ::=
          LISTE-D'INDICES    ::=
                              //
 IDENTIFICATEUR-DE-TABLEAU   ::=
         VARIABLE-INDICEE    ::=
                 VARIABLE    ::=
                              //
IDENTIFICATEUR-DE-PROCEDURE  ::=
       PARAMETRE-EFFECTIF    ::=
                              //
                              //
                              //
                              //
        CHAINE-DE-LETTRES    ::=
                              //
  DELIMITEUR-DE-PARAMETRES   ::=
                              //
LISTE-DE-PARAMETRES-EFFECTIFS ::=
                              //
PARTIE-PARAMETRE-EFFECTIF    ::=
   INDICATEUR-DE-FONCTION    ::=
                              //
       CPERATEUR-ADDITIF     ::=
                              //
 CPERATEUR-MULTIPLICATIF     ::=
                              //
                              //
   PRIMAIRE-ARITHMETIQUE     ::=
                              //
                              //
                              //
                 FACTEUR     ::=
                              //
                   TERME     ::=
                              //
EXPRESSION-ARITHMETIQUE-SIMPLE ::=
                              //
```

```
-  ENTIER-SANS-SIGNE
.  ENTIER-SANS-SIGNE
'10'  ENTIER
ENTIER-SANS-SIGNE
PARTIE-DECIMALE
ENTIER-SANS-SIGNE  PARTIE-DECIMALE
NOMBRE-DECIMAL
FACTEUR-DE-CADRAGE
NOMBRE-DECIMAL  FACTEUR-DE-CADRAGE
CHAINE-PROPRE
'('  CHAINE-OUVERTE  ')'
CHAINE-OUVERTE  CHAINE-OUVERTE
'('  CHAINE-OUVERTE  ')'
EXPRESSION-ARITHMETIQUE
EXPRESSION-BOOLEENNE
EXPRESSION-DE-DESIGNATION
IDENTIFICATEUR
IDENTIFICATEUR-DE-VARIABLE
EXPRESSION-ARITHMETIQUE
EXPRESSION-EN-INDICE
LISTE-D'INDICES  ,  EXPRESSION-EN-INDICE
IDENTIFICATEUR
IDENTIFICATEUR-DE-TABLEAU  .(  LISTE-D'INDICES  ).
VARIABLE-SIMPLE
VARIABLE-INDICEE
IDENTIFICATEUR
CHAINE
EXPRESSION
IDENTIFICATEUR-DE-TABLEAU
IDENTIFICATEUR-D'AIGUILLAGE
IDENTIFICATEUR-DE-PROCEDURE
LETTRE
CHAINE-DE-LETTRES  LETTRE
,
)  CHAINE-DE-LETTRES  :  (
PARAMETRE-EFFECTIF
LISTE-DE-PARAMETRES-EFFECTIFS  DELIMITEUR-DE-PARAMETRES  PARAMETRE-EFFECTI
(  LISTE-DE-PARAMETRES-EFFECTIFS  )
IDENTIFICATEUR-DE-PROCEDURE  PARTIE-PARAMETRE-EFFECTIF
IDENTIFICATEUR-DE-PROCEDURE
+
-
*
/
'/'
NOMBRE-SANS-SIGNE
VARIABLE
INDICATEUR-DE-FONCTION
(  EXPRESSION-ARITHMETIQUE  )
PRIMAIRE-ARITHMETIQUE
FACTEUR  **  PRIMAIRE-ARITHMETIQUE
FACTEUR
TERME  OPERATEUR-MULTIPLICATIF  FACTEUR
TERME
OPERATEUR-ADDITIF  TERME
```

GRAMMAIRE ALGOL TRANSFORMEE IIIa

6044,0192,0000 F. MARTIN

```
                                                    //
                                PROPOSITION-SI  ::=
                            EXPRESSION-ARITHMETIQUE  ::=
                                                    //
                                      RELATION  ::=
                                 PRIMAIRE-BOOLEEN  ::=
                                                    //
                                                    //
                                                    //
                                                    //
                                SECONDAIRE-BOOLEEN  ::=
                                                    //
                                   FACTEUR-BOOLEEN  ::=
                                                    //
                                     TERME-BOOLEEN  ::=
                                                    //
                                      IMPLICATION  ::=
                                                    //
                          EXPRESSION-BOOLEENNE-SIMPLE  ::=
                                                    //
                                 EXPRESSION-BOOLEENNE  ::=
                                                    //
                                         ETIQUETTE  ::=
                                                    //
                        IDENTIFICATEUR-D'AIGUILLAGE  ::=
                            INDICATEUR-D'AIGUILLAGE  ::=
                      EXPRESSION-DE-DESIGNATION-SIMPLE  ::=
                                                    //
                                                    //
                               EXPRESSION-DE-DESIGNATION  ::=
                                                    //
                      INSTRUCTION-NON-ETIQUETEE-DE-BASE  ::=
                                                    //
                                                    //
                                   INSTRUCTION-DE-BASE  ::=
                                                    //
                                                    //
                          INSTRUCTION-INCONDITIONNELLE  ::=
                                                    //
                                                    //
                                       INSTRUCTION  ::=
                                                    //
                                                    //
                              FIN-D'INSTRUCTION-COMPOSEE  ::=
                                                    //
                                                    //
                                                    //
                                     EN-TETE-DE-BLOC  ::=
                                                    //
                                  COMPOSEE-NON-ETIQUETEE  ::=
                                      BLOC-NON-ETIQUETE  ::=
                                  INSTRUCTION-COMPOSEE  ::=
                                                    //
                                              BLOC  ::=
                                                    //
```

```
(PRESSION-ARITHMETIQUE-SIMPLE  OPERATEUR-ADDITIF  TERME
IF'  EXPRESSION-BOOLEENNE  'THEN'
(PRESSION-ARITHMETIQUE-SIMPLE
(OPOSITION-SI  EXPRESSION-ARITHMETIQUE-SIMPLE  'ELSE'  EXPRESSION-ARITHMETIQU
(PRESSION-ARITHMETIQUE-SIMPLE  OPERATEUR-DE-RELATION  EXPRESSION-ARITHMETIQUE-
ALEUR-LOGIQUE
ARIABLE
NDICATEUR-DE-FONCTION
ELATION
  EXPRESSION-BOOLEENNE  )
RIMAIRE-BOOLEEN
NOT'  PRIMAIRE-BOOLEEN
ECONDAIRE-BOOLEEN
ACTEUR-BOOLEEN  'AND'  SECONDAIRE-BOOLEEN
ACTEUR-BOOLEEN
ERME-BOOLEEN  'OR'  FACTEUR-BOOLEEN
ERME-BOOLEEN
MPLICATION  'IMPL'  TERME-BOOLEEN
MPLICATION
(PRESSION-BOOLEENNE-SIMPLE  'IDENT'  IMPLICATION
(PRESSION-BOOLEENNE-SIMPLE
(OPOSITION-SI  EXPRESSION-BOOLEENNE-SIMPLE  'ELSE'  EXPRESSION-BOOLEENNE
DENTIFICATEUR
NTIER-SANS-SIGNE
DENTIFICATEUR
DENTIFICATEUR-D'AIGUILLAGE  .(  EXPRESSION-EN-INDICE  ).
'IQUETTE
NDICATEUR-D'AIGUILLAGE
  EXPRESSION-DE-DESIGNATION  )
(PRESSION-DE-DESIGNATION-SIMPLE
(OPOSITION-SI  EXPRESSION-DE-DESIGNATION-SIMPLE  'ELSE'  EXPRESSION-DE-DESIGN
NSTRUCTION-D'AFFECTATION
NSTRUCTION-ALLER-A
NSTRUCTION-PROCEDURE
NSTRUCTION-NON-ETIQUETEE-DE-BASE
'IQUETTE  :  INSTRUCTION-DE-BASE
'IQUETTE  :
NSTRUCTION-DE-BASE
NSTRUCTION-COMPOSEE
.OC
NSTRUCTION-INCONDITIONNELLE
NSTRUCTION-CONDITIONNELLE
NSTRUCTION-POUR
NSTRUCTION  'END'
END'
NSTRUCTION  ::  FIN-D'INSTRUCTION-COMPOSEE
  FIN-D'INSTRUCTION-COMPOSEE
EGIN'  DECLARATION
-TETE-DE-BLOC  ::  DECLARATION
EGIN'  FIN-D'INSTRUCTION-COMPOSEE
-TETE-DE-BLOC  ::  FIN-D'INSTRUCTION-COMPOSEE
MPOSEE-NON-ETIQUETEE
'IQUETTE  :  INSTRUCTION-COMPOSEE
.OC-NON-ETIQUETE
'IQUETTE  :  BLOC
```

GRAMMAIRE ALGOL TRANSFORMEE IVa

6044,0192,0000 F. MARTIN

```
                                PROGRAMME  ::=
                                           //
                                PARTIE-GAUCHE  ::=
                                           //
                        LISTE-DE-PARTIES-GAUCHES  ::=
                                           //
                        INSTRUCTION-D'AFFECTATION  ::=
                                           //
                                INSTRUCTION-ALLER-A  ::=
                                    INSTRUCTION-SI  ::=
                                           //
                        INSTRUCTION-CONDITIONNELLE  ::=
                                           //
                                           //
                                           //
                                           //
                    ELEMENT-D'UNE-LISTE-DE-POUR  ::=
                                           //
                                           //
                                LISTE-DE-POUR  ::=
                                           //
                                PROPOSITION-POUR  ::=
                                    INSTRUCTION-POUR  ::=
                                           //
                                           //
                        INSTRUCTION-PROCEDURE  ::=
                                           //
                                DECLARATION  ::=
                                           //
                                           //
                                           //
                                LISTE-DE-TYPE  ::=
                                           //
                                        TYPE  ::=
                                           //
                                           //
                            DECLARATION-DE-TYPE  ::=
                        TYPE-LOCAL-OU-REMANENT  ::=
                                           //
                                BORNE-INFERIEURE  ::=
                                BORNE-SUPERIEURE  ::=
                                 PAIRE-DE-BORNES  ::=
                        LISTE-DE-PAIRES-DE-BORNES  ::=
                                           //
                                SECTION-DE-TABLEAU  ::=
                                           //
                                LISTE-DE-TABLEAU  ::=
                                           //
                            DECLARATION-DE-TABLEAU  ::=
                                           //
                                LISTE-D'AIGUILLAGE  ::=
                                           //
                            DECLARATION-D'AIGUILLAGE  ::=
                                 PARAMETRE-FORMEL  ::=
                        LISTE-DE-PARAMETRES-FORMELS  ::=
```

```
)C
STRUCTION-COMPOSEE
RIABLE  :=
ENTIFICATEUR-DE-PROCEDURE  :=
RTIE-GAUCHE
STE-DE-PARTIES-GAUCHES  PARTIE-GAUCHE
STE-DE-PARTIES-GAUCHES  EXPRESSION-ARITHMETIQUE
STE-DE-PARTIES-GAUCHES  EXPRESSION-BCOLEENNE
OTO'  EXPRESSION-DE-DESIGNATION
OPOSITION-SI  INSTRUCTION-INCONDITIONNELLE
OPOSITION-SI
STRUCTION-SI
STRUCTION-SI  'ELSE'  INSTRUCTION
STRUCTION-SI  'ELSE'
OPOSITION-SI  INSTRUCTION-POUR
IQUETTE  :  INSTRUCTION-CONDITIONNELLE
PRESSION-ARITHMETIQUE
PRESSION-ARITHMETIQUE  'STEP'  EXPRESSION-ARITHMETIQUE  'UNTIL'  EXPRESSION-A
PRESSION-ARITHMETIQUE  'WHILE'  EXPRESSION-BCOLEENNE
EMENT-D'UNE-LISTE-DE-POUR
STE-DE-POUR  ,  ELEMENT-D'UNE-LISTE-DE-POUR
OR'  VARIABLE  :=  LISTE-DE-POUR  'DO'
OPOSITION-POUR  INSTRUCTION
OPOSITION-POUR
IQUETTE  :  INSTRUCTION-POUR
ENTIFICATEUR-DE-PROCEDURE  PARTIE-PARAMETRE-EFFECTIF
ENTIFICATEUR-DE-PROCEDURE
CLARATION-DE-TYPE
CLARATION-DE-TABLEAU
CLARATION-D'AIGUILLAGE
CLARATION-DE-PROCEDURE
RIABLE-SIMPLE
RIABLE-SIMPLE  ,  LISTE-DE-TYPE
EAL'
NTEGER'
OOLEAN'
'PE-LOCAL-OU-REMANENT  LISTE-DE-TYPE
'PE
WN'  TYPE
PRESSION-ARITHMETIQUE
PRESSION-ARITHMETIQUE
)RNE-INFERIEURE  :  BORNE-SUPERIEURE
IRE-DE-BORNES
STE-DE-PAIRES-DE-BORNES  ,  PAIRE-DE-BORNES
)ENTIFICATEUR-DE-TABLEAU  .(  LISTE-DE-PAIRES-DE-BORNES  ).
)ENTIFICATEUR-DE-TABLEAU  ,  SECTION-DE-TABLEAU
ECTION-DE-TABLEAU
ISTE-DE-TABLEAU  ,  SECTION-DE-TABLEAU
ARRAY'  LISTE-DE-TABLEAU
'PE-LOCAL-OU-REMANENT  'ARRAY'  LISTE-DE-TABLEAU
(PRESSION-DE-DESIGNATION
ISTE-D'AIGUILLAGE  ,  EXPRESSION-DE-DESIGNATION
SWITCH'  IDENTIFICATEUR-D'AIGUILLAGE  :=  LISTE-D'AIGUILLAGE
)ENTIFICATEUR
ARAMETRE-FORMEL
```

GRAMMAIRE ALGOL TRANSFORMEE Va

6044,0192,0000 F. MARTIN

```
                                                    //
PARTIE-PARAMETRE-FORMEL                      ::=
LISTE-D'IDENTIFICATEURS                      ::=
                                                    //
PARTIE-VALEUR                                ::=
SPECIFICATEUR                                ::=
                                                    //
                                                    //
                                                    //
                                                    //
                                                    //
                                                    //
                                                    //
PARTIE-SPECIFICATION                         ::=
                                                    //
                                                    //
TETE-DE-PROCEDURE                            ::=

                                                    //
                                                    //
                                                    //
                                                    //
                                                    //
                                                    //
                                                    //
CORPS-DE-PROCEDURE                           ::=
                                                    //
DECLARATION-DE-PROCEDURE                     ::=
                                                    //
                                                    //
                                                    //
```

```
LISTE-DE-PARAMETRES-FORMELS  DELIMITEUR-DE-PARAMETRES  PARAMETRE-FORMEL
(  LISTE-DE-PARAMETRES-FORMELS  )
IDENTIFICATEUR
LISTE-D'IDENTIFICATEURS  ,  IDENTIFICATEUR
'VALUE'  LISTE-D'IDENTIFICATEURS  ::
'STRING'
TYPE
'ARRAY'
TYPE  'ARRAY'
'LABEL'
'SWITCH'
'PROCEDURE'
TYPE  'PROCEDURE'
SPECIFICATEUR  LISTE-D'IDENTIFICATEURS  ::
PARTIE-SPECIFICATION  SPECIFICATEUR  LISTE-D'IDENTIFICATEURS  ::
SPECIFICATEUR  LISTE-D'IDENTIFICATEURS  ::
IDENTIFICATEUR-DE-PROCEDURE  PARTIE-PARAMETRE-FORMEL  ::  PARTIE-VALEUR
PARTIE-SPECIFICATION
IDENTIFICATEUR-DE-PROCEDURE  ::  PARTIE-VALEUR  PARTIE-SPECIFICATION
IDENTIFICATEUR-DE-PROCEDURE  PARTIE-PARAMETRE-FORMEL  ::  PARTIE-SPECIFICATI
IDENTIFICATEUR-DE-PROCEDURE  PARTIE-PARAMETRE-FORMEL  ::  PARTIE-VALEUR
IDENTIFICATEUR-DE-PROCEDURE  ::  PARTIE-SPECIFICATION
IDENTIFICATEUR-DE-PROCEDURE  ::  PARTIE-VALEUR
IDENTIFICATEUR-DE-PROCEDURE  PARTIE-PARAMETRE-FORMEL  ::
IDENTIFICATEUR-DE-PROCEDURE  ::
INSTRUCTION
CODE
'PROCEDURE'  TETE-DE-PROCEDURE  CORPS-DE-PROCEDURE
'PROCEDURE'  TETE-DE-PROCEDURE
TYPE  'PROCEDURE'  TETE-DE-PROCEDURE  CORPS-DE-PROCEDURE
TYPE  'PROCEDURE'  TETE-DE-PROCEDURE
```

6044,0192 F. MARTIN

GRAMMAIRE EQUIVALENTE SANS VIDE **(avec l'axiome NOMBRE)**

```
                    CHIFFRE  ::=  0
                             //   1
                             //   2
                             //   3
                             //   4
                             //   5
                             //   6
                             //   7
                             //   8
                             //   9
          ENTIER-SANS-SIGNE  ::=  CHIFFRE
                             //   ENTIER-SANS-SIGNE  CHIFFRE
                    ENTIER   ::=  ENTIER-SANS-SIGNE
                             //   +  ENTIER-SANS-SIGNE
                             //   -  ENTIER-SANS-SIGNE
         PARTIE-DECIMALE     ::=  .  ENTIER-SANS-SIGNE
       FACTEUR-DE-CADRAGE    ::=  '10'  ENTIER
          NOMBRE-DECIMAL     ::=  ENTIER-SANS-SIGNE
                             //   PARTIE-DECIMALE
                             //   ENTIER-SANS-SIGNE  PARTIE-DECIMALE
       NOMBRE-SANS-SIGNE     ::=  NOMBRE-DECIMAL
                             //   FACTEUR-DE-CADRAGE
                             //   NOMBRE-DECIMAL  FACTEUR-DE-CADRAGE
                    NOMBRE   ::=  NOMBRE-SANS-SIGNE
                             //   +  NOMBRE-SANS-SIGNE
                             //   -  NOMBRE-SANS-SIGNE
```

IV. CONCLUSIONS ET REMARQUES

1. Sur la partie théorique.

Les résultats présentés ici ne sont pas nouveaux. Sous une forme ou une autre, ils se trouvent dans la littérature, et notamment dans [3] . Cependant, nous avons choisi de les présenter sous une forme un peu particulière, la plus propre à justifier exactement les algorithmes programmés dans ce travail.

Les principales différences avec GINSBURG sont les suivantes :

a) Tout d'abord, nous avons donné une définition inhabituelle de la grammaire; au lieu de la définition classique de Chomsky, par un quadruplet, nous avons défini une grammaire par un couple (règles, axiome); les vocabulaires terminal et non-terminal étant définis après coup, comme "associés à l'ensemble des règles".

b) Pour la construction de la grammaire sans vides, nous nous sommes inspirés de la définition de Ginsburg. Cependant, celui-ci considère un seul type de "symboles vides", qui subsistent lors de la transformation. Nous en avons distingué deux types: les symboles "semi-vides", que nous traîtons comme Ginsburg; et les symboles "essentiellement vides", qui sont éliminés. Nous obtenons ainsi une grammaire sans parasites; alors que dans la grammaire transformée selon Ginsburg, les symboles essentiellement vides sont devenus parasites. (D'après nos définitions, certains d'entre eux seraient même considérés comme terminaux, et par conséquent les grammaires ne seraient plus équivalentes.)

c) Dans l'étude des diverses transformations d'une grammaire, nous avons renoncé à démontrer certaines propriétés théoriques, comme la conservation de la non-ambiguïté, pour nous attacher à des propriétés servant plus directement notre propos, telles que la conservation des relations calculées.

2. Sur les programmes.

1° Le choix du langage ALGOL pour réaliser de tels programmes peut ne pas sembler heureux. D'ailleurs, la description donnée, au chapitre II, de la représentation en machine de la grammaire et du dictionnaire, utilise (abusivement) le terme de "pointeur", ce qui suggère l'utilisation d'un langage de liste. Nous nous sommes interrogés sur l'opportunité de récrire tout ou partie de ces programmes en LISP, ou d'utiliser les procédures LISP incorporées à ALGOL ([16]) . Il en serait résulté des avantages, mais aussi des inconvénients.

a) En ce qui concerne les entrées/sorties, l'utilisation d'un langage fait pour les manipulations de caractères aurait évité l'écriture de procédures en code.

Cependant, LISP-ALGOL utilise les procédures INSYMBOL et OUTSYMBOL, et nous avons déjà dit qu'elles nous semblent trop lourdes, et inadéquates pour notre problème. De plus, les listes de propriétés des atomes occupent deux mots-machine par caractère, ce qui nous aurait obligé à limiter la longueur des symboles; alors que nos programmes supportent aisément le symbole INSTRUCTION-NON-ETIQUETEE-DE-BASE.

Quand à LISP, il exigerait que la grammaire "en clair" soit donnée sous forme de liste, et il la récrirait de même.

Dans les deux cas, on perdrait donc une certaine liberté dans la présentation des données et des résultats.

b) Si la grammaire était représentée en mémoire par une liste et non par un tableau, ses diverses transformations (et particulièrement celles qui consistent à supprimer des règles) seraient simples et rapides, alors qu'ALGOL oblige à déplacer tous les éléments du tableau. Sur ce point, l'utilisation de LISP ou LISP-ALGOL serait avantageuse.

c) En ALGOL, les symboles supprimés ne disparaissent en réalité que de la grammaire; ils restent présents dans le dictionnaire, où ils conservent leur code. Aucun "garbage collector" ne récupère ces parties du dictionnaire vers lesquelles ne pointe plus aucun élément de la grammaire. D'autre part, les codes des symboles servent d'indices aux tableaux booléens représentant les relations; il en résulte que dans ces tableaux aussi subsistent les éléments relatifs aux symboles supprimés. Par l'intermédiaire des disques, les programmes qui s'enchaînent se transmettent de l'un à l'autre des tableaux dont certaines zones n'ont plus de signification.

En LISP, les symboles supprimés disparaîtraient automatiquement. En revanche, les relations ne seraient plus représentées par des tableaux booléens, mais, par exemple, par des listes de couples de symboles. Le calcul des relations s'en trouverait légèrement alourdi; quant à l'utilisation ultérieure de ces relations, elle serait singulièrement plus compliquée qu'en ALGOL.

Quant à LISP-ALGOL, il permettrait à la fois la disparition des symboles supprimés, et l'utilisation de tableaux booléens. Cependant, cela nécessiterait d'affecter aux symboles des codes allant de 1 à N, pour servir d'indices aux tableaux; et les éléments de tableaux relatifs aux symboles supprimés seraient quand même conservés.

2° Comme nous l'avons dit au chapitre II, paragraphe 7, l'enchaînement des programmes, tels qu'ils se présentent actuellement, n'est pas assez souple. Il est regrettable que la tentative de réaliser une meilleure articulation, utilisant DIAMAG I, n'ait pas abouti; peut-être serait-il intéressant de reprendre ce problème pratique avec un système plus récent.

3. En général.

Le programme de lecture des règles s'est révélé d'une assez grande utilité: il permet à tout programme manipulant des grammaires C.F. d'entrer celles-ci "en clair" sur cartes.

Les programmes de "calcul" ne sauraient servir à des programmes "opérationnels" d'analyse syntaxique; ceux-ci en effet travaillent généralement sur des grammaires d'un type particulier (de précédence, LR(K), etc.), souvent construites "à la main", et qui ne nécessitent pas de transformation préalable.

Quand à l'éditeur, il est trop général pour être opérationnel.

Si nos programmes n'ont pas l'ambition d'être intégrés à un programme de compilation ou de traduction, il s'avèrent en revanche efficaces pour l'étude "expérimentale" d'un langage ou d'une grammaire. Nous entendons par là qu'un langage en voie de définition, ou un langage nouveau pour un utilisateur, peut être analysé d'abord par nos programmes, puis "expérimenté" par un programme simple d'analyse syntaxique s'y enchaînant.

Citons, comme exemples, les essais faits par G. DAGAND pour la définition de son langage LSD; et l'étude faite à Grenoble du projet de WIRTH et HOARE pour ALGOL ([8]), étude qui

a permis de découvrir, dans la définition des règles, quelques petites erreurs formelles, qui étaient passées inaperçues à la lecture. En revanche, nos programmes ne permettront pas l'étude de l'ALGOL 68 de van WIJNGAARDEN, car il n'est pas défini par une grammaire C.F.

Cette utilisation de nos programmes à des fins essentiellement expérimentales justifie bien le titre du travail : "Détermination de certaines caractéristiques des grammaires et langages C.F." .

BIBLIOGRAPHIE

[1] N. CHOMSKY. Formal properties of Grammars.
Handbook of Mathematical Psychology.
Vol 2. John Wiley and Sons. 1963.

[2] N. CHOMSKY-G.A. MILLER. L'analyse formelle des langues naturelles.
Gauthier-Villars. 1968.

[3] S. GINSBURG. The mathematical theory of context-free languages.
McGraw -Hill. 1966.

[4] M. GROSS-A.LENTIN. Notions sur les grammaires formelles.
Gauthier-Villars. 1967.

[5] P. NAUR (Ed.) Revised report on the Algorithmic language ALGOL 60.
Communications of the ACM. Vol.6/1 1963.

[6] Report on Input-Output Procedures for ALGOL 60.
Communications of the ACM. Vol.7/5 1964.

[7] L. BOLLIET-N. GASTINEL-P.J LAURENT.
Un nouveau langage scientifique: ALGOL
Hermann. 1964.

[8] N. WIRTH-C.A.R. HOARE.
A contribution to the development of ALGOL.
Communications oh the ACM. Vol.9/6 1966.

[9] S. WARSHALL. A theorem on Boolean matrices.
Journal of the ACM Vol.9/1 1962.

[10] L. BOLLIET. Notations et processus de traduction des langages symboliques.
Thèse - Université de Grenoble. 1967.

[11] M. BRASSEUR-J. COHEN.
Algorithmes d'analyse syntaxique pour langages "context-free".
Revue "Chiffres". Vol.8/2,3. 1965.

[12] J. COHEN. Langages pour l'écriture des compilateurs.
Thèse - Université de Grenoble. 1967.

[13] A. COLMERAUER. Notion d'opérateurs dans une grammaire "context-free".
Revue d'Informatique et de Recherche opérationnelle. Vol. 1/2 1967.

[14] A. COLMERAUER. Précédence, analyse syntaxique et langages de programmation.
Thèse - Université de Grenoble. 1967.

[15] A. COLMERAUER. Total precedence relations.
Journal of the ACM. 1967.

[16] J. COHEN- NGUYEN-HUU-DUNG.
Définition de procédures LISP en ALGOL.
Revue "Chiffres". Vol.8/4. 1965.

[17] NGUYEN-HUU-DUNG.
Aspects non numériques de la programmation ALGOL.
Thèse - Université de Grenoble. 1965.

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

[18] NGUYEN-DINH-XUAN.
Méthodes d'analyse syntaxique descendante pour langages "context-free".
Thèse - Université de Grenoble. 1966.

[19] L. TRILLING. Contribution à l'étude des mécanismes de traduction des langages de programmation.
Thèse - Université de Grenoble. 1967.

[20] L. COURTIN. Langages analysables de gauche à droite. Construction d'un analyseur pour langages LR (1).
Thèse - Université de Grenoble. 1968.

[21] F. MARTIN. Quelques algorithmes sur des grammaires "context-free".
Congrès AFIRO. Nancy. 1967.