UNIVERSITE DE NANCY FACULTE DES SCIENCES

ETUDE DE COGENT

LANGAGE DE TRAITEMENT DE STRUCTURES ARBORESCENTES

-=-=-=-=-

THESE

pour l'obtention du

DOCTORAT de SPECIALITE MATHEMATIQUES (3ème CYCLE)

Soutenue devant le Jury le 30 Novembre 1968

par

Hervé SCHIVI

Jury : Mr J. LEGRAS Président
Mr C. GILORMINI Examinateurs
Mr C. PAIR

UNIVERSITE DE NANCY FACULTE DES SCIENCES

ETUDE DE COGENT

LANGAGE DE TRAITEMENT DE STRUCTURES ARBORESCENTES

par Hervé SCHIVI

UNIVERSITE DE NANCY — FACULTE DES SCIENCES

Doyen : M.AUBRY

Assesseur : M.GAY

Doyens Honoraires: MM. CORNUBERT - DELSARTRE - ROUBAULT.

Professeurs Honoraires : MM. RAYBAUD - LAFFITTE - LERAY - JULY - LAPORTE - EICHHORN - GODEMENT - DUBREUIL - L.SCHWARTZ - DIEUDONNE - de MALLEMANN - LONGCHAMBON - LETORT - DODE - CAUTHIER - GOUDET - OLMER - CORNUBERT - CHAPELLE - GUERIN - WAHL.

Maîtres de conférences honoraires : MM. LIENHART - PIERRET

Professeurs

MM.	DELSARTE	Analyse supérieure	SUHNER	Physique expérimentale
	ROUBAULT	Géologie	CHAPON	Chimie biologique
	CAPELLE	Mécanique rationnelle	HEROLD	Chimie minérale industrielle
	VEILLET	Biologie animale	SCHWARTZ B.	Exploitation minière
	BARRIOL	Chimie théorique	*MALAPRADE	Chimie
	BIZETTE	Physique	MANGENOT	Botanique
	GUILLIEN	Electronique	GAYET	Physiologie
	GIBERT	Chimie physique	HADNI	Physique
	LEGRAS	Mécanique rationnelle	*BASTICK	Chimie
	BOLFA	Minéralogie	DUCHAUFOUR	Pédologie
	NICLAUSE	Chimie	GARNIER	Agronomie
	FAIVRE	Physique appliquée	NEEL	Chimie organique industrielle
	AUBRY	Chimie minérale	BERNARD	Géologie appliquée
	COPPENS	Radiogéologie	*CHAMPIER	Physique
	DUVAL	Chimie	*GAY	Chimie biologique
	FRUHLING	Physique	STEPHAN	Zoologie
	HILLY	Géologie	*CONDE	Zoologie
	LE GOFF	Génie chimique	*WERNER	Botanique

* Professeur titulaire à titre personnel)

Mr.	EYMARD	Calcul différentiel intég.	N....	Chimie biologique
	LEVISALLES	Chimie organique	N....	Mécanique appliquée
Mme	HERVE	Méthodes mathématiques de la physique		
	FELDEN	Physique		
	*GOSSE	Mécanique physique		
	*DAVOINE	Physique(E N S M I M)		
	HORN	Physique 1er cycle		
*Mme	LUMER	Mathématiques		

Maîtres de conférences

Mme	BASTICK	Chimie P.C. Epinal	JACQUIN	Pédologie et Chimie agri
Mr.	GUDEFIN	Physique	LAINARD	Physique M.F.
	ROCCI	Géologie	CACHAN	Entomologie appliquée(EI
	VUILLAUME	Psychophysiologie	MARTIN	Chimie P.C.
	FRENTZ	Biologie animale	PAULMIER	Mécanique expérimentale
	MARI	Chimie (I S I N)	PROTAS	Minéralogie
	AUROUZE	Géologie	JOZEFOWICZ	Physico Chimie
	DEVIOT	Physique du solide	JURAIN	Géologie C B G
	FLECHON	Physique P.C.	RIVAIL	Chimie appliquée (Mines)
Mlle	HUET	Mathématiques C B G	VILLERMAUX	Génie chimique
	VIGNES	Métallurgie	METCHE	Biochimie appliquée(Bras
	BALESDENT	Thermodynamique chimique appliquée (ENSIC)	PAIR	Mathématiques appliquées
			BAUMANN	Physique 1er cycle
	BLASY	Minéralogie appliquée(ENSG)	DURAND	Physique
	JANOT	Physique PC Epinal	HUSSON	Physique (I.S.I.N.)
	N...	Mécanique des fluides (ISIN)	N...	Mathématiques P.C.
			N...	Mathématiques C.B.G.
	N...	Mécanique ISIN	N...	Physiologie animale
	N...	Probabilité et statisti.	N...	Mathématiques M.P.
	N...	Mathématiques	N...	Exploitation minière (ENSMIN)

* Professeur titulaire à titre personnel)

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

Ce travail a été effectué sous la direction de Monsieur PAIR a qui j'exprime mes plus vifs remerciements, pour la bienveillante attention qu'il n'a cessé de me témoigner, et, pour la formation qu'il m'a donnée et que je dois à son aide constante.

J'assure de ma reconnaissance Monsieur GILORMINI qui a accepté de participer au Jury.

Que le personnel de l'I. U. C. A. trouve ici l'expression de mes remerciements pour l'aide qu'il m'a apportée dans la réalisation de ce travail.

Cette étude a fait l'objet d'une convention de la Délégation Générale de la Recherche Scientifique et Technique. (D. G. R. S. T.)

Convention n° 66 00 230

SOMMAIRE

COGENT 1.2 (COmpiler GENeralized Translator) est un langage de traitement de structures arborescentes crée en 1965 par John C. REYNOLDS sous la dénomination COGENT.

Son premier but est de décrire des compilateurs, mais il peut aussi être utilisé à des problèmes de :

- traitement algébrique
- démonstration de théorème
- programmation heuristique.

Ce langage fut entendu en 1966 et on se propose ici de décrire en la formalisant, cette dernière version datant de 1966. Cette étude présente COGENT à partir des ouvrages de Reynolds dont on trouvera les références en appendice, mais toutes les restrictions résultants de l'implémentation de COGENT sur CONTROL DATA 3600 on été éliminées.

On présente une formalisation des informations traitées par COGENT permettant de les traiter algébriquement et une formalisation des opérations de base, permettant de les définir plus précisément. On trouvera [PAIR C - QUERE A] une étude formelle détailléée de la définition des informations traitées, principalement, des structures arborescentes et une définition des opérations sur ces structures. On arrive ainsi à une définition plus rigoureuse de COGENT.

Toutes les définitions ont été rassemblées dans le chapitre 1 et sont ainsi séparées de la définition du langage.

Le chapitre 2 définit la syntaxe et la sémantique du langage.

Le chapitre 3 étudie une réalisation d'un système analogue à l'aide du langage A. T. F. [L. NOLIN].

CHAPITRE I

INTRODUCTION - DEFINITIONS

1.1 INTRODUCTION

Les informations traitées par COGENT sont des mots sur certains alphabets, des ramifications (paragraphe 2.2) et des grammaires de Chomsky (paragraphe 2.3).

COGENT apparait comme un métalangage qui décrit d'autres langages que l'on appellera langages "traités" et qui sont engendrés par une grammaire de Chomsky. La structure des phrases de ces langages est décrite par des pseudo-arborescences (D. 1.2.2.1) qui seront le type essentiel d'information traitée par COGENT. Dans chaque programme les traitements sont définis par des procédures dont les paramètres et le résultat sont des pseudo-arborescences.

b) Un programme COGENT (paragraphe 2.2) se compose de trois parties :

- Description de caractères qui permet de préciser l'alphabet des langages traités (paragraphe 2.3).

- Section des grammaires qui décrit les grammaires des langages traités (paragraphe 2.4).

- Description des procédures qui définit les procédures utilisées pour le traitement des informations (paragraphe 2.7).

c) L'exécution d'un programme procède en deux phases essentielles :

- la première phase est une opération d'analyse ; étant donné un mot α appartient-il au langage traité "d'entrée" ; si oui trouver ses structures et la seconde phase est éxecutée sinon le programme est interrompu ;

- dans la seconde phase les opérations définies par le programme sont éxécutées par des procédures, associées aux régles de la grammaire définissant le langage d'entrée.

d) Les opérations fondamentales utilisées par COGENT sont décrite paragraphe 1.3.

Elles permettent d'effectuer trois sortes d'instructions d'affectatio (paragraphe 2.6) qui peuvent éventuellement échouer. Ce sont ces échecs permettent le contrôle du déroulement du programme par des instructio de saut conditionnel (paragraphe 2.6.5).

1.2 DEFINITIONS

1.1.1 Monoïde libre

D.1.2.1.4 MONOIDE

On appelle monoïde un ensemble M muni d'une loi de composition interne associative et possédant un élément neutre.

D.1.2.1.2 MONOIDE LIBRE SUR UN ENSEMBLE V

Soit un ensemble V et V^* l'ensemble des suites finies d'éléments de V (y compris la suite vide notée Λ) :

$$V^* = \left\{ a_1 a_2 \dots a_k \setminus a_i \in V \text{ pour } 1 \leqslant i \leqslant k \right\} \cup \{ \Lambda \}$$

V^* est muni de la loi de composition interne concaténation telle que :

$$(a_1 a_2 \dots a_p, b_1 b_2 \dots b_q) \longmapsto a_1 a_2 \dots a_p \; b_1 b_2 \dots b_q .$$

V^* est un monoïde, appelé monoïde libre sur V.

Définitions

- Les suites d'éléments de V sont appelées mots sur V, ou chaînes sur V.
- Soit un mot $\alpha = a_1 \dots a_k$: k est la longueur de α notée $|\alpha|$. Le mot vide a pour longueur O.
- Soient trois mots α, β, γ tels que $\alpha = \beta \gamma$ on dit que β est facteur gauche et γ facteur droit de α.

1.2.2 RAMIFICATIONS

Toutes les définitions qui suivent sont extraites d'un article de Monsieur PAIR cité en appendice sous la référence PAIR, C. On reprendra ici les points essentiels de cet article.

GRAPHE [BERGE]

On appelle graphe un couple (E, Γ) ou E est un ensemble et Γ une relation binaire dans E.

On notera $\Gamma(x)$ l'ensemble des $y \in E$ tels que $x \Gamma y$.

D.1.2.2.0 ARBORESCENCE [BERGE]

On appelle arborescence un graphe fini sans circuit tel que

a) il existe un point a qui n'est extrêmité d'aucun arc.

b) tout point $x \neq a$ est l'extrémité d'un arc unique, a s'appelle la racine de l'arborescence.

ORIENTATION D'UNE ARBORESCENCE

Une orientation d'une arborescence (E, Γ) est un ordre partiel $\emptyset$ dans l'ensemble E tel que

a) les restrictions de $\emptyset$ à chacun des ensembles $\Gamma(x)$ $(x \in E)$ sont des ordres totaux ;

b) Si $y \in \Gamma(x)$ et $z \notin \Gamma(x)$, y et z ne sont pas comparables par la relation $\emptyset$.

Un triplet $(E, \Gamma, \emptyset)$ où (E, Γ) est une arborescence et $\emptyset$ une de ses orientations s'appelle arborescence orientée.

Soit deux arborescences orientées $(E, \Gamma, \emptyset)$ et $(E', \Gamma', \emptyset')$; un isomorphisme de la première sur la deuxième est une bijection h de E sur E' telle que :

$$(\forall x, y \in E) \left[(x \Gamma y \iff h(x) \Gamma' h(y)) \text{ et } (x \emptyset y \iff h(x) \emptyset' h(y)) \right].$$

D.1.2.2.1 PSEUDO-ARBORESCENCE SUR UN ENSEMBLE V

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

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

Une pseudo-arborescence sur V est une classe de cette équivalence. Elle pourra être représentée par l'un de ses couples (I, f). On appelle ordre de cette pseudo-arborescence le nombre de points de l'arborescence I ; pour chaque $A \in V$, on appelle nombre d'occurrences de A dans la pseudo-arborescence le nombre de points x de I tels que $f(x) = A$.

Une pseudo-arborescence d'ordre 1 sera identifiée à l'unique élément de V qui y possède une occurence.

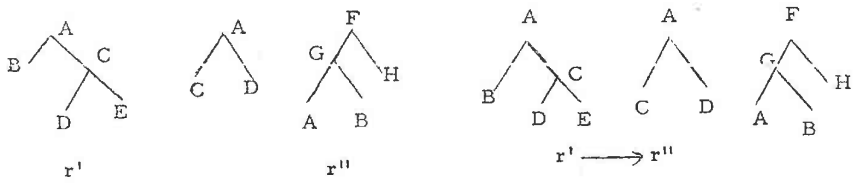
L'ensemble des pseudo-arborescences sur V sera noté $a(V)$.

D.1.2.2.2. RAMIFICATION

On appelle ramification sur V toute suite finie de pseudo-arborescences sur V. L'ensemble des ramifications sur V c'est-à-dire le monoïde libre déduit de l'ensemble des pseudo-arborescences sur V sera noté $\hat{V}$. La loi de composition de ce monoïde sera eppelée produit et notée $\rightarrow$: intuitivement $r \rightarrow s$ est obtenue en plaçant la ramification s à droite de la ramification r. L'élément neutre de cette loi sera nommée ramification vide et notée Λ.

Exemple

La figure ci-contre schématise deux ramifications r' et r'' et leur produit $r' \rightarrow r''$.

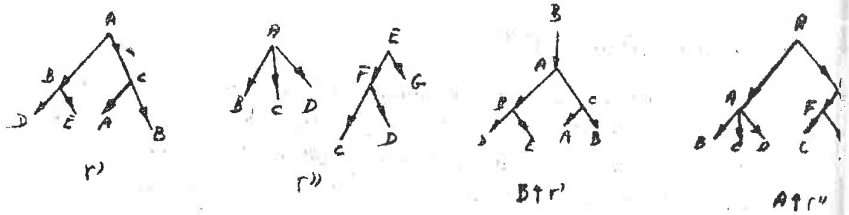


D.1.2.2.3 ENRACINEMENT

Ce sera une loi de composition externe à opérateurs dans un ensemble V, notée ↑ avec opérateurs à gauche : intuitivement la ramification $A \uparrow r$ est la pseudo-arborescence obtenue en adjoignant à une racine d'étiquette A.

Exemple

La figure ci-contre schématise deux ramifications r' et r'' et présente $B \uparrow r'$ et $A \uparrow r''$.



D.1.2.2.4 PRINCIPE DE RECURRENCE DANS V.

Proposition 1

Pour toute pseudo-arborescence s sur V, il existe un élément A de V et un seul, une ramification r sur V et une seule tels que $s = A \uparrow r$.

De la définition des ramifications comme suite de pseudo-arborescences et de la proposition 1, il résulte :

Proposition 2

Pour tout $r \in \hat{V}$, non vide, il existe $A \in V$, $r' \in \hat{V}$, $r'' \in \hat{V}$ uniques tels que $r = (A \uparrow r') \rightarrow r''$

On en déduit un principe de récurrence dans $\hat{V}$:

Principe de récurrence

Soit $\mathcal{P}$ un prédicat tel que :

a - $\mathcal{P}(\Lambda)$ soit vrai

b - $(\forall a \in \mathcal{A}(V),\ r \in \hat{V})\ (\mathcal{P}(a)$ et $\mathcal{P}(r) \Longrightarrow \mathcal{P}(a \rightarrow r))$

c - $(\forall r \in \hat{V})\ (\forall A \in V)\ (\mathcal{P}(r) \Longrightarrow \mathcal{P}(A \uparrow r))$;

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

A partir de ce principe on démontrerait [PAIR C. p. 10] en particulier que : toute ramification sur V est une combinaison finie par $\rightarrow$ et $\uparrow$, d'éléments de V.

Les applications de $\hat{V}$ dans un ensemble E seront commodément définies par récurrence. Plus précisément on démontre [PAIR C.] :

Théorème

Soient un ensemble E, un élément e de E, deux applications f_1, de $\hat{V} \times \mathcal{A}(V) \times E^2$ dans E et f_2 de $V \times \hat{V} \times E$ dans E. Il existe une application λ et une seule de $\hat{V}$ dans E, telle que :

a) $\lambda(\Lambda) = e$

b) $(\forall r \in \hat{V})\ (\forall a \in \mathcal{A}(V))\ [\lambda(a \rightarrow r) = f_1(r, a, \lambda(r), \lambda(a))]$

c) $(\forall A \in V)\ (\forall r \in \hat{V})\ [\lambda(A \uparrow r) = f_2(A, r, \lambda(r))]$

D.1.2.2.5 MOT des RACINES d'une RAMIFICATION

Appelons mot des racines de $r \in \hat{V}$ le mot $\rho(r)$ sur le vocabulaire V défini par

$\rho(\Lambda) = \Lambda$;

$\rho(r \rightarrow s) = \rho(r) \rightarrow \rho(s)$;

$\rho(A \uparrow r) = A$.

Si le mot des racines de r est $A_1 \ldots A_n$, A_j est appelé la $j^{\text{ième}}$ racine de r.

D. 1. 2. 2. 6 MOT des FEUILLES d'une RAMIFICATION

Appelons mot des feuilles de r le mot $\varphi(r)$ sur V, tel que

$\varphi(\Lambda) = \Lambda$;

$\varphi(r \rightarrow s) = \varphi(r) \rightarrow \varphi(s)$;

$\varphi(A \uparrow r) =$ si $\varphi(r) \neq \Lambda$ alors $\varphi(r)$ sinon A.

Si le mot des de r est $A_1 \ldots A_n$, on dit que A_j est la $j^{ième}$ feuille d

D. 1. 2. 2. 7 FAMILLES d'une RAMIFICATION

Pour la ramification $r' \rightarrow r''$ de D. 1. 2. 2. 2. nous dirons qu le mot C D est une famille de prédécesseur A, que le mot vide est une f mille de prédécesseur D.

Soit A un élément de V et F_A l'application de $\hat{V}$ dans l'ensemble $P(V^*)$ des parties de V^* définie par :

$F_A(\Lambda) = \emptyset$ $F_A(r \rightarrow s) = F_A(r) \cup F_A(s)$

$F_A(B \uparrow r) =$ si $B = A$ Alors $F_A(r) \cup \{\rho(r)\}$ sinon $F_A(r)$.

Par définition $F_A(r)$ est l'ensemble des familles de prédecesseur A dan

1. 2. 3. GRAMMAIRE ET LANGAGES DE CHOMSKY

1. 2. 3. 1 Grammaire de Chomsky

Une grammaire de Chomsky G est un triplet comprenant

- un ensemble fini T appelé vocabulaire terminal (ses éléments sont app lés symboles terminaux ou symbole de base),
- un ensemble fini N disjoint de T vocabulaire non terminal (ses élément sont des symboles non-terminaux) ; on pose V = TUN (vocabulaire de la grammaire ou alphabet),
- une relation binaire, notée ::= entre N et V^* appelée relation de produc tion telle que :
  - le nombre des couples en relations est fini.
  - $(\forall A, \varphi \in V^*)$ $(A ::= \varphi \Longrightarrow A \in N$ et $\varphi \in V^*)$.

G est notée G = (T, N, ::=)

Un couple de la relation de production est appelé règle de la grammaire G. Le nombre des règles de la grammaire est supposé fini.

1. 2. 3. 2. Ramification engendrée - Dérivation

Une ramification sur V est engendrée au sens large par la grammaire G lorsque chacune de ses familles α de prédécesseur A vérifie $A ::= \alpha$ (notons que le mot α peut être le mot vide Λ, A est alors une feuille de la ramification).

Soit X un élément de N, α un mot sur V, α dérive de X lorsqu'il existe dans $\mathcal{A}(V)$ une pseudo-arborescence dont α est le mot des feuilles et X la racine.

1. 2. 3. 3. Langage engendré

Choisissons dans N un élément X : l'ensemble des mots de T^* qui dérivent de X pour la grammaire G s'appelle langage engendré par le couple (G, X). L'élément X est souvent nommé axiome.

1. 2. 3. 4. Analyse

Soit une grammaire G et un non-terminal X. Etant donné un mot α sur le vocabulaire de G analyser α pour le couple (G, X) c'est trouver les pseudo-arborescences engendrées par la grammaire G dont α est mot des feuilles.

S'il en existe plusieurs, la grammaire G est ambigüe pour l'axiome X.

1. 2. 3. 5. Notation de BACKUS [ALGOL]

Dans la suite les élément du vocabulaire non-terminal seront représentés par des séquences de caractères enfermés dans les crochets < > .

Les éléments du vocabulaire terminal seront des caractères ne figurant pas entre crochets.

Par commodité d'écriture plusieurs règles qui ont même premie membre : $A ::= \varphi_1 \cdots A ::= \varphi_n$ pourront former une règle composée qu s'écrira $A ::= \varphi_1 | \cdots | \varphi_n$.

On dira que les n règles $A ::= \varphi_1, \cdots, A ::= \varphi_n$ sont les règles simples déduites de cette règles composée. (paragraphe 2.4.4.3-a)

Exemple : Grammaire des polynômes.

49 règles simples

1 à 26 < 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

27 à 36 < CHIFFRES > ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

37-38-39 <CHAINE> ::= <LETTRE> | <CHAINE> <LETTRE> | <CHAINE> <CHIFFI

40 < VARIABLE > ::= < CHAINE >

41 - 42 < FACTEUR > ::= < VARIABLE > | (< POLYNOME >)

43 - 44 < TERME > ::= < FACTEUR > | < TERME > * < FACTEUR >

45 à 49 < POLYNOME > ::= < TERME > | + < TERME > | - < TERME
< POLYNOME > + < TERME > |
< POLYNOME > - < TERME >

Cette grammaire avec l'axiome < POLYNOME > engendre les polyno mes à plusieurs variables au sens habituel. On utilisera cette grammair dans plusieurs exemples figurant dans ce travail. Les règles simples d' grammaire traitée par un programme COGENT seront supposées numéro tées comme elles le sont ci-dessus.

1.3 OPERATIONS FONDAMENTALES DE COGENT

1.3.1 Analyse d'un mot pour une grammaire d'un axiome

Déjà définie paragraphe 1.2.3.4 son résultat est un ensemble de pseudo-arborescences sur le vocabulaire (D.1.2.3.1) de G.

En réalité, une telle pseudo-arborescence est complétement déterminée si on remplace tout prédécesseur A d'une famille φ (D.1.2.2.7) par le numéro repérant la règle $A ::= \varphi$. On peut même alors enlever les feuilles (D.1.2.2.6.) de la pseudo-arborescence. On transformera donc les pseudo-arborescences résultat de l'analyse par l'application θ définie récursivement par :

$\theta(\Lambda) = \Lambda$

$\theta(r \rightarrow s) = \theta(r) \rightarrow \theta(s)$

$\theta(A \uparrow r) = \Lambda$ si $r = \Lambda$

$= n \uparrow \theta(r)$ si $r \neq \Lambda$ et n est le numéro de la règle $A ::= \rho(r)$.

Soit une grammaire G, T son vocabulaire terminal et X un axiome.

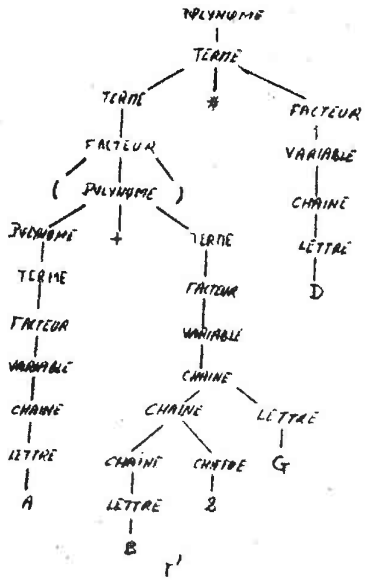
Etant donné un mot α sur T, en transformant par θ, les pseudo-arborescences résultant de l'analyse de α pour G et X on obtient un ensemble de pseudo-arborescences sur l'ensemble des numéros de règle de G, que l'on appellera l'analyse simplifiée de α pour G et X.

Exemple 1 - Analyse d'un mot sur T

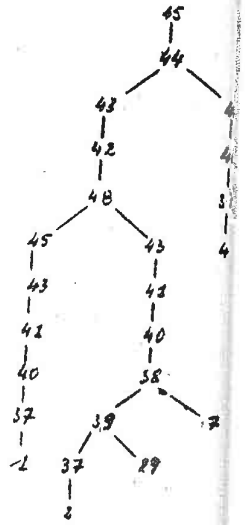
Données

- Grammaire paragraphe 1.2.2
- Axiome < POLYNOME >
- mot (A + B2 G) * D

Résultat



r'



r

La pseudo-arborescence de droite est obtenue par application de la transformation θ à la pseudo-arborescence de gauche.

1.3.2 Synthèse de deux ramifications

Cette opération est définie dans l'ensemble des ramifications (D.1.2.2.2) sur un vocabulaire contenant des entiers. La donnée de cette opération est formée :

- d'une ramification r'
- d'une seconde ramification r' c'est-à-dire d'une suite de p pseudo-arborescences $r'_1, r'_2, \ldots, r'_p$.

La synthèse substituera r'_i à tout entier i du mot des feuilles de r. (D.1.2.2.6).

Plus précisement la synthèse de r et de r' est définie par récurrence sur r, r' étant supposée fixée.

$$S(\Lambda) = \Lambda$$

$$S(r \rightarrow s) = S(r) \rightarrow S(s)$$

$$S(A \rightarrow r) = r'_i \quad \text{si } r = \Lambda \text{ et } A = i \leqslant p$$

$$= A \uparrow S(r) \quad \text{sinon.}$$

Propriété :

Si $r'_i \neq r''_i$ et si i possède une occurrence dans $\varphi(r)$, la synthèse de r et r' est différente de celle de r et r'', ce résultat se démontre immédiatement par récurrence sur r.

Exemple

Données

- la ramification r,
- la suite des pseudo-arborescences r'_1, r'_2, r'_3.

Remarque :

r'', r''_1, r''_2, r''_3 sont les pseudo-arborescences résultats de l'analyse des mots < FACTEUR/2 > * < FACTEUR/1 > * < FACTEUR/3 > AB, (< VARIABLE/1 > + D), (< FACTEUR/3 > * < FACTEUR/1 >) pour la grammaire du paragraphe 1. 2. 3. 5, et les axiomes respectifs < TERME >, < FACTEUR >, < FACTEUR >.

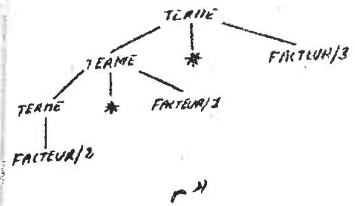
r, r'_1, r'_2, r'_3 sont les résultats de la transformation θ appliquée respectivement à r'', r''_1, r''_2, r''_3.

Résultat

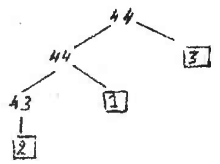
La pseudo-arborescence s.

Remarque

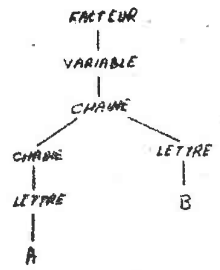
La pseudo-arborescence s'' est donnée pour mémoire, elle est le résultat de la synthèse de r'' et de la suite de pseudo-arborescence r''_1, r''_2, r''_3, toutes ces pseudo-arborescences étant définies sur le vocabulaire de la grammaire.



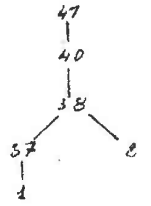
r''



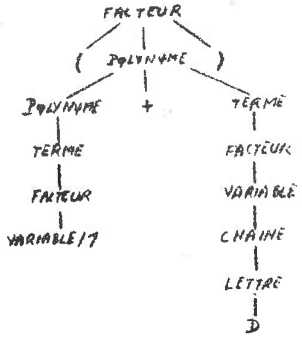
r



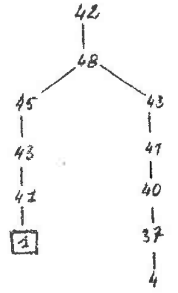
r''_1



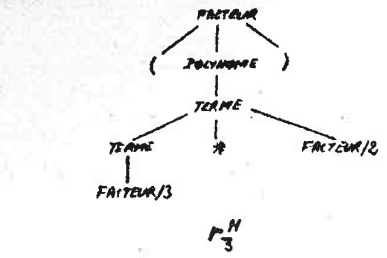
r'_1



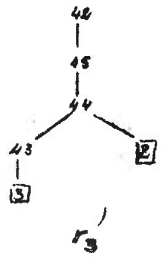
r''_2



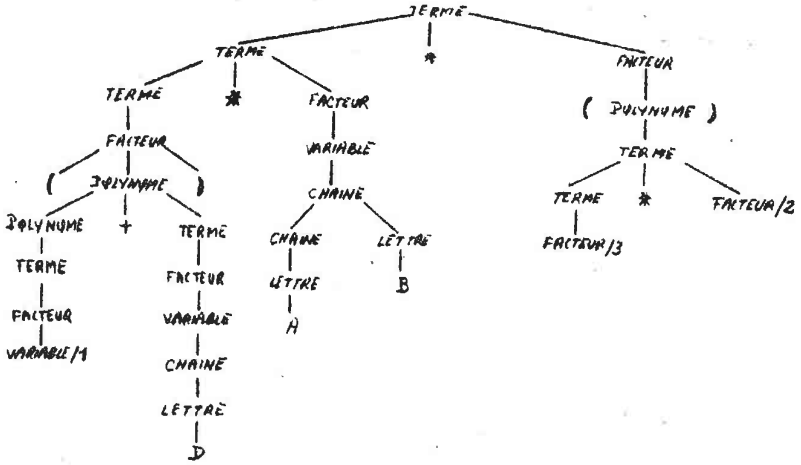
r'_2



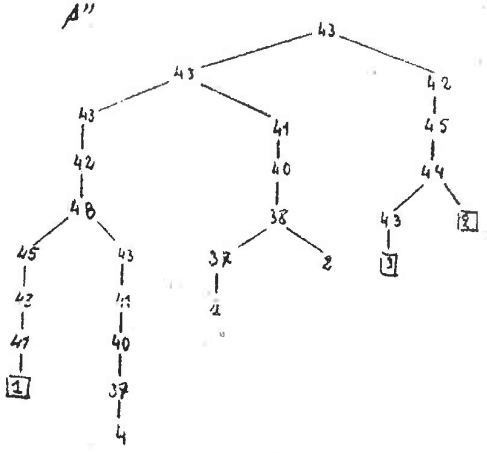
r_3''



r_3'



A''



A

1.3.3 Analyse d'une ramification r par une ramification s

Cette opération est inverse de la synthèse. Comme la synthèse elle est définie dans l'ensemble des ramifications sur un vocabulaire et des entiers, mais seulement pour certains couples de ramifications.

Soit une ramification s telle que les entiers contenus dans son mot des feuilles soient 1, 2, ..., p ($p \geq 0$), chacun une fois et une seule, et d'autre part une ramification r. Si r et la synthèse de s et d'une ramification t formée de p pseudo-arborescence, t est unique d'après la propriété indiquée au paragraphe 1.3.2, t est alors le résultat de l'analyse de r par s. Dans les autres cas l'analyse de r par s n'est pas définie.

Exemple

On se reporte à l'exemple du paragraphe 1.3.2

Donnée

- la ramification r
- la ramification s

Résultat

Le résultat de l'analyse de s par r sera la ramification $r2 \rightarrow r1 \rightarrow r3$.

Remarque

Si p = 0 c'est-à-dire si le mot des feuilles de s ne contient aucun entier, l'analyse de r par s est définie si et seulement si r = s ; le résultat est la ramification vide (D. 1.2.2.2).

CHAPITRE II

SYNTAXE et SEMANTIQUE DU LANGAGE

2.1 ELEMENTS DE BASE DU LANGAGE

2.1.1 Symbole de base

< SYMBOLE DE BASE > ::= < CARACTERE NORMAL > | < CARACTERE SPECIAL >

< CARACTERE NORMAL > ::= < LETTRE > | < CHIFFRE > | = | + | - | * | / | $

< 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 > ::= < CHIFFRE OCTAL > | 8 | 9 |

< CHIFFRE OCTAL > ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7

< CARACTERE SPECIAL > ::= . | , | (|)

2.1.2 Commentaires

2.1.2.1 Syntaxe

< COMMENTAIRE > ::= $ COMMENT < LIBELLE > .|

$ TITLE < LIBELLE >.

< LIBELLE > ::= < VIDE > |

< LIBELLE > <CARACTERE>

< VIDE > :: =

< CARACTERE > ::= < CARACTERE NORMAL > | (|) | ,

2.1.2.2 Sémantique

Un commentaire est sans effet sur l'éxecution du programme, s'il commence par $ TITLE, il apparait dans la liste du programme imprimé à la compilation.

S'il commence par $ COMMENTil n'apparait pas dans la liste.

2.1.3. Identificateur

2.1.3.1 Syntaxe

< IDENTIFICATEUR > ::= < LETTRE > | < IDENTIFICATEUR > < LETTRE > |
< IDENTIFICATEUR > < CHIFFRE >

2.1.3.2 Sémantique

Un identificateur peut être un identificateur de variab
un identificateur de procédure, un identificateur de constante, selon l
type de sa déclaration (paragraphe 2.7).

2.1.4 Nombre

< ENTIER > ::= < CHIFFRE > | < ENTIER > < CHIFFRE >

< ENTIER OCTAL > ::= < CHIFFRE OCTAL > |
< ENTIER OCTAL > < CHIFFRE OCTAL >.

2.2 PROGRAMME

< PROGRAMME COGENT > ::= < DESCRIPTION DE CARACTERES >
< SECTION DES GRAMMAIRES > < SECTION DES PROCEDURES > |
< COMMENTAIRE > < PROGRAMME COGENT >.

La description de caractères définit l'alphabet des langages étudiés et la section des grammaires, leur grammaire. La section des procédures définit des fonctions dont la suite des arguments est une suite de pseudo-arborescences, c'est-à-dire une ramification (D.1.2.2.2) sur un vocabulaire défini au paragraphe 2.4.4.3 b, et dont le résultat est une pseudo-arborescence sur ce même vocabulaire.

L'exécution d'un programme est définie au paragraphe 2.4.4.3.e.

2.3 DESCRIPTION DE CARACTERES

2.3.1 Syntaxe

< SYMBOLE TRAITE > ::= < CARACTERE NORMAL > | (< CARACTERE SEPCIAL >) |
($ < IDENTIFICATEUR >)

< DESCRIPTION DE CARACTERES > ::= < VIDE > | $ CHARDEF
< SUITE DE DEFINITIONS DE CARACTERES >

< SUITE DE DEFINITIONS DE CARACTERES > ::= < VIDE > |
< SUITE DE DEFINITIONS DE CARACTERES > < DEFINITION DE CARAC
< SUITE DE DEFINITIONS DE CARACTERES > < COMMENTAIRE >

< DEFINITION DE CARACTERE > ::= < SYMBOLE TRAITE > =
< SUITE DE CODES D ENTREE > < SUITE DE CODES DE SAUT > <CODE

< SUITE DE CODES D ENTREE > ::= (< SUITE DE CODES >)

< SUITE DE CODES DE SAUT > ::= < VIDE > |
(< SUITE DE CODES >)

< SUITE DE CODES > ::= < ENTIER OCTAL > |
< SUITE DE CODES > , < ENTIER OCTAL >

< CODE DE SORTIE > ::= < ENTIER OCTAL >

2.3.2 Exemples

(1) $ CHARDEF A = (221) 222. (.) = (223) 224. ($EOF) = (101) 101.

(2) $ CHARDEF ($FP) = (120, 121) (122, 123) 106.

(3) $ CHARDEF ($AB) = (116, 117) (118, 119, 130) 105.
* (201) (302, 102, 101) 107. (|) = (402, 403, 404) (405) 1

2.3.3 Sémantique

Un symbole traité est utilisé dans la descr
de caractères, la définition des règles (paragraphe 2.4.1), les prima
(§ 2.5.2.1) pour représenter un caractère d'un langage traité. Les s
de l'alphabet d'un langage traité (D. 1.2.3.3) peuvent être des caract
normaux du langage COGENT, mais pas des caractères spéciaux. En
les quatre caractères spéciaux sont

ceux qui apparaissent dans la description des règles de grammaire (paragraphe 2.4.1) et il y aurait donc risque de confusion. Si on désire employer un caractère spécial dans un langage traité on le mettra entre parenthèses.

Lorsqu'un symbole du langage traité n'est pas un caractère de COGENT, on le désigne par un identificateur précédé d'un $ et mis entre parenthèse

Pour un tel symbole on doit préciser les codes externes qui peuvent le représenter à l'entrée et à la sortie pour cela on utilise une description de caractère.

On peut aussi utiliser la description de caractère pour un symbole de base, dans ce cas la description remplacera la description standard.

La description de caractère comprend :

- la suite des codes d'entrée qui désigne le ou les codes pouvant représenter le caractère défini dans la zone d'entrée.
- la suite des codes de saut qui précise le ou les codes à ignorer quand ils suivent un des codes d'entrée.
- le code de sortie (qui est unique) qui est utilisé pour représenter le caractère correspondant dans la zone de sortie et aussi dans la représentation interne des mots.

2.4 SECTION DES GRAMMAIRES

2.4.1 Règles

2.4.1.1 Syntaxe

< REGLE > ::= <SUITE D'ETIQUETTES> <PRIORITE> < REGLE DE GRAMMAIRE >

< REGLE DE GRAMMAIRE > ::= < NON TERMINAL > = < SECOND MEMBRE >.

< NON TERMINAL > ::= (< IDENTIFICATEUR >)

< SECOND MEMBRE > ::= < MOT > | < SECOND MEMBRE > , < MOT >

< MOT > ::= < VIDE > | < MOT > < SYMBOLE TRAITE > |

< MOT > < NON TERMINAL >

< SUITE D'ETIQUETTES > ::= < VIDE > | < ETIQUETTE SPECIALE >

< ETIQUETTE > < SUITE D'ETIQUETTES >

< ETIQUETTE > ::= < IDENTIFICATEUR > /

< ETIQUETTE SPECIALE > ::= $ IDENT, < ENTIER > /|

$ OCT /| $ DEC /| $ FLOAT/| $ NOP /| $ NOTRAN /

< PRIORITE > ::= < VIDE > | *

2.4.1.2 Exemples

PROCEMULT/ (TERME) = (TERME) * (FACTEUR).

DIFFERENTIATION/ (PRIMAIRE) = $ D (( ) (VARIABLE) (,) (EXPRESSION) ( )

DEPART / (SEQUENCE DE PHRASES) = .

PROC1/ PROC2 / PROC3 / (A) = (B) (C) (D), (A) (B) (D).

(TERME) = (TERME) * (FACTEUR) ; (TERME) / (FACTOR).

PROCADD / $ NOTRAN / (POLYNOME) = (TERME)

$ IDENT, 14 / (VARIABLE) = (CHAINE).

* (CODE BCD) = BCD.

2.4.1.3 Sémantique

a) Chaque règle de grammaire représente la règle en notation de Backus (1.2.3.5) obtenue :

- en supprimant le . (destiné ici à séparer les règles),
- en remplaçant = par ::=

(par <

) par >

, par |

Exemple

La règle représentée en GOGENT par

(SUITE DE PHRASES) = (PHRASE) (.) , (SUITE DE PHRASES) (PHRASE) (.).

s'écrira en notation de BACKUS

<SUITE DE PHRASES> ::= <PHRASE> . <SUITE DE PHRASES> < PHRASE >.

b) Etiquettes

Les étiquettes normales sont des identificateurs de fonction du programme ou des identificateurs de variable dont la valeur est une fonction (paragraphe 2.7.3.3). Le paragraphe 2.4.4.3.d associera une fonction aux étiquettes spéciales.

Les étiquettes d'une règle, y compris l'étiquette spéciale si elle existe, déterminent ainsi les procédures qui sont appelées lors de l'exécution du programme (paragraphe 2.4.4.3.e), à chaque rencontre de la règle. L'appel des procédures se fait en suivant l'ordre de droite à gauche des étiquettes, c'est-à-dire l'ordre habituel en notation fonctionnelle : autrement dit la suite d'étiquettes désigne une procédure obtenue par composition fonctionnelle de procédures désignées par les étiquettes de la suite.

La première procédure exécutée (correspondant à la dernière étiquette) a pour arguments des pseudo-arborescences associées aux divers symboles non terminaux (D.1.2.3.1) du second membre ; chacune des autres procédures a pour argument le résultat de la procédure exécutée avant elle ; seule la première procédure peut donc posséder plusieurs arguments.

En l'absence d'étiquette spéciale le nombre de paramètres de la procédure associée à la première étiquette doit être égal au nombre de symboles non terminaux du second membre.

c) Priorité

Une règle contenant un signe * est de préférence prise en c
ration en cas d'ambiguité (voir paragraphe 2.4.4.3 f).

2.4.2 Grammaire Primaire

2.4.2.1 Syntaxe

< GRAMMAIRE PRIMAIRE > ::= $ PRIMSYN (< SUITE D'AXIOMES >)
< SUITE DE REGLES >

< SUITE D'AXIOMES > ::= < IDENTIFIEUR D'AXIOME > |
< SUITE D'AXIOMES > , < IDENTIFIEUR D'AXIOME >

< IDENTIFIEUR D'AXIOME > ::= < NON TERMINAL > < SUITE DE TERMINAISONS

< SUITE DE TERMINAISONS > ::= < TERMINAISON > |
< SUITE DE TERMINAISONS > < TERMINAISON >

< TERMINAISON > ::= < SYMBOLE TRAITE >

< SUITE DE REGLES > ::= < VIDE > | < SUITE DE REGLES > < REGLES > |
< SUITE DE REGLES > < COMMENTAIRE >

2.4.2.2 Exemples

```
$PRIMSYN ((PHRASE) (.))
    (VAR) = A, B, C, D, E, F.
    (OPM) = *, / .
    (EXP) = (VAR).
    (EXP) = (OPM) (EXP) (EXP).
    (SEQ EXP) = (EXP) , (SEQ EXP) (,) (EXP).
    (PHRASE) = (SEQ EXP).
```

2.4.2.3 Sémantique

La grammaire (D.1.2.3.1) primaire ne peut être omise.

Elle décrit le langage traité d'entrée et contrôle l'analyse des chaines lues.

A chaque instant de l'éxécution d'un programme (paragraphe 2.4.4.3 e) un des axiomes (D.1.2.3.3) au plus est dit actif : au départ, il s'agit du premier axiome ; l'axiome de rang n est rendu actif par l'éxécution de l'appel de fonction SETIVGOAL (n), c'est pour l'axiome actif que se fait l'analyse d'une partie de la chaîne lue. Cette partie de chaine est limitée par une des terminaisons associées à l'axiome actif (paragraphe 2.4.4.3.e). L'appel de la fonction SETIVGOAL (0) rend inactif tous les axiomes, ce qui empêche toute nouvelle analyse.

2.4.3 Grammaire Secondaire

2.3.3.1 Syntaxe

< GRAMMAIRE SECONDAIRE > ::= $ SECSYN < SUITE DE REGLES >

2.3.3.2 Sémantique

La grammaire secondaire est utilisée avec la grammaire primaire pour analyser certaines chaines (paragraphe 2.4.4.3.c).

2.4.4 Section des Grammaires

2.4.4.1 Syntaxe

< SECTION DES GRAMMAIRES > ::= < GRAMMAIRE PRIMAIRE > |
< GRAMMAIRE PRIMAIRE > < GRAMMAIRE SECONDAIRE >

2.4.4.2 Exemple

```
    $ CHARDEF ($EOF) = (101) 101.
    $ PRIMSYN ((PHRASE) ($EOF)
          (VARIABLE) = A, B, C, D, E, F, G.
          (OPM) = *, /.
          (OPA) = +, -.
          (OPU) = =.
PVAR/     (EXPO) = (VARIABLE).
PMULT/    (EXPO) = (OPM) (EXPO).
PADD/     (EXPO) = (OPA) (EXPO) (EXPO).
       (SEQ EXPO) = (EXPO), (SEQ EXPO) (,) (EXPO).
SORTIE / (PHRASE) = (SEQ EXPO)
    $ SECSYN
          (FACTEUR) = (VARIABLE), ( ( ) (EXPR) ( ) ).
          (TERME) = (FACTEUR) , (TERME) (OPM) (FACTEUR).
          (EXPR) = (TERME) , - (TERME) , (EXPR) (OPA) (TERM
```

2.4.4.3 Sémantique

a) Traitement des règles

Chaque règle de la grammaire primaire ou de la grammaire condaire est décomposée en plusieurs règles simples (D. 1. 2. 3. 5) a toutes pour premier membre et étiquettes ceux de la règle considé pour second membre les parties, séparées par des virgules dans le membre de la règle considérée.

Exemple

```
PROCADD / $ NOTRAN / (POLY) = (POLY)+(TERME),
                                (POLY) - (TERME).
```

sera décomposée en trois règles simples ci-contre :

```
PROCADD / $ NOTRAN / (POLY) = (POLY) + (TER).
PROCADD / $ NOTRAN / (POLY) = (POLY) - (TER).
```

Les règles simples sont numérotées, deux règles différentes ayant des numéros différents.

b) Alphabet sur lequel sont construites les pseudo-arborescences traitées.

Les informations traitées par un programme COGENT sont des ramifications (et en particulier le plus souvent des pseudo-arborescences) sur l'alphabet formé :

(1) des numéros de règles simples de la grammaire primaire ;

(2) des numéros de règles simples de la grammaire secondaire. ;

Les éléments de type (1) ou (2) seront dits éléments grammaticaux ;

(3) des entiers ou éléments indexés ;

(4) des chaînes représentation des nombres entiers ou réels en numérotation décimale ou octale ou éléments numériques ;

(5) des couples formés par un numéro de table et une chaîne de symboles terminaux ou éléments caténaires ; un tel couple repère une entrée dans une table où pourra être rangée une pseudo-arborescence (paragraphe 2. 7. 3. 4) ; cette information sera explicitée par des fonctions standard (paragraphe 3. 5. 2)

(6) des procédures du programme ou éléments procédure

Cet alphabet sera nommé alphabet fondamental.

En particulier, on emploiera la ramification vide et des pseudo-arobrescences d'ordre 1 (D. 1. 2. 2. 1) c'est à dire réduite à un élément d'un des types précédents.

Les éléments de type (3), (4), (5), (6) ne pourront être que des feuilles (D. 1. 2. 2. 6) des ramifications traitées.

c) Analyse des chaines

L'analyse simplifiée (paragraphe 1.3.1) d'une chaîne lue sur organe d'entrée formée de symboles terminaux (D.1.2.3.1), est effe pour les règles de la grammaire primaire et un de ses axiomes. Son tat est une pseudo-arborescence sur l'alphabet fondamental réduit à s éléments de type (1).

L'analyse d'une chaine formée de symboles terminaux ou non te naux, pour les règles des grammaires primaire et secondaire et u symbole non terminal quelconque pourra être aussi effectuée pour obt la valeur d'un primaire (paragraphe 2.5.2).
Cette valeur sera un pseudo-arborescence sur l'alphabet fondamental.

Dans les deux cas si dans la pseudo-arborescence obtenue, N es prédécesseur d'une famille $P_1 \dots P_k$

- N est le numéro d'une règle $A ::= B_1 \dots B_{k'}$
- P_i est un numéro de règle de premier membre B_i ou bien P_i e un élément indexé.

Les éléments de type (4) et (5) sont introduits dans les pseudo-ar rescences par les transformations qu'effectuent les fonctions standard $ OCT, $ DEC, $ FLOAT, $ IDENT.

d) Transformation associée à la règle simple de numéro n

Au paragraphe 2.4.1.3.b, nous avons vu qu'une fonction est a ciée à chaque étiquette d'une règle.

A $ NOTRAN on associe la fonction f_n dépendant du numéro n de règle simple considérée et définie par

$$f_n(r) = n \uparrow r$$

pour toute ramification r.

Les autres étiquettes spéciales sont des fonctions standard. $ N a pour argument une pseudo-arborescence unique et

$$\$\ NOP\ (r) = r.$$

Pour toute ramification r et tout entier p $ IDENT, p / (r) est l' ment caténaire (paragraphe 2.4.4.3.b) formé du numéro de table p et mot des feuilles $\varphi(r)$ de r.

$ DEC, $ FLOAT, $ OCT transforment r en un élément numérique (paragraphe 2.4.4.3.b) représentant le mot des feuilles de r (D.1.2.2.6).

Ces fonctions seront examinées en détail aux paragraphes 2.9 et 2.9.

On obtient ainsi une suite de fonctions $g_1, \dots, g_p$ associées aux p étiquettes. La transformation associée à la règle de numéro n est

$$T_n = g_1 \circ \dots \circ g_p$$

g_p a pour arguments une suite de pseudo-arborescences, c'est-à-dire une ramification ; il en est donc de même pour T_n. Les autres fonctions doivent avoir pour argument une seule pseudo-arborescence puisque le résultat d'une fonction est toujours une pseudo-arborescence.

Si la règle de numéro n ne contient aucune étiquette, on lui associe la transformation T_n définie par

$$T_n(S) = n \uparrow s.$$

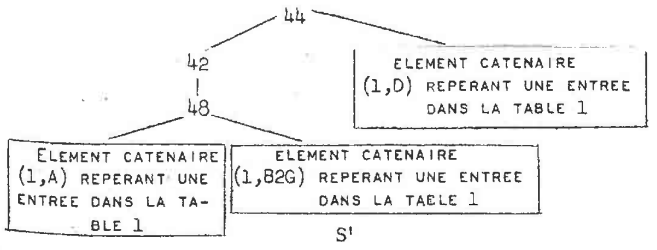
Exemple d'utilisation d'étiquettes spéciales

Considérons la grammaire du paragraphe 1.2.3.5 dans laquelle nous remplaçons respectivement les règles (40), (41), (43), (45) par les règles étiquetées :

40	$ IDENT, 1	(VARIABLE) = (CHAINE).
41	$ NOP /	(FACTEUR) = (VARIABLE).
43	$ NOP /	(TERME) = (FACTEUR).
45	$ NOP /	(POLYNOME) = (TERME).

On obtiendra la grammaire G'.

Le résultat de l'analyse du mot (A+B2G) * D pour le couple (G', (POLYNOME)) sera la pseudo-arborescence s' ci-contre que l'on comparera à la pseudo-arborescence r du paragraphe 1.3.1.



S'

e) Exécution d'un programme COGENT

Les transformations T_n associées aux règles permettent de d
récursivement une transformation Φ de l'ensemble des ramifications
l'alphabet fondamental (paragraphe 2.4.4.3.b)

$$\Phi(\Lambda) = \Lambda$$
$$\Phi(r \rightarrow s) = \Phi(r) \rightarrow \Phi(s)$$
$$\Phi(n \uparrow r) = T_n\left[\Phi(r)\right] \quad \text{si } n \text{ est un numéro de règle}$$
$$= n \uparrow \Phi(r) \quad \text{sinon}$$

Un programme COGENT est exécuté de la manière suivante :

a) Le plus petit facteur gauche (D.1.2.1.2) de la chaine à lire sur un
ne d'entrée, qui est suivi par une des terminaisons associées à l'axion
tif (paragraphe 2.4.2.3.) est lu : on en effectue l'analyse simplifiée (p
graphe 1.3.1) pour la grammaire primaire et cet axiome actif.

b) Si l'analyse ne conduit pas à exactement une pseudo-arborescence il
erreur. Sinon la pseudo-arborescence obtenue est transformée par Φ.

c) S'il existe un axiome actif, on reprend l'étape a, sinon l'éxécution s
rête.

Remarques

1 - Lors de la transformation effectuée en b, les procédures éxéc
peuvent avoir des effets annexes, par exemple : impression de résulta

2 - en réalité, les étapes a et b peuvent être effectuées simultané
l'analyse déterminant dans un ordre convenable les diverses parties de
pseudo-arborescence cherchée.

Cette simultanéité n'est cependant possible que si la procédure d'
lyse ne rencontre pas plusieurs possibilités.

Exemple

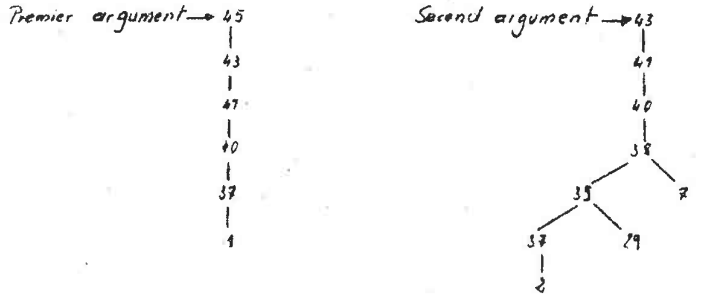
Considérons la grammaire G obtenue en remplaçant respectivem
les règles (44) et (48) de la grammaire du paragraphe 1.2.3.5 par les

(44) PROC MULT / (TERME) = (TERME) * (FACTEUR)

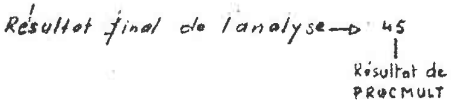
(48) PROC ADD / (POLYNOME) = (POLYNOME) + (TERME)

Les ramifications ci-dessous précisent : les arguments des procédures
OCMULT et PROCADD, l'ordre d'appel de ces procédures (1), (2) et le ré-
ltat final de l'analyse de la chaîne (A+B2G) * D pour la grammaire G
n comparera ce résultat à la ramification r du paragraphe 1.3.1)

(1) Arguments de la procédure PROC ADD



(2) Arguments de la procédure PROC MULT



f) Traitement des ambiguités

Si lors de l'analyse d'une chaine d'entrée (paragraphe 2.3.4.3.c plusieurs possibilités se présentent, chacune d'elles engendrera une anal séparée, qui pourra à son tour engendrer des analyses séparées ou être abandonnée.

Lorsque pendant une anlyse on utilise une règle précédée d'un asté risque toutes les autres analyses en cours sont abandonnées. Il est clai le traitement des ambiguités dépend de la procédure d'analyse utilisée.

Exemple

Considérons la grammaire formée des règles :

(CODE OPERATION) = (LETTRE), (CODE OPERATION) (LETTRE),
(CODE OPERATION) (CHIFFRE) .

* (CODE BCD) = BCD.

(INSTRUCTION) = (CODE OPERATION) (ZONE ADRESSE).

(INSTRUCTION) = (CODE BCD) (CHAINE DE CARACTERES).

"BCD 097" répond à chacune des définitions de (INSTRUCTION) : l'analy lui donnera la seconde interprétation (CODE BCD) (CHAINE DE CARAC- TERES), puisque BCD sera analysé comme (CODE BCD) et non comme (CODE OPERATION) la règle correspondante * (CODE BCD) marquée d astérisque étant prioritaire dans l'orientation du choix.

2.5 EXPRESSION

2.5.1

2.5.1.1. Syntaxe

< EXPRESSION > ::= < ENTIER > | < IDENTIFICATEUR > | < PRIMAIRE > |
< APPEL DE FONCTION >

2.5.1.2 Sémantique

L'évaluation d'une expression lui associe une valeur qui est une pseudo-arborescence sur l'alphabet fondamental défini au paragraphe 2.4.4.3.b.

Si l'expression est un entier, sa valeur est la pseudo-arborescence réduite à cet entier.

Si une expression est un identificateur sa valeur est la valeur possédée par cet identificateur au moment de l'évaluation.

Si une expression est un appel de fonction sa valeur est la pseudo-arborescence qui est la valeur de l'appel de fonction ; de plus dans ce cas l'expression peut échouer.

2.5.2 Primaire

2.5.2.1 Syntaxe

< PRIMAIRE > ::= (< NON TERMINAL > / < CHAINE >)

< CHAINE > ::= < VIDE > |
< CHAINE > < SYMBOLE TRAITE > |
< CHAINE > < NON TERMINAL > |
< CHAINE > (< NON TERMINAL > / < ENTIER >)

2.5.2.2 Exemples

(FACTEUR)
(FACTEUR/4)
(FACTEUR/2) ⋆ AB ± (FACTEUR)

(FACTEUR/ AB ⋆ D + C)
(TERME/ (FACTEUR/2) ⋆ AB ⋆- (FACTEUR))

2.5.2.3 Sémantique

Une chaîne est un mot (D. 1.2.1.2) sur la réunion du vocabul terminal et du vocabulaire non terminal d'une grammaire traitée (D. 1.

Un primaire (X/α) a pour valeur la pseudo-arborescence détermi

a) en remplaçant tout non terminal B de α non suivi d'un entier p B/i où i est le rang de ce non terminal parmi les symboles non termina de α.

b) en analysant la chaine α pour les règles des grammaires prima et secondaire et le non-terminal X ;
un non terminal indexé (B/i) par un entier joue lors de l'analyse le mê rôle que ce non-terminal non-indexé ;

c) en transformant la pseudo-arborescence obtenue, supposée uni par la transformation θ' définie récursivement par :
(on comparera θ' à θ : paragraphe 1.3.1).

$$\theta'(\Lambda) = \Lambda$$
$$\theta'(r \rightarrow s) = \theta'(r) \rightarrow \theta'(s)$$
$$\theta'(A \uparrow r) = \Lambda \quad \text{si } r = \Lambda \text{ et A est un symbole traité}$$
$$= i \quad \text{si } r = \Lambda \text{ et A est un non-terminal B/i d'in}$$
$$= n \uparrow \theta'(r) \quad \text{si } r \neq \Lambda \text{ et n est le numéro de la règle A}$$

d) en transformant la pseudo-arborescence obtenue par la transfo tion ξ (paragraphe 2.3.4.3) définie par les seules étiquettes spéciales règles de grammaire.

L'index est utilisé pour fixer l'ordre des substitutions lors de l'opération de synthèse, l'ordre des affectations lors de l'opération d'analyse.

La valeur d'un primaire peut être déterminée dès la compilation.

Si lors de l'analyse de la chaine on trouve plusieurs pseudo-arborescences (cas d'une grammaire ambiguë), un message d'erreur est imprimé ; la compilation continue cependant après avoir choisi arbitrairement l'une des possibilités.

Exemples

EX1

cf. paragraphe 2.5.2.2.

EX2

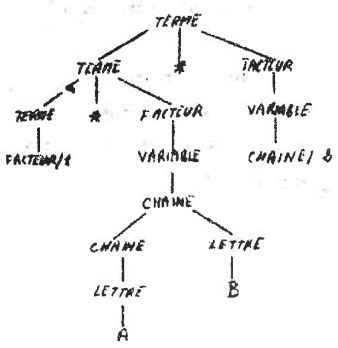
Analyse d'un Mot sur V

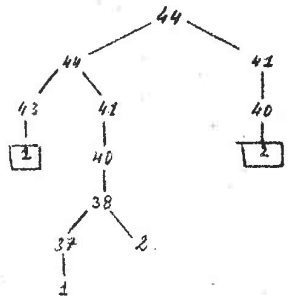
Données

- Grammaire paragraphe 1.2.2
- Axiome < TERME >
- Mot < FACTEUR/1 > ⋆ AB ⋆ < CHAINE/2 >

< FACTEUR/1 > et < CHAINE/2 > associe respectivement à < FACTEUR > et < CHAINE > les entiers 1 et 2.

Résultat





La pseudo-arborescence de droite est obtenue par application de la transformation θ' à la pseudo-arborescence de gauche.

Les entiers encadrés sont associés à des symboles non terminaux de à analyser.

2.5.3 Appel de Fonction

2.5.3.1 Syntaxe

< APPEL DE FONCTION > ::= < EXPRESSION > (< ARGUMENTS >)

< ARGUMENTS > ::= < VIDE > | < ARGUMENTS > , < EXPRESSION >.

2.5.3.2 Exemples

```
PARENTHESAGE (X, 1)
PARENTHESAGE (DIFF1 (Y), 3)
SOMME (QUOTIENT (DIFF1 (Y), Z), NEGATIF (QUOTIENT
    (PRODUIT (DIFF (Z), Y), PUISSANCE (Z, 2)))
EXEMPLE (X) (Y)
OUTP ( )
$RETURN (0).
```


2.5.3.3 Sémantique

Un appel de fonction demande le traitement des arguments par une fonction procédure précisée par l'expression de tête.

L'appel de fonction est évalué comme suit :

- l'expression de tête et les arguments sont évalués.

La valeur de l'expression de tête doit être une pseudo-arborescence d'ordre 1 réduite à une procédure dont le nombre des paramètres formels est égal au nombre des arguments, leur affectation est déterminée par une correspondance ordonnée avec les arguments. Un argument vide désigne une procédure sans arguments.

- la procédure est exécutée,
- le résultat de la procédure, qui est une pseudo-arborescence sur l'alphabet fondamental défini paragraphe 2.4.4.3.b, est la valeur de l'appel de fonction.

Un appel de fonction échoue (paragraphe 1.1.d) dans les trois cas suivants :

- l'évaluation de l'expression de tête ou d'un argument échoue.
- une des instructions de la fonction (non contenue dans une instruction de saut échoue).
- certaines fonctions standard peuvent échouer dans certains cas qui seront précisés lors de leur définition (paragraphe 2.9 en particulier $ FAILURE (paragraphe 2.9.9.1) échoue toujours.

Lors de l'appel de fonction dès qu'un échec apparaît dans une étape de l'appel, l'étape suivante n'est pas exécutée et l'appel de fonction échoue.

2.6 INSTRUCTION

2.6.1 Syntaxe

< INSTRUCTION > ::= < INSTRUCTION D'AFFECTATION > |
< INSTRUCTION DE SAUT > | < APPEL DE FONCTION >. |
< INSTRUCTION > < COMMENTAIRE >

< INSTRUCTION D'AFFECTATION > ::= < AFFECTATION SIMPLE > |
< AFFECTATION DE SYNTHESE > | < AFFECTATION D'ANALYSE >

< ETIQUETTE D'INSTRUCTION > ::= < NUMERO D'INSTRUCTION > / |
< ETIQUETTE D'INSTRUCTION > < COMMENTAIRE >

< NUMERO D'INSTRUCTION > ::= < ENTIER >

2.6.2. Sémantique

Une instruction du type < APPEL DE FONCTION > . est exécutée évaluant l'appel de fonction comme défini paragraphe 2.5.3.3. mais en rant son résultat, en cas d'échec de l'appel de fonction l'instruction éch Cette forme est utile pour ses résultats annexes.

Une instruction d'affectation peut également échouer.

Une instruction de saut peut exploiter un échec, mais n'échoue jama

Une étiquette d'instruction repère l'instruction qui la suit, elle doit être unique dans la plus petite procédure qui la contient.

Le numéro d'instruction est un entier différent de zéro.

2.6.3 Affectation Simple

2.6.3.1 Syntaxe

< AFFECTATION SIMPLE > ::= < IDENTIFICATEUR > = < EXPRESSION

2.6.3.2 Exemples

X = (FACTEUR / DC).
Y2 = PROCADD (Y, (TERME / AB * BC)).
DVAR = X

2.6.3.3 Sémantique

L'affectation simple est exécutée en évaluant l'expression et en affectant sa valeur à l'identificateur qui se trouve à gauche du signe = .

Si l'évaluation de l'expression échoue (voir paragraphe 2.5.3.3) l'instruction échoue sans modifier la valeur de la variable.

2.6.4 Affectation de synthèse

2.6.4.1 Syntaxe

< INSTRUCTION D'AFFECTATION DE SYNTHESE > ::=
< IDENTIFICATEUR > /=< EXPRESSION MODELE > < LISTE DE SYNTHESE >.

< EXPRESSION MODELE > ::= < EXPRESSION >

< LISTE DE SYNTHESE > ::= < VIDE > | < LISTE DE SYNTHESE >,
< ELEMENT DE SYNTHESE >

< ELEMENT DE SYNTHESE > ::= < VIDE > | < EXPRESSION >

2.6.4.2. Exemples

T1 / = (PRIMAIRE / COS (( ) (SEQ EXP) ( ))), Y.
X / = (TERME / (EXP) * (EXP)), X, Y.
XY / = (PHRASE / (EXP) (.)), PARENTHESAGE (X).
VAR/ - (POLYNOME/ (POLYNOME) - (TERME)), SOMME (X, Y), X.

2.6.4.3 Sémantique

L'instruction d'affectation de synthèse est effectuée comme suit :

a - L'expression modèle et les expressions de la liste de synthèse évaluées ; il y a erreur si une feuille (D.1.2.2.6) de la valeur de l'expr modèle est un élément indexé supérieur au nombre d'éléments de la list synthèse.

b - l'opération de synthèse (définie paragraphe 1.3.2) est effectué avec comme opérandes.

- la pseudo-arborescence qui est valeur de l'expression modèle ;
- la suite des pseudo-arborescences valeur des éléments de la list synthèse ; si l'un de ces éléments est vide, on lui donne comme valeur rang dans la liste.

c - le résultat de l'opération de synthèse est affecté à l'identificat qui se trouve à gauche de / = .

La pseudo-arborescence résultat peut elle même contenir des élém indexés, si un élément de la liste de synthèse est vide (l'élément indexé la pseudo-arborescence modèle reste inchangé) ou contient un élément i

L'instruction d'affectation de synthèse échoue lorsque l'expression modèle ou une des expressions de la liste de synthèse échoue, dans ce ca étapes suivantes ne sont pas exécutées.

2.6.5 Affectation d'Analyse

2.6.5.1 Syntaxe

< INSTRUCTION D'AFFECTATION D'ANALYSE > ::=

< EXPRESSION TEST > = / < EXPRESSION MODELE > < LISTE D'ANALYSE >

< EXPRESSION TEST > ::= < EXPRESSION >

< LISTE D'ANALYSE > ::= < VIDE > | < LISTE D'ANALYSE >, <ELEMENT D'ANA

< ELEMENT D'ANALYSE > ::= < VIDE > | < IDENTIFICATEUR >

2.6.5.2 Exemples

N = /1.
Y = / (EXPRESSION / EXPRESSION) + (TERME)), Y, T1.
X = / (EXPRESSION / (EXPRESSION) + (TERME)) , , .
X = / (EXPRESSION / - (TERME)) , .
VAR = / (TERME / (FACTEUR) * (FACTEUR) * (FACTEUR)), X, , Z.

2.6.5.3 Sémantique

L'instruction d'affectation d'analyse est effectuée comme suit :

a - Les expressions test et modèle sont évaluées ; soit r et s leurs valeurs respectives ; on désignera respectivement par p et q le plus grand entier apparaissant dans le mot des feuilles (D.1.2.2.5) de s et le nombre d'éléments de la liste d'analyse ;
si le $i^{ième}$ élément de la liste d'analyse n'est pas vide, l'élément indexé i doit apparaître une fois et une seule dans le mot des feuilles φ (s) de s autrement il y a erreur ;

b - Supposons que φ (s) contienne une fois et une seule chacun des p premiers entiers, et aucun autre ; nécessairement $p \leqslant q$. On effectue l'analyse de r pour s, et pour i = 1, 2, ..., q, si le $i^{ième}$ élément de la liste d'analyse est un identificateur on lui affecte la $i^{ième}$ des pseudo-arborescences qui composent le résultat de cette analyse.

c - Dans le cas général, on se ramène à l'hypothèse faite en (b) par la transformation suivante :
supposons que l'entier i ($1 \leqslant i \leqslant p$) apparaisse k fois ($k = 0$ ou $k \geqslant 2$) dans φ (s) ; le $i^{ième}$ élément de la liste d'analyse est nécessairement vide ; on remplace les k occurrences de i dans φ (s), par k entiers à partir de i, on ajoute k-1 aux entiers supérieurs à i dans φ (s) et on remplace le $i^{ième}$ élément de la liste d'analyse par k éléments vides. On peut alors effectuer l'étape b).

Cas d'échecs.

- les expressions test et modèle échouent;
- l'opération d'analyse échoue (paragraphe 1.3.3) ;

dès qu'un échec apparait lors d'une étape, les étapes suivantes ne sont pas exécutées.

Remarque

Une instruction d'affectation d'analyse, avec une expression mode dont le mot des feuilles de la valeur ne contient aucun élément indexé et expression d'analyse vide permet de comparer les valeurs r et s de l'ex sion modèle et de l'expression test : il y a échec si et seulement si r ≠ ce qui peut être explicité par une instruction de saut conditionnel.

Par exemple A = /2 permet de tester si l'identificateur A a pour leur 2.

2.6.6 Instruction de Saut - Déroulement de Séque

2.6.6.1 Syntaxe

< INSTRUCTION DE SAUT > ::= < INSTRUCTION DE SAUT INCONDITIONNEL > |
< INSTRUCTION DE SAUT CONDITIONNEL > | < INSTRUCTION DE SORTIE >

< INSTRUCTION DE SAUT INCONDITIONNEL > ::= + < NUMERO D'INSTRUCTION >.

< INSTRUCTION DE SAUT CONDITIONNEL > ::=
+ < NUMERO D'INSTRUCTION > UNLESS < INSTRUCTION DE CONDITION > |
+ < NUMERO D'INSTRUCTION > IF < INSTRUCTION DE CONDITION >

< INSTRUCTION DE CONDITION > ::= < INSTRUCTION D'AFFECTATION > |
< APPEL DE FONCTION >

< INSTRUCTION DE SORTIE > ::= $RETURN (< EXPRESSION >). | $ RETURN.

2.6.6.2 Exemples

```
4/ X/ = (POLYNOME / - (TERME)), X.
+ 10.
+ 1 UNLESS X = / (PRIMAIRE/ (() (POLYNOME) ())). X.
  1/ $ RETURN (X).
1/ +2 IF NORMTEST (Y).
55 / + 48 IF X = / (TERME / (TERME) * (FACTEUR)) , , .
     + 48 IF X = / (TERME / (TERME) / (FACTEUR)) , , .
     + 2 UNLESS N = /2.
```



```
48 / X/ = (PRIMAIRE / (() (POLYNOME) ()), X.  $ RETURN (X).
     + 1 UNLESS Y = / VAR.
$ RETURN ( SOMME (PRODUIT (PRODUIT (DIFF (Y), Z),
                PUISSANCE (Y, SOMME (Z, (POLYNOME/-1) ))),
                PRODUIT (DIFF (Z), T1 ))) ).
```

2.6.5.3 Sémantique

Une instruction de saut conditionnel ou inconditionnel permet une rupture de séquence, à l'intérieur de la plus petite procédure où elle se trouve, vers une instructions repérée par son numéro d'instruction (voir paragraphe 2.6.2).

Une instruction de saut conditionnel contenant UNLESS est exécutée comme suit :

- l'instruction de condition est exécutée,
- en cas d'échec le contrôle passe à l'instruction repérée par le numéro d'instruction, en cas de succés le contrôle passe à l'instruction suivante.

Si l'instruction conditionnelle est construite avec IF les branchements sont inversés.

Une instruction de saut inconditionnelle entraine une rupture de séquence vers l'instruction reperée par son numéro d'instruction, sans autre action.

Instruction de Sortie

La forme $ RETURN (EXPRESSION) est executée comme suit :

- l'expression est évaluée, en cas d'échec de l'évaluation la procédure échoue sans autre action ;
- dans le cas contraire la valeur de l'expression sera le résultat de la plus petite procédure contenant l'instruction de sortie, cette procédure perdant le contrôle.

Forme & RETURN

la procédure la contenant perd le contrôle, son résultat est la ramification vide.

Une procédure peut aussi se terminer par l'éxécution de sa dernière instruction ; dans ce cas son résultat est la ramification vide.

2.7 SECTION DES PROCEDURES

2.7.1. Syntaxe

< SECTION DES PROCEDURES > ::= $ PROGRAM
    < SUITE DE DECLARATIONS PRINCIPALES >
    < SUITE DE DEFINITIONS DE PROCEDURES >

< SUITE DE DECLARATIONS PRINCIPALES > ::= < VIDE > |
    < SUITE DE DECLARATIONS PRINCIPALES > < DECLARATION PRINCIPALE > |
    < SUITE DE DECLARATIONS PRINCIPALES > < COMMENTAIRES >

< DECLARATION PRINCIPALE > ::= < DECLARATION REMANENTE > |
    < DECLARATION DE CONSTANTE > |
    < DECLARACTION D'ELEMENT CATENAIRE > |
    < DECLARATION DE PROCEDURE >

< DECLARATION REMANENTE > ::= $ OWN < LISTE D'INITIALISATIONS >.

< DECLARATION DE CONSTANTE > ::= $ PCON < LISTE D'INITIALISATION >.

< DECLARATION D'ELEMENT CATENAIRE > ::=
    $ IDA < LISTE DE DECLARATION D'ELEMENTS CATENAIRES >.

< DECLARATION DE PROCEDURE > ::= $ GEN < LISTE D'IDENTIFICATEURS >.

< LISTE D'IDENTIFICATEURS > ::= < IDENTIFICATEUR > |
    < LISTE D'IDENTIFICATEURS > , < IDENTIFICATEUR >

< LISTE D'INITIALISATIONS > ::= < IDENTIFICATEUR > < INITIALISATION > |
    < LISTE D'INITIALISATIONS > , < IDENTIFICATEUR > < INITIALISATION >

<INITIALISATION> ::= <RAMIFICATION VIDE> | = <PRIMAIRE> | = <ENTIER> |
    = < IDENTIFICATEUR >

< LISTE DE DECLARATION D'ELEMENTS CATENAIRES > ::=
    < PRIMAIRE > < INITIALISATION > |
    < LISTE DE DECLARATION D'ELEMENTS CATENAIRES >, <PRIMAIRE> <INITIALISATION>

< SUITE DE DEFINITIONS DE PROCEDURES > ::= < VIDE > |
    < SUITE DE DEFINITIONS DE PROCEDURES > < DEFINITION DE PROCEDURE >

< DEFINITION DE PROCEDURE > ::= $ GENERATOR < IDENTIFICATEUR >
    ((< PARTIE DES PARAMETRES >)
    < SUITE DE DECLARATIONS > < SUITE DE DEFINITIONS DE PROCEDURES >
    < SUITE D'INSTRUCTIONS >). |
    < DEFINITION DE PROCEDURES > < COMMENTAIRE >

< PARTIE DES PARAMETRES > ::= < VIDE > |
    < LISTE DE PARAMETRES FORMELS >

< LISTE DE PARAMETRES FORMELS > ::= < IDENTIFICATEUR > |
    < LISTE DE PARAMETRES FORMELS > , < IDENTIFICATEUR >

< SUITE DE DECLARATIONS > ::= < VIDE > |
    < SUITE DE DECLARATIONS > < DECLARATION > |
    < SUITE DE DECLARATIONS > < COMMENTAIRE >

< DECLARATION > ::= < DECLARATION PRINCIPALE > | < DECLARATION LOCALE >

< DECLARATION LOCALE > ::= $ LOCAL < LISTE D'INITIALISATION >

< SUITE D'INSTRUCTIONS > ::= < VIDE > |
    < SUITE D'INSTRUCTIONS > < INSTRUCTION > |
    < SUITE D'INSTRUCTIONS > < ETIQUETTE D'INSTRUCTION >.

2.7.2 Exemples

1° -
```
$ OWN SON, DON. $ PCON BELLE = (FACTEUR / ABC).
$ GENERATOR BETTA ((BO) $ GEN SUE, MAT.
        $ GENERATOR SUE ((SON)) $ LOCAL TAU.
        ... TAU = ADD (MAT (BELLE), SON). ...).
        $ GENERATOR MAT ((JO) $ OWN BO.
        ... BO = BETTA (SON).  ...).
... DON = MAT.  SON = BETTA (SUE (BO)). ...).
```

2° -

```
$ LOCAL  TON.
$ OWN  SON, DON.
$ OWN  X, Y = (FACTEUR/AB), Z.
$ PCON  BELLE = (TERME/A±B).
$ PCON  TERM3 = (TERME/ (FACTEUR) * (FACTEUR)*(FA
$ IDA  (NOM FONCTION / SIN) = SINDIF.
```

3° -

```
$ GENERATOR  PR ((  )  CR (33). ).
```

2.7.3 Sémantique

2.7.3.1 Généralités

Un identificateur de procédure (non-standard) désigne une pro re principale s'il n'apparaît pas dans une déclaration de procédure, ou apparait dans la suite de déclaration principale.

Les déclarations d'éléments caténaires seront traités au para phe 2.7.3.6 .

Les autres déclarations permettent d'identifier chaque occure d'identificateur contenue dans une expression comme désignant une cons une variable ou une procédure. Pour cela à une telle occurrence d'identi cateur :

a) on associe la déclaration du même identificateur (ou le para tre formel qui lui est identique) en tête de la plus petite procédure conte l'occurrence considérée et une telle déclaration (ou un tel paramètre for s'il en existe ; le genre de la déclaration (constante, remanente ou local procédure, un paramètre formel ne possède aucun genre) indique alors l tification cherchée et une initialisation peut être faite (voir paragraphes vants).

b) Sinon et si l'identificateur considéré est l'identificateur d'u procédure principale, l'occurrence considérée désigne cette procédure;

c) sinon et si l'identificateur considéré est l'identificateur d'u fonction standard, l'occurrence considérée désigne cette fonction standa

d) Sinon le programme est erroné.

2.7.3.2 Déclaration de constante

Une constante est un identificateur dont la valeur, précisée par l'initialisation, ne peut être modifiée, elle ne peut donc apparaître à gauche d'une instruction d'affectation (paragraphe 2.6.2) ou dans une liste d'analyse (paragraphe 2.6.5.1).

2.7.3.3 Déclaration rémanente

Si elle apparait dans une procédure, la valeur de la variable est disponible pour les appels suivants de la procédure lorsque cette procédure perd le contrôle, sinon cette valeur est disponible dans tout le programme. L'initialisation lui attribue une valeur avant l'éxecution du programme.

Remarque

Une variable rémanente peut apparaître dans une suite d'étiquettes (paragraphe 2.4.1.3.b) si sa valeur est une procédure et si sa déclaration apparaît dans la suite de déclaration principale (voir exemple ci-contre DONNEE).

Exemple

```
$ PRIMSYN
  ...
  DONNEE / (A) =  (B) (C).
$ PROGRAM  $ OWN DONNEE.
$ GENERATOR  ALPHA ((X)  $ GEN  SP1 , SP2
        ...
        $ GENERATOR  SP1 (  ...  ).
        $ GENERATOR  SP2 (  ...  ).
        ...
        DONNEE = SP2            ...).
```

2.7.3.4 Déclaration locale

A chaque appel de la procédure contenant la déclaration, la valeur de la variable est indéfinie, elle est précisée par une initialisation une instruction d'affectation (paragraphe 2.6.3).

2.7.3.5 Déclaration de procédure

Pour chaque identificateur d'une déclaration de procédure, une procédure de même nom doit être définie ensuite dans la plus petite procédure contenant la déclaration.
Si une occurrence d'identificateur est associée à une déclaration de procédure il désigne alors la procédure de même nom ainsi définie.

Un identificateur de procédure ne peut être écrit que dans la plus petite procédure contenant sa déclaration, cependant une variable peut avoir pour valeur une procédure F même en dehors de la plus petite procédure où F est déclarée, ce qui permet d'appeler F hors de G.

Exemple

```
$ PROGRAM  $ OWN  T.
     ...
$ GENERATOR G ((X) $ GEN F.
|      [$ GENERATOR  F ((Y, Z)
|                     ...  ).
|  ...   T = F.
|                     ...  ).
     ...
  W = T (U, V).
```

2.7.3.6 Déclaration d'élément caténaire

Le premier membre d'une déclaration d'élément caténaire doit avoir pour valeur une pseudo-arborescence réduite à un élément caténaire (paragraphe 2.4.4.3.b). Avant l'exécution du programme la valeur du second membre sera rangée dans l'entrée de table associée à cet élément.
La signification d'une déclaration d'élément caténaire est donc indépendante de la place qu'elle occupe dans le programme.

Exemple :

```
$ PRIMSYN
  ...
  $ IDENT, 2/ (NOM DE FONCTION) = (CHAINE)
  ...
```



```
$ PROGRAM
  ...
  $ IDA (NOM DE FONCTION / SIN) = SINDEF
```

la déclaration range dans l'entrée reperée par SIN dans la table 2, la pseudo-arborescence SINDEF.

Exemple De déclarations et de signification de leur portée

```
1    $ PRIMSYN ...
     1.1  BETTA / ...
     1.2  DON / ...
2    $ PROGRAM
2.1       $ OWN, SOL, DON.  $ PCON BEL = (FACTEUR/ABC).
3    $ GENERATOR
3.1       BETTA ((BON) $ GEN SON,  MON.
4             $ GENERATOR  SON ((SOL) $ LOCAL  TON.
5             ... TON = ADD (MON (BEL), SOL).   ...).
6             $ GENERATOR  MON ((JO),  $ OWN  BON.
7                 BON = BETTA (SOL) .
8  ... DON = MON.    SOL = BETTA (SON (BON)).    ...).
```

L'OCCURRENCE DE :

(L.1.1)	BETTA	désigne une procédure principale.
(L.1.2)	DON	est associée à la déclaration de L.2.1
L.5	TON	est associée à la déclaration de L.4
	ADD	désigne une procédure standard.
	MON	est associée à la déclaration L.3.1
	BEL	est associée à la déclaration L.2.1
	SOL	est associée à la liste des paramétres formels L.4
(L.7)	BON	est associée à la déclaration L.6
	BETTA	désigne une procédure principale.
	SOL	est associée à la déclaration L.2.1.

(L.3)

DON	est associée à la déclaration de L. 2.1
MON	est associée à la déclaration de procédure L. 3.1
SOL	est associée à la déclaration de L. 2.1
BETTA	désigne une procédure principale.
SON	est associée à la déclaration de L. 3.1
BON	est associée à la liste des paramètres formels de L.1

2.8 EXTENSIONS

Ces extensions conduisent surtout à une généralisation de la notion d'expression définie paragraphe 2.5.1.

Expressions

2.8.1 Syntaxe

Elle complète le paragraphe 2.5.1.1.

(1) < EXPRESSION > ::= < NOMBRE SANS SIGNE > | - < NOMBRE SANS SIGNE >

(2) $$ < SYMBOLE TRAITE > |

(3) ($ IDENT, < ENTIER > / < SUITE DE SYMBOLES TRAITES >) |

(4) ** |

(5) $ CSB (<CONSTANTE MODELE > < LISTE D'EXPRESSIONS >)

(6) $ SB (<EXPRESSION MODELE> < LISTE DE SYNTHESE >)

< NOMBRE SANS SIGNE > ::= < NOMBRE DECIMAL > |

< NOMBRE SANS SIGNE > < FACTEUR DE CADRAGE >

< NOMBRE DECIMAL > ::= < ENTIER > | < PARTIE SIGNIFICATIVE >

< PARTIE SIGNIFICATIVE > ::= * < ENTIER > | < ENTIER > *

< ENTIER > * < ENTIER >

< FACTEUR DE CADRAGE > ::= E < ENTIER SIGNE > Q < ENTIER SIGNE >

< ENTIER SIGNE > ::= < ENTIER > | - < ENTIER >

< SUITE DE SYMBOLES TRAITES > ::= < VIDE > |

< SUITE DE SYMBOLES TRAITE > < SYMBOLE TRAITE >

< CONSTANTE MODELE> ::= < EXPRESSION >

< LISTE D'EXPRESSIONS > ::= <VIDE> | <LISTE D'EXPRESSIONS> , <ELEMENT DE LISTE>

< ELEMENT DE LISTE > ::= < VIDE > | < EXPRESSION >

2.8.2 Exemples

(2)
```
$$ A
$$ (()
$$ ($ EF)
```

(3)
```
($ IDENT, 2/  A+B*C-+D // E)
($ IDENT, 4/  C (.) A (,)  ($EF)
```

(4)
```
$ CSB  ((FACTEUR/A))
$ CSB  ((TERME / A*C-D), (FACTEUR / BCD), (POLYNOME / A
$ CSB  (VAR1, VAR2, ..., VARn).
```

2.8.3 Sémantique

Forme 1

Elle désigne un nombre réel décimal positif ou négatif avec ou sans facteur de cadrage.

Forme 2 $$ < SYMBOLE TRAITE >

Sa valeur est l'entier octal valeur du code de sortie du symbole tra (paragraphe 2.3).
dans les exemples précédents si :

A est défini par la description standard de caractère (21) 21
(() est défini par la description standard de caractère (74) 74
($EF) est défini par la description de caractère $ CHARDEF ($EF
(101) 100.

$$A aura la valeur 21
$$(() aura la valeur 74
$$($EF) aura la valeur 100.

Forme 3 ($ IDENT, < ENTIER > / < SUITE DE SYMBOLES TRAITE
($ IDENT, n/s)

sa valeur est la chaîne des codes de sortie (paragraphe 2.2.3) de c que élément de s.

($ IDENT, 4/ A (() , ($EF)) représentera la suite d'entiers octau 21 74 100.

Forme 4 **

sa valeur est la ramification vide.

Forme 5 $ CSB (< CONSTANTE TAMPON > < LISTE D'EXPRESSIONS >)

Son évaluation, exécutée à la compilation, est analogue à celle de l'instruction d'affectation de synthèse, (paragraphe 2.6.4.3.a,b). Les opérandes seront la pseudo-arborescence valeur de la constante modèle et la suite des pseudo-arborescences valeur des éléments de la liste d'expression, le résultat de l'opération de synthèse sera celui de l'expression.

La valeur d'un élément de la constante modèle ou de la liste d'expressions est une expression à l'exclusion d'un appel de fonction.

Forme 6 $ SB (< EXPRESSION MODELE > < LISTE DE SYNTHESE >)

Compléte le paragraphe 2.5.3.1.

Son évaluation est analogue à celle de l'instruction d'affectation de synthèse (avec des expressions plus complexes), les opérandes seront la pseudo-arborescence valeur de l'expression modèle, la suite des pseudo-arobrescences, valeur des éléments de la liste de synthèse, le résultat de l'opération de synthèse sera celui de l'expression.

Exemple :

SORTIE ($ SB (TERME/ (FACTEUR) * (FACTEUR), X, Y).

Remarque

1)
```
$ SB ((POLYNOME / (TERME) + (TERME)), (TERME / (FACTEUR) *
          (FACTEUR)), PROC (X, Y))
```
est équivalent à :

(POLYNOME/ (TERME) + (TERME), U, V) avec
U = (TERME / (FACTEUR) * (FACTEUR))
V = PROC (X, Y)

2) La forme

< IDENTIFICATEUR > = $ SB (< EXPRESSION MODELE > < LISTE DE SYNTHESE >)

est équivalente à l'instruction d'affectation de synthèse.

< IDENTIFICATEUR >/= < EXPRESSION MODELE > < LISTE DE SYNTHESE >

Exemple d'un programme COGENT réunissant les 3 sections, la description des procédures n'étant pas explicitée.

```
$ CHARDEF    ($ EOF) = (101) 101.
$ PRIMSYN    ((SEQ PHR) ($ EOF))
             (LETTRE) = A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q,
                        S, T, U, V, W, X, Y, Z.
             (CHIFFRE) = 0, 1, 2, 3, 4, 5, 6, 7, 8, 9.
             (CHAINE NUM) = (CHIFFRE), (CHAINE NUM) (CHIFFRE)
             (CHAINE ID) = (LETTRE), (CHAINE ID) (LETTRE),
                                     (CHAINE ID) (CHIFFRE
$ DEC /      (ENTIER ) = (CHAINE NUM).
$ IDENT, 1/  (VARIABLE) = (CHAINE ID).
$ IDENT, 2/  (ID FONCTION) = (CHAINE ID).
$ NOP /      (PRIMAIRE) = (ENTIER), (VARIABLE).
             (PRIMAIRE) = (( ) (EXP) ( )).
             (PRIMAIRE) = (ID FONCTION) (() (SEQ EXP) ()).
DIFF/        (PRIMAIRE) = $ D (() (VARIABLE) (,) (EXP) ()).
$ NOP        (FACTEUR) = (PRIMAIRE).
             (FACTEUR) = (FACTEUR) * * (PRIMAIRE).
$ NOP /      (TERME)   = (FACTEUR).
             (TERME) = (TERME) * (FACTEUR), (TERME) / (FACTEU
$ NOP/       (EXP) = (TERME), + (TERME).
             (EXP) = - (TERME), (EXP) + (TERME), (EXP) - (TERME
$ NOP/       (SEQ EXP) = (EXP).
             (SEQ EXP) = (SEQ EXP (,) (EXP).
SORTIE/      (PHR) = EXP (.).
DEPART/      (SEQ PHR) = .
PAS DE RES/  (SEQ PHR) = (SEQ PHR) (PHR)

$ PROGRAM    $ OWN   VAR, VX, VY.
```



```
$ GENERATOR  DIFF ((X, Y)
       DAR = X.   $ RETURN (CLOTHE (DIFFT (Y), 3)).).

$ GENERATOR  SORTIE ((X)
       X/ = (PHR/ (EXP) (.)), STRIP (X).
       SPACED () . STANDSON (X, PUIP). OUTP ( ). ).

$ GENERATOR  DEPART (() SETIVGOL (0). ).

$ GENERATOR  PAS DE RES ((X) ... ).

$ GENERATOR  DIFF1  ((T, U) ... ) .

$ GENERATOR  STRIP ((X)  ...  ).
```

2.9 PROCEDURES STANDARD

2.9.1 Généralités

2.9.1.1

Comme les autres procédures, les procédures standard ont pou arguments et pour résultat des pseudo-arborescences sur le vocabulaire fondamental ; dans certains cas il s'agit plus particulièrement de pseudo- arborescences réduites à un élément, ce qui permet par exemple d'effec des opérations arithmétiques en utilisant des éléments numériques. Une cédure standard peut, comme une autre, soit donner un résultat, soit éch

2.9.1.2 Variables internes

Certaines procédures standard emploient des variables interne c'est-à-dire des variables non directement accessibles au programmeur, ont pour but de permettre une communication entre procédures standard noms de ces variables seront représentés dans la suite par des mots sou gnés, pour les distinguer des identificateurs. Chaque variable interne po de une valeur initiale au début de l'exécution. Elle est accessible par deu fonctions standard :

- la première possède un argument et affecte à la variable la va de cet argument ; son résultat est la ramification vide ; son identificateur formé du "prefixe" SETIV (SET Internal Variable) et du nom de la variabl interne ;

- la seconde, qui n'a pas d'argument admet pour résultat la valeu de la variable interne ; son identificateur est formé du préfixe IV (Interna Variable) et du nom de la variable.

Exemple :

rem désigne une variable interne, qui après l'exécution de la fon tion DIVIDE (r, s) a pour valeur l'élément numérique égal au reste de la di sion entière de r par s ; sa valeur initiale est zéro ; SETIVREM (r) donne rem la valeur de r ; IVREM ( ) a pour résultat la valeur de rem.

La variable interne isn a pour valeur une procédure. Cette proc transforme une pseudo-arborescence sur l'alphabet fondamental réduit à

éléments de type 1, 2, 3, 4, 5 (paragraphe 2.4.4.3.b) en une chaîne sur l'alphabet des codes de sortie des symboles traités. Cette procédure possède deux arguments, la pseudo-arborescence à transformer et une procédure qui opérera sur les codes de la chaîne. La chaîne est rangée dans une zone tampon commune à plusieurs procédures.

Le paragra. 2.9.10 donne la liste des variables internes et précise pour chacune d'elle les valeurs qu'elle peut prendre, sa valeur initiale et son usage.

2.9.1.3 Rappels et compléments sur l'alphabet fondamental

Rappelons qu'un programme COGENT traite des pseudo-arborescences sur l'alphabet fondamental (paragraphe 2.4.4.3.b) formé des élément ments suivants :

(1) - (2) éléments grammaticaux

Ils sont formés d'un numéro de règle simple et de deux marques M1 et M2 prenant les valeurs 1 ou 0. Les marques sont positionnées à 0 après l'analyse d'un mot, elles sont recopiées sans modification dans une opération de synthèse, elles entrainent un échec de l'opération d'analyse (paragraphe 1.3.3) si deux éléments qui se correspondent dans les pseudo-arborescences test et tampon n'ont pas des marques identiques.

(3) - éléments indexés

Ce sont des entiers associés à des non-terminaux (paragraphe 2.5.2.3).

(4) - éléments numériques

Ils sont formés d'une indication de mode (entier ou flottant) et de valeur de la chaîne numérique qu'ils représentent, la longueur d'une chaîne numérique entière n'est pas limitée, la longueur et la précision d'une chaîne numérique flottante dépend du système.

(5) - éléments caténaires

Ce sont des couples, formés par un entier et une chaîne construite sur l'alphabet des codes de sortie ; un tel couple désigne une entrée dans une table dont le numéro est l'entier ; dans cette table est rangée une information associée à l'élément caténaire. Si l'entier est égal à 0 il ne désigne aucune entrée.

(6) - éléments procédures

Ils représentent les procédures du programme.

2.9.2 Procédures associées aux marques

Deux groupes aux fonctions analogues, distinguées par un suffixe 1.2, sont associés à chaque marque. Leur argument doit être une pseudo arborescence d'ordre supérieur à 1 et n'apparaissant pas dans un primair Leur résultat est la ramification vide si la marque correspondante vaut 1, sinon elles échouent, dans tous les cas l'opération qu'elles désignent est e cutée.

TSTMARK1 [2]	marque reste inchangée
SETMARK1 [2]	positionne la marque à 1
CLRMARK1 [2]	positionne la marque à 0
CMPMARK1 [2]	complémente la marque.

2.9.3 Procédures Associées aux Elements Indexés

PINDEX (B)

son résultat est l'entier formant l'élément indexé B.

2.9.4 Procédures Associées aux Elements Numéri

2.9.4.1 Création d'éléments numériques

s désigne une pseudo-arborescence quelconque.

ZERO

possède zéro ou plusieurs arguments, son résultat est l'entier

DECCON (s)

a) Si s est un élément numérique entier, le résultat sera s.

b) Si s est un élément numérique flottant, le résultat sera un élément numérique entier dont la valeur absolue sera la partie entiere de la leur absolue de s et le signe celui de s.

c) Si s n'est pas un élément numérique, la procédure valeur de est exécutée avec pour arguments s et une procédure de traitement de caractères non accessible au programmeur, cette procédure crée une chaîne construite sur l'alphabet des codes de sortie et rangée dans la zone tampon, résultat est la ramification vide.

Le résultat de DECCON est un élément numérique, entier, positif, obtenu à partir du contenu de la zone tampon, les codes de cette zone qui ne sont pas associés à des chiffres sont ignorés. DECCON échoue si la procédure valeur de isn échoue.

OCTCON (s)

fonctionne comme DECCON mais donne une représentation octale de s.

FLOATCON (s).

a) Si s est un élément numérique flottant, le résultat sera s.

b) Si s est un élément numérique entier, le résultat sera obtenu par conversion de s en mode flottant.

c) Si s n'est pas un élément numérique, on éxecute DECCON (s). Puis le résultat de DECCON est converti en mode flottant ; ce nombre flottant est ensuite divisé par 10^n où n représente le nombre de chiffres suivant le dernier code de la zone tampon non associé à un chiffre ; ce dernier code est donc interprété comme un point décimal. L'élément numérique flottant, résultat de la division, sera le résultat de FLOATCON.

Remarque

Elle concerne les procédures opérant en mode flottant.

Pour un débordement inférieur de l'exposant dans une opération arithmétique flottante le résultat de la procédure est un zéro flottant, pour un débordement supérieur de l'exposant résultant d'une telle opération ou d'une conversion entier-flottant la procédure échoue.

Exemple :

Pour plus de clarté on opérera directement sur les caractères et non leur code de sortie contenus dans la zone tampon.

Contenu de la zone tampon	Résultat entier de DECCON	Conversion entier-flottant	Résultat de FLOATCON
1.23 ± 45 A 67	1234567	$0.1234567\ 10^7$	0.123456710^5

$ DEC /, $ OCT /, $ FLOAT /

ces étiquettes spéciales définies paragraphe 2.4.4.3.d sont exécutées respectivement comme DECCON, OCTCON, FLOATCON.

2.9.4.2 Traitement d'éléments numériques

A, B, N désignent des éléments numériques, N est de mode entier (sinon il y a erreur), r et s des pseudo-arborescences.

a) le résultat des procédures suivantes sera un élément numérique en mode entier si A et B sont de mode entier, en mode flottant sinon. L'opération est exécutée dans le mode du résultat (voir remarque du paragraphe 2.9.4.1).

PROCEDURES	RESULTAT
ADD (A, B)	A+B
SUB (A, B)	A-B
INCREASE (A)	A+1
DECREASE (A)	A-1
NEG (A)	-A
ABS (A)	|A|
MULT (A, B)	A * B
DIVIDE (A, B)	A/B

Si A et B sont entiers rem est affecte du reste entier de la division de A par B, le signe de rem sera celui de A.
DIVIDE (A, B) échoue si B = 0.

b)

POWER2 (A, N)

le résultat égal à $A * 2^N$, aura le mode de A. Si A est entier et $N < 0$, le signe du résultat sera celui de A, sa valeur absolue la partie entière de $|A * 2^N|$.

POWER10 (A, N)

A doit être en mode flottant

le résultat égal à $A * 10^N$ sera en mode flottant.

RANDOM (A)

Génère des valeurs pseudo-aléatoires impaires sur le segment $[s, A]$. Ces valeurs sont ensuite affectées à la variable interne rdm.
RANDOM est exécuté comme suit :

a) la valeur de la variable interne rdm est affectée de $(5^{15} \cdot \text{rdm}) \bmod 2^{47}$.

b) Le résultat est $(A * \text{rdm}) / 2^{47}$ son mode est celui de A, ensuite on reprend l'étape a).

LARGER (A, B).

Si la valeur de A est supérieure à celle de B, le résultat est la ramification vide, sinon il y a échec. Si les arguments ne sont pas de même mode, l'argument entier est d'abord converti en mode flottant (voir remarques paragra. 2.9 4.1), sinon la comparaison utilise le mode commun.

PSLARGER (r, s)

Si r > s par rapport à un ordre total sur les pseudo-arborescences défini par l'implémentation son résultat est la ramification vide, sinon il y a échec.

2.9.5 Procédures Associées aux Elements Catenaires

2.9.5.1 Création d'éléments caténaires

s désigne une pseudo-arborescence quelconque et i un entier.

IDENT (s, i)

a) La procédure valeur de isn est exécutée avec pour arguments s et une procédure de traitement de caractère, non accessible au programmeur, cette procédure crée une chaîne α construite, sur l'alphabet des codes de sortie et rangée dans la zone tampon, son résultat est la ramification vide.

b) le résultat de IDENT sera :

si $i \neq 0$ et si le contenu α de la zone tampon désigne une entrée de la table i, l'élément catenaire (i, α) sinon un élément caténaire formé avec i et le contenu de la zone tampon, ce couple (i, α) désigne une entrée de la table i où sera rangée la valeur de ial.
IDENT échoue si la procédure valeur de isn échoue, en cas de débordement de la zone tampon, il y a erreur avec arrêt du programme.

$ IDENT, i/

est une étiquette spéciale identique à IDENT définie paragraphe 2.4.4.3.d.

CIDENT (s, i)

analogue à IDENT (s, i), son résultat est un élément caténaire (i, α) si un tel élément désigne une entrée de la table i, sinon il y a échec sans création d'élément.

Remarque

La procédure valeur de isn ne peut appeler IDENT, CIDENT, DECCON, OCTCON, ou FLOATCON qui utilisent la même zone tampon.

2.9.5.2 Traitement d'éléments caténaires

A désigne un élément caténaire et i un élément numérique entier (sinon il y a erreur).

SETA (A, X)

la pseudo-arborescence rangée dans l'entrée de table repérée par est affectée à la variable X, son résultat sera la ramification vide

ALIST (A).

Son résultat est la pseudo-arborescence rangée dans l'entrée de table reperée par A sauf s'il s'agit de la ramification vide dans ce cas la procédure sans argument valeur de dal (initialisée par FAILURE) est appelée et son résultat est celui de ALIST, si cette procédure échoue ALIST échoue.

TABLENO (A).

Son résultat est le numéro de table de A.

NEXTID (A).

Son résultat est l'élément caténaire désignant l'entrée qui suit A dans la même table. NEXTID échoué si le numéro de table de A est égal à zéro, ou si A est le dernier élément de la table alors il y a échec.

ERASID (A).

Efface A de sa table et met à O le numéro de table de A, sauf si le numéro de table de A est égal à zéro ou si A appartient à une constante, son résultat est la ramification vide.

FIRSTID (i)

Son résultat est l'élément catnéaire désignant la première entrée de la table indiquée par i. FIRSTID échoue si i=0, ou si la table est vide.

ERASIT (i)

Efface tous les éléments (n'appartenant pas à des constantes) de la table i et met a zéro leur numéro de table sauf si i=0. Son résultat est la ramification vide.

2.9.6 Test de Pseudo-Arborescences

Ces procédures ont pour argument une pseudo-arborescence quelconque. Si le test est satisfait, leur résultat est la ramification vide, sinon elles échouent.

PROCEDURE	Test satisfait si s est
IDENTEST (s)	un élément caténaire
NUMTEST (s)	un élément numérique
FIXTEST (s)	un élément numérique entier
FLOATEST (s)	un élément numérique flottant
POSTEST (s)	un élément numérique (entier ou flottant) à valeur positive ou nulle
ZEROTEST (s)	un élément numérique (entier ou flottant) à valeur nulle
NEGTEST (s)	un élément numérique (entier ou flottant) à valeur négative
PARATEST (s)	un élément indexé
GENTEST (s)	un élément procédure
DUMTEST (s)	la ramification vide
NOMTEST (s)	si tous les tests précédents échouent (c'est-à-dire si la pseudo-arborescence a pour racine un élément grammatical).

2.9.7 Opérations sur les Pseudo-Arborescences

X est un identificateur dont la valeur est une pseudo-arborescence qui dans ce qui suit, on suppose égale à n↑r, où n doit être un élément grammatical (sinon il y a erreur).

s désigne une pseudo-arborescence quelconque, i un entier.

PRODCODE (X).

son résultat est n.

SIZE (X).

son résultat est le nombre de pseudo-arborescences composant la ramification r.

SUBLIST (X, i).

son résultat est la $i^{ième}$ des pseudo-arborescences composant r si elle existe sinon il y a erreur.

SUBLIST1 (X) équivaut à SUBLIST (X, 1)

SUBLIST2 (X) équivaut à SUBLIST (X, 2)

REPLACE (X, s, i).

la $i^{ième}$ pseudo-arborescence composant r si elle existe est remplacée par s, le résultat sera la ramification vide, sinon il y a erreur.

REPLACE1 (X, s) équivaut à REPLACE (X, s, 1)

REPLACE2 (X, s) équivaut à REPLACE (X, s, 2)

Les procédures qui suivent peuvent avoir un nombre quelconque d'argument éventuellement nul. (sauf NOP).

NORESULT

son résultat est toujours la ramification vide.

NOP

doit avoir au moins un argument

Son résultat est toujours son dernier argument, on rappelle que $ NOP/ étiquette spéciale (paragraphe 2.4.4.3.d) doit avoir un seul argument, sinon il y a erreur.

2.9.8 Procédure de Sortie

2.9.8.1 Généralités

Une opération se sortie est exécutée en trois phases, on transforme la pseudo-arborescence qui représente le résultat à éditer, en une chaîne sur l'alphabet des codes de sortie, par l'utilisation d'une procédure de réduction (paragraphe 2.9.8.2 ex. STANDSCN) également utilisées pour la création d'éléments caténaires et d'éléments numériques (paragraphes 2.9.4.1 et 2.9.5.1), puis chaque code de sortie est généralement argument d'une procédure de traitement de caractère valeur de chr (paragraphe 2.9.8.3 - ex. PUTP) qui place chaque code dans une des trois zones de sortie disponibles, enfin une procédure d'impression (paragraphe 2.9.8.5 ex. OUTP) affecte le contenu de la zone de sortie ou périphérique correspondant.

En réalité une procédure de réduction exécute les deux premières phases, ses arguments sont la pseudo-arborescence à éditer et une procédure de traitement de caractère.

Les chaînes créées par les procédures de réduction, de création d'éléments numériques, de création d'élément caténaire, sont rangées dans une zone commune ou zone tampon, le contenu de la zone tampon est ensuite utilisé par les procédures de traitement de caractère, un débordement de cette zone entraîne une erreur.

Exemple d'instruction de sortie

STANDSCN (X, PUTP). OUTP ( ).

Une table d'impression qui gère la mise en page et des procédures qui définissent des opérations annexes (dump, suppression de message, ...) complètent les opérations de sortie.

2.9.8.2 Zones de sortie

2.9.8.2.1 Définitions

Zone P (imprimante)

Elle est formée d'un caractère de contrôle utilisé pour la mise en page et de 135 positions accessibles, numérotées de 1 à 135 et qui recevront chacune le code de sortie d'un symbole de base. Une initialisation de la zone affecte

toutes les positions (y compris le caractère de contrôle) d'un blanc.

Zone C (carte)

Elle représente 80 positions accessibles, numérotées de 1 à 80, qui recevront chacune le code de sortie d'un symbole de base. Une initialisation de la zone affecte toutes les positions d'un blanc .

Zone b (carte binaire)

Elle représente 960 positions binaires accessibles numérotées de 1 à 960, de haut en bas (12 positions) et de gauche à droite (80 fois) une initialisation de la zone affecte toutes les positions d'un 0, sauf les positions 10 et 12 positionnées à 1 et qui identifient la zone b.

2.9.8.2.2 Procédures associées aux zones de sortie

Ces fonctions remplissent une zone de sortie, généralement en y plaçant un caractère à la fois, le programmeur devra prévoir le traitement des symboles des langages traités qui ne sont pas des caractères de COGENT (paragraphe 2.3.3), dans la suite on différenciera ces caractères en supposant leur code de sortie supérieur à 77_8.

Il existe un groupe de procédures analogues pour chaque zone de sortie

2.9.8.2.2.1 Zone P

PUTP1 (C, P)

C est un élément numérique entier dont la valeur est celle du code de sortie d'un caractère normal ou spécial (paragraphe 2.3.1), P est un entier tel que $1 \leq P \leq 135$, sinon il y a erreur. PUTP1 place la valeur de c dans la position de la zone P dont le rang est égal à la valeur de P. Son résultat est la ramification vide.

PUTP (C).

est une procédure de traitement de caractère, ses appels successifs placeront son argument dans les positions successives de la zone P.
C est un entier dont la valeur est celle du code de sortie d'un symbole traité (paragraphe 2.3.1) sinon il y a erreur.
a) Si C > 77_8, il sera argument de la procédure valeur de por dont le résultat sera celui de PUTP, son échec entraine celui de PUTP.

b) Si pin = pmr c est argument de la procédure valeur de pmc dont le résultat sera celui de PUTP, son échec enchaîne celui de PUTP.
c) Sinon le caractère c est placé dans la position de la zone P dont le rang est la valeur de pin, pin est augmenté de 1. Le résultat sera la ramification vide.

2.9.8.2.2.2 Zone C

L'exécution de PUTP1 et PUTP est analogue respectivement à celle de PUTC1 et PUTC, les variables internes cor, cin, cmr, cmc remplaçant respectivement por, pin, pmr, pmc.

PUTC1 (C, P)

Voir sémantique de PUTP1 ou $1 \leq P \leq 80$.

PUTC (C)

Voir sémantique de PUTP (C).

PUTPC (C)

Procédure de traitement de caractère qui remplit simultanément la zone P et la zone c, sa déclaration est
$ GENERATOR PUTPC ((C) PUTP (C). $ RETURN (PUTC (C)).)

2.9.8.2.2.3 Zone b

PUTB1 (N, P, L)

N, P, L sont des entiers tels que $1 \leq P \leq 960$, $1 \leq L \leq 961-P$, $N \geq 0$ sinon il y a erreurs.
Les positions dont le rang est compris entre les valeurs de P et P+L-1 donnent une représentation binaire de N, réduit module 2^L, la position de rang P contient le bit le plus significatif. Le résultat est la ramification vide.

PUTB (N).

N est un entier supérieur ou égal à zéro sinon il y a erreur, si bin + bwℓ > bmr, N est argument de la procédure valeur de bmc et dont le résultat sera celui de PUTB, son échec entraîne celui de PUTB, sinon les positions dont le rang est compris entre les valeurs de bin et bin + bwℓ -1 donnent une représentation binaire de N réduit module $2^{bw\ell}$, la

position dont le rang est la valeur de bin contient le bit le plus signi-
ficatif, bin est ensuite affecté de la valeur de bin + bwℓ, le résul-
tat est la ramification vide.

2.9.8.3 Procédures de réduction

Role : elle transforme une pseudo-arborescence en une chaîne sur l'alphabet des codes de sortie et par l'intermédiaire d'une procédure de traitement de caractère, place une image de cette chaîne dans une zone de sortie.

NORMSCN (r)

r doit être une pseudo-arborescence (r = n ¶ s) dont la racine (D.1.2.
0.) n est un élément grammatical, auquel est associé une règle simp
de la grammaire primaire ou de la grammaire secondaire. Les sym-
boles du second membre de cette règle sont traités successivement de la manière suivante :

a) s'il s'agit d'un symbole traité, on exécute la procédure valeur de la variable interne chr avec pour argument le code de sortie de ce symbole.

b) s'il s'agit du $i^{ième}$ symbole non terminal du second membre de la règle, on exécute la procédure valeur de la variable interne lsr avec pour argument la $i^{ième}$ des pseudo-arborescences qui composent la ramification s.

Le résultat de NORMSCN est celui de la dernière procédure appelé si une ou plusieurs procédures sont appelées, la ramification vide sinon(aucune procédure n'est appelée si le second membre de la dernière règle traitée est vide).

NORMSCON échoue si une procédure appelée échoue.

Exemple : Si r est le transformé par θ (voir paragraphe 1.3.1 et son exemple) d'une pseudo-arborescence r' engendrée par la réunion des grammaires primaires et secondaires, si chr a la valeur PUTP et lsr la valeur STANDSCN, NORMSCN (r) place dans la zone P le mot des feuilles de r'.

IDENTSCN (A)

A doit être un élément caténaire (i, α) sinon il y a erreur. Pour chaque symbole traité de la chaîne α, on exécute la procédure valeur de la variable interne chr avec pour argument le code de sortie de ce symbole.

Le résultat de IDENTSCN est la ramification vide. IDENTSCN échoue si la procédure valeur de chr échoue.

FDINTSCN (A)

A doit être un élément numérique entier sinon il y a erreur.

a) Si A > 0 FDINTSCN donne une représentation décimale normalisée de A. Pour chaque symbole traité de l'élément numérique on exécute la procédure valeur de la variable interne chr avec pour argument le code de sortie de ce symbole ;
le résultat de FDINTSCN est la ramification vide.

b) si A = 0 la représentation est réduite au chiffre 0. Le résultat est la ramification vide.

c) si A < 0, on exécute la procédure valeur de la variable interne nir (initialisée par FAILURE) avec pour argument A, le résultat de cette procédure sera celui de FDINTSCN.

FDINTSCN échoue si les procédures valeurs de chr et nir échouent.

FOINTSCN (A).

est analogue à FDINTSCN mais donne de A une représentation octale.

XDINTSCN (A)

Comme pour FDINTSCN A doit être un élément numérique entier sinon il y a erreur. XDINTSCN se différentie de FDINTSCN par le traitement des éléments numériques positifs ou nuls

a) si A < 0 on exécute FDINTSCN (A).

b) si A ⩾ 0 XDINTSCN (A) donne de A une représentation décimale sur n positions où n est la valeur de la variable interne fal (initialisée par 0). A devra donc être inférieur à 10^n, les chiffres de l'élément numérique sont traités de la manière suivante :

- s'il s'agit d'un zéro de tête on exécute la procédure valeur de chr avec pour argument la valeur de la variable interne %zc (initialisée par 0). (Si A = 0 tous les chiffres représentent des zéros de tête, sauf le zéro de droite).
- s'il ne s'agit pas d'un zéro de tête on exécute la procédure valeur de chr avec pour argument le code de sortie du chiffre traité. Le résultat de XDINTSCN est la ramification vide. XDINTSCN échoue si $A \geqslant 10^n$ ou si fdl = 0, ou si les procédures valeur de nir et chr échouent.

XOINTSCN (A)

est analogue à XDINTSCN mais donne de A une représentation octale (A devra être inférieur à 8^n).

DFLTSCN1 (A, N)

A doit être un élément numérique flottant, positif différent de zéro, N un élément numérique entier sinon il y a erreur. La procédure DFTSCN1 opère en deux étapes.

a) Elle donne de A une représentation flottante normalisée de la forme $f_1\ f_2 \ldots f_N * 10^e$ où les f_i sont des chiffres décimaux.

b) elle range la chaine $f_1\ f_2 \ldots f_N$ dans une zone tampon secondaire accessible par FLTSCN2.

Le résultat de DFLTSCN1 est l'élément numérique ℓ.

OFLTSCN1 (A, N)

est analogue à DFLTSCN1, mais donne de A une représentation octale de la forme $f_1 . f_2 \ldots f_N * 2^e$ où les f_i sont des chiffres de 0 à 7.

FLTSCN2 (PC, CR, PR)

PC, doit être un élément numérique entier positif ou nul, CR une procédure de traitement de caractère, PR une procédure sans argument sinon il y a erreur.

FLTSCN2 opère sur les N chiffres ($f_1\ f_2 \ldots f_N$) rangés dans la zone tampon secondaire par le dernier appel de DFLTSCN1 ou OFLTSCN1 et PC doit être au plus égal à N.

FLTSCN2 est exécutée en trois étapes :

a) on éxécute la procédure CR, PC fois, au $j^{ième}$ appel ($j \leqslant PC$) son argument sera le code de sortie du chiffre f_j.

b) on exécute PR (généralement son rôle est d'insérer un caractère spécial après le $PC^{ième}$ chiffre, exemple un point décimal).

c) on exécute la procédure CR (N-PC) fois au $k^{ième}$ appel ($PC+1 \leqslant k \leqslant N$) son argument sera le code de sortie du chiffre f_k.

Le résultat de FLTSCN2 est la ramification vide. FLTSCN2 échoue si les procédures valeur de PR et CR échouent, ces procédures ne peuvent appeler FLTCN2, DFLTSCN1, OFLTSCN1.

Exemple :

Supposons que la zone tampon secondaire contienne la chaîne 1 2 3 4 5 6 7 8 et que PR soit défini par

$ GENERATOR PR (( ) CR (33).) où 33 est le code de sortie du symbole de base point (.), alors FLTSCN2 (3, CR, PR) rangera dans une zone de sortie 1 2 3 . 4 5 6 7 8

Contrairement aux précédentes les procédures qui suivent permettent de traiter une pseudo-arborescence quelconque (r) qu'elles transforment en une chaîne de symbole traité, le code de sortie de chaque symbole traité est ensuite argument d'une procédure de traitement de caractère.

STNDSCN1 (r)

est définie par la déclaration de procédure

```
$ GENERATOR  STNDSCN1 ((r)
+1 UNLESS  NORMTEST (r) .  $ RETURN (NORMSCN (r)).
1/+2  UNLESS  IDENTEST (r).  $ RETURN (IDENTSCN (r)).
2/+3  UNLESS  FIXTEST (r).  $ RETURN (IVINR ( ) (r)).
          3/  FLOATEST (r).  $ RETURN (IVFLR ( ) (r).)
```

Dans son usage courant

STNDCN1 s'appelle récursivement (par NORMSCN) si lsr à pour valeur STNDSCN1, chaque code de sortie est argument de la procédure valeur de chr.

STANDSCN (r, CR)

est utilisé par NORMSCN, la procédure de traitement de caractère est précisée par le second argument CR.
STANDSCN est défini par la déclaration de procédure.

```
$ GENERATOR STANDSCN ((r, CR) $ LOCAL CHRSAVE, LSRSAVE
CHRSAVE = IVCHR ( ). SETIVCHR (CR).
LSRSAVE = IVLSR ( ). SETIVLSR (STNDSCN1).
+1 · UNLESS STNDCN1 (r).
SETIVCHR (CHRSAVE). SETIVLSR (LSRSAVE). $ RETURN.
1/ SETIVCHR (CHRSAVE). SETIVLSR (LSRSAVE). $ FAILURE).
```

2.9.8.4 Table d'impression

2.9.8.4.1 Définitions

La table d'impression possède pour chaque unité de sortie une entrée accessible par des fonctions standard, chaque entrée permet l'exécution d'opérations annexes au moment de la sortie de l'enregistrement d'une zone de sortie. Une entrée est formée par un caractère de mise en page (C_{mp}), un compteur de lignes (C_{ℓ}), un compteur de pages (C_p), un libellé (L_b), un caractère de conversion de la zone c (C_{cc}), un caractère de contrôle de partié de la zone b (C_{pb}).

La table d'impression peut être schématisée de la manière suivante :

	C_{mp}	C_{ℓ}	C_p	L_b	C_{cc}	C_{pb}
Unité-1						
Unité-2						

Le libellé à pour valeur la chaîne d'un élément caténaire ou la ramification vide.

Au début de l'exécution tous les caractères de contrôle de la table ne sont pas chargés, tous les compteurs de lignes et de pages sont positionnés à zéro, tous les libellés ont pour valeur la ramification vide :

Le Caractère de mise en page

Il définit des opérations annexes au moment de la sortie d'un enregistrement de la zone P, d'un commentaire, d'un dump. S'il n'est pas chargé les opérations annexes ne sont pas exécutées. S'il est chargé les opérations annexes suivantes sont exécutées :

1) Le compteur de lignes est diminué de 1, si le caractère de contrôle de la zone P est un blanc (espacement simple), de 2 (espacement double) sinon.

2) Si le compteur de lignes est négatif ou si le caractère de contrôle de la zone P vaut 1, les opérations suivantes sont exécutées :

a) le compteur de pages est incrémenté de 1;

b) un enregistrement spécial est crée. Il est formé : d'un caractère de contrôle spécial positionné à 1 (saut de page), de la date, du compteur de pages et éventuellement du libellé ;

c) l'enregistrement spécial est imprimé ;

d) le compteur de lignes est positionné à 56,

e) le caractère de contrôle de la zone P est positionnée à 0 (espacement double).

3) L'enregistrement (de la zone P ou du commentaire ou du dump) est imprimé.

Caractère de conversion

Un enregistrement de la zone-c est sorti sans modification sur une unité si le caractère de conversion n'est pas chargé sinon, il est remplacé par un enregistrement binaire.

Caractère de contrôle de parité.

Avant la sortie d'un enregistrement de la zone-b, si le caractère de contrôle de parité est chargé un contrôle de parité de l'enregistrement sera fait et son résultat inséré dans les positions binaires 25 à 48 de la zone-b, sinon la sortie se fera sans contrôle de parité.

2.9.8.4.2 Procédures associées à la table d'impression

LUN doit être un entier désignant une unité du système s'il est égal à zéro, la procédure est sans action, sinon, l'action de la procédure porte sur la LUN^{ième} entrée de la table d'impression.
Toutes ces procédures sauf PGCNT ont pour résultat la ramification vide.

PAGE (LUN, r) NOPAGE (LUN).

r doit être un élément caténaire (i, α) ou la ramification vide. PAGE charge le caractère de mise en page, positionne le compteur de ligne à zéro, affecte au libellé la chaîne α ou la ramification vide NOPAGE décharge le caractère de mise en page.

CLR PGCNT (LUN).

positionne le compteur de pages à zéro.

CMDCNV (LUN) NOCMDCNV (LUN).

respectivement charge et décharge le caractère de conversion.

BCHKSM (LUN) NOBCHKSM (LUN)

charge et décharge respectivement le caractère de contrôle de parité.

PGCNT (LUN)

son résultat est la valeur du compteur de pages.

2.9.8.5 Procédure d'impression

Elle affecte l'enregistrement d'une zone de sortie à une unité de sortie et l'imprime.

Rôle du caractère de contrôle de la zone-P et procédures associées

Il contrôle la mise en page au moment de la sortie d'un enregistrement de la zone-P. Il peut prendre les valeurs blanc après une initialisation de la zone-P (impression normale) et les valeurs 0 et 1 par l'utilisation de procédures associées
SPACE P et EJECTP.

SPACE P ( ). EJECTP ( ).

Avant impression positionnent respectivement le caractère de contrôle à zéro (saut de ligne) et à un (saut de page). Leur résultat est la ramification vide.

2.9.8.5.1 Impression de la zone-P

OUTP ( ).

assigne l'enregistrement de la zone-P à l'unité désignée par pno. Après impression une nouvelle zone-P est crée et initialisée, pin prend la valeur un. Le résultat est la ramification vide.

STANDPMC (c).

valeur initiale de pmr, sa déclaration est :
$ GENERATOR STANDPMC ((C) OUTP ( ). $ RETURN (PUTP(C)).).

2.9.8.5.2 Impression de la zone-c

OUTC ( ).

voir sémantique de outp et cin remplacent respectivement pno et pin

STANDCMC (C).

Valeur initiale de cmc, sa déclaration est :
$ GENERATOR STANDCMC ((C) OUTC ( ). $ RETURN (PUTC (C)).).

2.9.8.5.3 Impression de la zone-b

OUTB ( ).

voir sémantique de OUTP ou bno et bin remplacement respectivement pno et pin.

2.9.8.6 Gestion de bande

LUN doit être un élément numérique. Si LUN est égal à zéro la procédure est sans action, sinon, l'action porte sur l'unité du système désignée par LUN ; le résultat est toujours la ramification vide sauf pour MASTLUN.

BSPR (LUN)

saut arrière d'un enregistrement.

BSPF (LUN).

saut arrière d'un fichier.

REWIND (LUN).

Rebobinage jusqu'au point de chargement.

UNLOAD (LUN).

Rebobinage et déchargement.

SKIP (LUN).

saut jusqu'à la fin de fichier.

MARKEF (LUN).

Ecriture d'une marque de fin de fichier

SKIPR (LUN)

Saut d'un enregistrement.

MAST LUN (LUN)

son résultat est un élément numérique entier donnant le numéro attribué à l'unité LUN par une carte contrôle. S'il n'y a pas eu d'attribution le résultat est LUN.

2.9.8.7 Procédure de DUMP

Imprime, sur l'unité désignée par pno dans un enregistrement fixe et sans modifier la zone-P, des pseudo-arborescences avec leur structure et les éléments de l'alphabet fondamental qui les composent.

DUMPV (r).

la valeur de r est imprimée.

DUMP1 ( ).

les noms et valeurs des variables locales, rémanentes, des paramétrés formels de la procédure qui appelle DUMP1 sont imprimés.

DUMPALL ( )

fonctionne comme DUMP1, mais en plus pour toute procédure appelée DUMPALL imprime tout élément caténaire (i, α) avec l'information rangée dans l'entrée qu'il repère dans la table i.

Remarque :

Le résultat des procédures de dump, les messages son également sortis sur l'unité désignée par pno.

2.9.9 Procédures Diverses

2.9.9.1 Procédure d'échec

FAILURE

elle échoue toujours, son action est indépendante du nombre de ses arguments, on notera que son écriture $ FAILURE ( ) peut être abrégée en $ FAILURE.

2.9.9.2 Procédures d'arrêt

Leur appel entraine un arrêt du programme et l'impression d'un message qui dépend du type de l'arrêt normal ou anormal.

NORMEXIT ( ).

entraîne un arrêt de type normal.

ABEXIT ( ).

entraîne un arrêt anormal.

2.9.9.3 Contrôle de message d'exécution

En cours d'exécution, un appel de sous-programme de gestion de la mémoire, ou la lecture de cent caractères consécutifs en mode ambigu (paragraphe 2.4.4.3 f) entraîne l'impression d'un message si un caractère de contrôle de message est chargé, si le caractère n'est pas chargé les mêmes opérations sont exécutées sans impression de message.

RUNMSS ( ) NO RUNMSS ( )

charge et décharge respectivement le caractère de contrôle de message qui est chargé au début de l'exécution,. Leur résultat est la ramification vide.

2.9.9.4 Procédure d'état d'exécution

DATE ( ).

Son résultat est un élément caténaire, dont le numéro de table vaut zéro et dont la chaîne de 8 caractères donne la date d'exécution du programme, en jour, mois et année.

TIME ( )

son résultat est analogue à celui de DATE ( ) mais il donne l'heure en heures, minutes, secondes.

CLOCK ( ). ICOUNT ( ) FREELIST ( )

ont respectivement pour résultat un élément numérique entier donnant, le nombre de millisecondes restant allouées par le systèmes, le nombre d'appels de l'éditeur avant l'exécution, le nombre de mots machines disponibles avant l'appel du sous-programme de gestion de la mémoire.

COLLECT ( ).

fonctionne comme FREELIST ( ) mais est appelé après le sous-programme de gestion de la !mémoire.

2.9.10 Tables des Variables Internes

NOM	VALEUR INITIALE	VALEUR PERMISE	UTILISE PAR	Affecté par ** Restaure **	
rem	0	pseudo-arbores-ce quelconque		* DIVIDE	Affecté du reste de la division de 2 entiers. (paragraphe 3.4.2)
dal	FAILURE	procédure	ALIST		désigne une procédure sans argument appelée par ALIST quand le résultat de ALIST est la ramification vide
gol	1	entier	Analyse		Evalué à chaque appel de l'analys (paragraphe 2.4.2.3)
ial	ramification vide	pseudo-arbores-cence quelconque	IDENT-analyse ($IDENT, n/)		valeur de l'information associée d'un élément caténaire crée par IDENT ou $IDENT, n/
ino	dépend du système	entier			désigne l'unité logique à partir d laquelle sera lue la chaîne traité d'entrée. Si sa valeur est changé elle est effective après la lecture de la chaîne en cours
rdm	1	entier impair tel que 1 ⩽ rdm ⩽ valeur de l'argument de RANDOM	RANDOM	* RANDOM	les appels successifs de RANDOM lui donnent des valeurs pseudo-aléatoires.

Variables internes liées aux opérations de sortie et à la création d'éléments

chr	FAILURE	Procédure	NORMSCN-IDENTSCN-FDINTSCN-FOINTSCN-XDINTSCN-XOINTSCN-	* STANDSON	désigne une procédure de traitement de caractères à un argument
lsr	STNDSCN1	Procédure	NORMSCN	* * * STANSCN	désigne une procédure de traitement de pseudo-arborescence à un argument
nir	FAILURE	Procédure	FDINTSCN-FOINTSCN-XDINTSCN-XOINTSCN-		désigne une procédure à un argument de traitement d'élément numérique négatif
inr	FDINTSCN	Procédure	STNDSCN1		désigne une procédure de traitement numérique, à un argument
flr	FAILURE	Procédure	STNDSCN1		désigne une procédure à deux arguments de traitement d'élément numérique flottant
isn	STANDSCN	Procédure	IDENT-CIDENT DECCON-OCTCON FLOATCON		désigne une procédure à deux arguments utilisée lors de la création d'éléments caténaires et numériques
pot	Ramification vide			DECCON-OCTCON FLOATCON - $DEC/, $OCT/, $FLOAT/.	Il est affecté du nombre des chiffres suivant le dernier caractère non numérique d'une séquence de caractère de la zone tampon - convertie en élément numérique.
fdl	0	Entier	XDINTSCN-XOINTSCN-		désigne le nombre de positions occupées par l'édition d'un élément numérique entier positif
lze	0	Entier	XDINTSCN XOINTSCN		caractère qui remplacera les zéros de tête dans l'édition d'un élément numérique entier positif.

Variables internes liées à la zone-P, à l'impression de messages

por	FAILURE	Procédure	PUTP-STANDPMC PUTPC		désigne une procédure appelée par PUTP, quand doit apparaître dans une zone-P, un code qui n'est pas associé à un symbole de base (code > 77_8)
pmr	Nombre de colonnes imprimantes	Entier	PUTP-STANDPMC PUTPC		désigne la première position hors de la zone standard de la zone-P
pin	1	entier tel que $1 \leqslant pin \leqslant pmr$	PUTP-STANDPMC PUTPC	*PUTP-*PUTPC OUTP *STANDPMC	désigne la position qu'occupera le caractère suivant placé par PUTP.
pmc	STANDPMC	Procédure	PUTP-STANDPMC PUTPC		désigne une procédure à un argument appelée par PUTP quand pin = pmr
pno	Dépend du système	entier	OUTP-STANDPMC DUMPV-DUMP1 DUMPALL		indique le numéro d'une unité logique utilisée pour l'impression de l'enregistrement de la zone-P et les messages du système. Si pno=0 l'impression n'est pas exécutée
sno	Ramification vide	entier et la ramification vide	Procédures du système avec messages		Permet l'impression des messages sur une unité autre que celle désignée par pno sauf si sno a pour valeur la ramification vide. sno=0 supprime tous les messages du système
dno	Ramification vide	Entier et ramification vide	DUMP1-PUMPV DUMPALL		Permet l'impression du résultat des procédures de dump sur une unité autre que celles désignées par pno et sno. dno = 0 supprime les impressions

Variables internes liées à la zone-C

cor	FAILURE	Procédure	PUTC-STANDCMC PUTPC		voir por
cmr		Entier $1 \leqslant cmr \leqslant 81$	PUTC-STANDCMC PUTPC		voir pmr
cin	1	Entier $1 \leqslant cin \leqslant cmr$	PUTC-STANDCMC PUTPC	*PUTC-PUTPC OUTC STANDCMC	voir pin
cmc	STANDCMC	Procédures	PUTC-STANDCMC PUTPC		voir pmc
cno		entier	OUTC-STANDCMC		voir pno

Variables internes liées à la zone-b

bmr	1	Entier $1 \leqslant bmr \leqslant 961$	PUTB		désigne la position du dernier bit représentant un nombre binaire.
bwl	1	Entier $1 \leqslant bwl \leqslant 960$	PUTB		désigne le nombre de bits utilisés pour représenter un nombre
bin	1	$1 \leqslant bin \leqslant bmr$	PUTB	*PUTB-OUTB	position du bit le plus significatif du nombre suivant
bmc	FAILURE	Procédure	PUTB		désigne une procédure à un argument appelée par PUTB quand bm + bwl > bmr.
bno		$1 \leqslant bno \leqslant 80$	OUTB		désigne l'unité logique affectée à la zone-C

CHAPITRE III

REALISATION D'UN SYSTEME ANALOGUE DANS UN AUTRE LANGAGE

On trouvera paragraphe 1 un rappel des caractéristiques "physiques" des informations traitées par COGENT : des mots sur un alphabet, des ramifications, des grammaires de Chomsky et les caractères des opérations de base : analyse d'un mot pour une grammaire et un axiome, synthèse de deux ramifications, analyse d'une ramification r pour une ramification s.

Le langage choisi pour réaliser un système analogue doit permettre une représentation de l'information et une définition des opérations de base.

On emploiera le langage A. T. F. gestion pour réaliser un tel système. On trouvera dans les ouvrages cités en appendice sous la référence [L. NOLIN] une définition de ce langage.

3.1 PROBLEMES POSES

3.1.1. Information traitée

Pour permettre une telle réalisation le langage choisi doit traiter des informations de type simple ou composé, et des ramifications. Une information de type simple représente un entier, un réel, une procédure, une chaîne de caractères. Une information de type composé (ou article) est une suite d'éléments dont chacun est une information de type simple ou un article. Les structures qui représentent ces informations ont une longueur fixe ou variable.

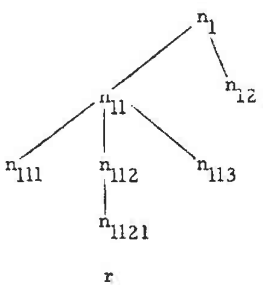
Parmi les représentations possibles d'une ramification, on choisit la représentation en liste, dont on trouvera une étude détaillée dans les paragraphes 3.1.1. et 3.2.2.

3.1.2 Représentation des pseudo-arborescences

On utilisera la même représentation pour les ramifications.

Elle est inspirée de la détermination d'une ramification par une matrice d'enchainement [IVERSON] . Il s'agit d'une matrice à trois colonnes une telle matrice représente une pseudo-arborescence (I, f) : les points de I sont les numéros de la ligne si i est l'un d'eux f (i) se trouve dans la première colonne de la ligne i ; le premier arc d'origine i aboutit à l'élément y de la deuxième colonne ; la troisième colonne contient le successeur de i dans sa famille s'il existe 0 sinon.

Exemple



r

n_1	3	0
n_{1121}	0	0
n_{11}	4	6
n_{111}	0	5
n_{112}	2	7
n_{12}	0	0
n_{113}	0	0

représentation de

En A. T. F. une telle matrice sera représentée par une file à une entrée (paragraphe 3.2.2.1).

Conclusion

Les caractéristiques dégagées dans le § 3.1.1. se trouvant réunies dans A. T. F. on choisit ce langage pour réaliser un système analogue.

3.2 REALISATION EN A. T. F.

3.2.1 Introduction

On se propose dans ce paragraphe de définir un certain nombre de procédures A. T. F. permettant de remplacer un programme COGENT par un programme A. T. F. Dans ce but on définit des procédures qui effectuent les opérations fondamentales de COGENT.

Dans tout ce qui suit on admet que les données traitées sont accessibles à partir de la mémoire interne. On ne précise donc pas les ordres d'entrée-sortie de plus, on admet que l'on dispose d'une procédure d'analyse dont les caractéristiques sont définies paragraphe 3.2.3. La colonne commentaire des feuilles de programmation n'étant jamais utilisée on la supprime dans la représentation de ces feuilles.

3.2.2 Structure et description des pseudo-arborescences

3.2.2.1 Généralités

La matrice d'enchainement qui détermine une pseudo-arborescence (I, f) est représentée par une file à une entrée. Dans une telle file un article défini par la partition ELEMENT représente une ligne de la matrice d'enchainement, les composants de l'article représentent les colonnes de cette matrice. Le premier et deuxième composant (TYPE et REPRESENTANT) correspondent à la première colonne, le troisième et quatrième composant (LIEN VERTICAL (LV) et LIEN HORIZONTAL (LH) correspondent respectivement au deuxième et troisième colonne de la matrice. On trouve dans REPRESENTANT un élément de l'alphabet fondamental et dans TYPE le type de cet élément.

Réciproquement Une telle file représente une pseudo-arborescence (I, f) où l'ensemble des points de I est l'ensemble des valeurs d'un index de la file.

Remarque :

La première racine du mot des racines d'un pseudo-arborescence est toujours rangé dans le premier élément de la file.

Tous les composants ont pour valeur un entier, la signification de l'entier dépend du type de l'élément représenté, on trouvera dans ce paragraphe une étude détaillée de cette signification, on rappelle que les éléments de type 3, 4, 5, 6 ne peuvent être que des feuilles d'une pseudo-arborescence.

Signification des Composants de ELEMENT

TYPE	REPRESENTANT	LIEN VERTICAL	LIEN HORIZONTAL
1 ou 2 grammatical	numéro de règle simple	vers le premier élément de la famille dont il est le prédecesseur	vers le successeur dans sa famille sa valeur est 0 s'il est le dernier de sa famille
3 indexé	valeur de l'index	0	vers le successeur dans sa famille
4 numérique	mode 0 entier 1 flottant	entrée de l'élement dans une table ou sera rangé la chaîne numérique	vers le successeur dans sa famille
5 caténaire	numéro de table	entrée de l'élement dans une table ou est rangée la chaîne et l'information associée à l'élément caténaire	vers le successeur dans sa famille
6 procédure	Rang de la procédure dans la table des procédures	0	vers le successeur dans sa famille

Il résulte de la représentation d'une pseudo-arborescence par une file, la déclaration suivante en A. T. F. d'un identificateur (PS ARB) ayant pour valeur une pseudo-arborescence :

TYPE SPECIFICATION

C	STATUT	NATURE	NOM	LONGUEUR	PARTITION NOM	PARTITION DANS	INDEX	PLACE
	DON/ MIXTE/ RES/ AUX	FILE	PSARB		ELEMENT	CATAL	I	MEMOIRE STOCK

TYPE PARTITION

C	PARTITION	CONDITION	JUSQUA	NB FOIS	NATURE	LONGUEUR	NOM
	ELEMENT				NATUREL NATUREL NATUREL NATUREL	1 3 4 4	TYPE REP LV LH

Représentation de la ramification vide

On lui donnera une forme structurée pour ELEMENT, il lui correspondra donc la déclaration suivante :

TYPE SPECIFICATION

STATUT	NATURE	NOM	LONGUEUR	PARTITION NOM	PARTITION DANS	INDEX	PLACE
AUX	ARTICLE	RAMVIDE		ELEMENT	CATAL		

L'apparition de RAMVIDE dans une feuille de définition en fait une constante.

TYPE DEFINITION

NOM	VALEUR
RAMVIDE	0, 0, 0, 0 ;

3.2.2.2 Procédures associées

SUCC (LISTE, X ; ; NBR)

1°) Introduction

Soit une file représentant une pseudo-arborescence (I, f), x un élément de I, SUCC (x) a pour résultat le nombre d'éléments qui suivent x (x compris)

dans la famille à laquelle il appartient si x représente une racine le résultat est l'entier 1.

Cette procédure est également valable pour une ramification si on convient que les racines forment une famille.

2°) Définition

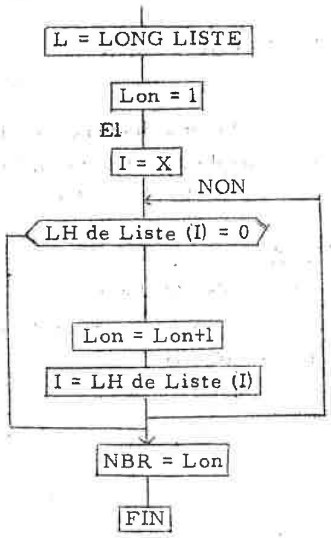
SUCC admet 2 paramètres données :

LISTE : file représentant une pseudo-arborescence

X : entier repérant un article de LISTE

1 paramètre résultat : NBR entier égal au nombre d'éléments qui suivent X (X compris) dans la famille à laquelle il appartient.

3°) Organigramme



4°) Programme

TYPE SPECIFICATION

STATUT	NATURE	NOM	LONGUEUR	PARTITION NOM	PARTITION DANS	INDEX NOM	PLACE
DON	FILE	LISTE		ELEMENT	CATAL	I	
DON	NAT	X	4				
RES	NAT	NBR					
AUX	NAT	LON	2				
AUX	INDEX	I					

```
TYPE TRAITEMENT

ETIQUETTE          INSTRUCTION
                   I := PREMIER ; LON := 1 ; L = LONG LISTE ;
        E1 :       AV I DE X ;
                   SI LH DE LISTE (I) = 0  VERS E2 ;,

                   LON := LON + 1 ;

                   I := PREMIER ; AV I DE LH DE LISTE (I) ;
                   VERS E1 ;
        E2 :       NBR := LON ;
                   FIN
```

LONFA (LISTE, ELT ; ; L)

1) Introduction

Soit une file représentant une pseudo-arborescence (I, f), x un élément de I, LONFA a pour résultat le nombre d'arcs de I dont x est l'origine. Cette procédure est également valable si LISTE représente une ramification.

2) Définition des paramètres

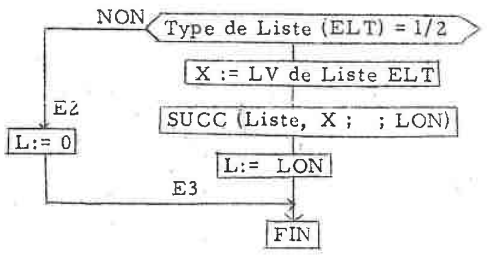
LONFA admet 2 paramètres données

LISTE file représentant une pseudo-arborescence

ELT entier repérant un article de LISTE

1 paramètre résultat : L entier égal à la longueur de la famille dont le prédecesseur est repéré par ELT dans LISTE.

3) Organigramme



4) Programme

```
TYPE REFERENCE

REFERENCES PROCEDURES EMPLOYEES
SUCC

TYPE SPECIFICATION
```

STATUT	NATURE	NOM	LONGUEUR	PARTITION NOM	PARTITION DANS	INDEX NOM	PLACE
DON	FILE	LISTE		ELEMENT	CATAL	I	
DON	NAT	ELT	4				
RES	NAT	I	4				
AUX	NAT	X	4				
AUX	NAT	LON	4				
AUX	INDEX	I					

```
TYPE TRAITEMENT

ETIQUETTE          INSTRUCTIONS
                   I := PREMIER ; AV I DE ELT ;
                   SI TYPE DE LISTE (I) ≠ 1  VERS E2 ;
                   X := LV DE LISTE (I) ; SUCC (LISTE, X ; ; LON)
                   L := LON ; VERS E3 ;
        E2 :       L := 0 ;
        E3 :       FIN ;
```

SPAR (LISTE, ELT ; ; RESULTAT)

1) Introduction

Soit une pseudo-arborescence (I, f) et x ∈ I. Les points de l'arborescence I appartenant à la descendance de x forment une sous-arborescence I' de racine x, qu'on peut orienter par la restriction de l'orientation de I.

Soit f' la restriction de f à I' : la pseudo-arborescence (I', f') sera appelée sous-pseudo-arborescence de (I, f) enracinée en x.

Soit une file représentant une pseudo-arborescence (I, f), x un élément de I, SPAR a pour résultat la sous-pseudo-arborescence enracinée en x.

2) Définition des paramètres

SPAR admet 2 paramètres donnée :

LISTE file représentant une pseudo-arborescence

ELT entier repérant un élément de liste

1 paramètre résultat : RESULTAT file représentant la sous-pseudo-arborescence dont la racine est repérée par ELT dans LISTE.

3) Généralités sur l'écriture de la procédure

Si ELT repère une feuille de la pseudo-arborescence représentée, RESULTAT est une file de longueur 1 réduite à l'élément repéré par ELT, sinon RESULTAT a pour valeur la sous-pseudo-arborescence dont la racine est repérée dans LISTE par ELT, tous les liens verticaux de la file RESULTAT ont pour valeur 1. TABLE est une pile à deux colonnes C1 et C2, on range dans C1 un entier permettant de repérer dans RESULTAT un élément qui n'est pas le dernier de sa famille et dans C2 un entier permettant d'accéder à son successeur dans LISTE, donc de donner une valeur au lien horizontal de l'élément de RESULTAT repère par C1 DE TABLE. La pile TABLE apparaissant dans la définition d'autres procédures à la même signification.

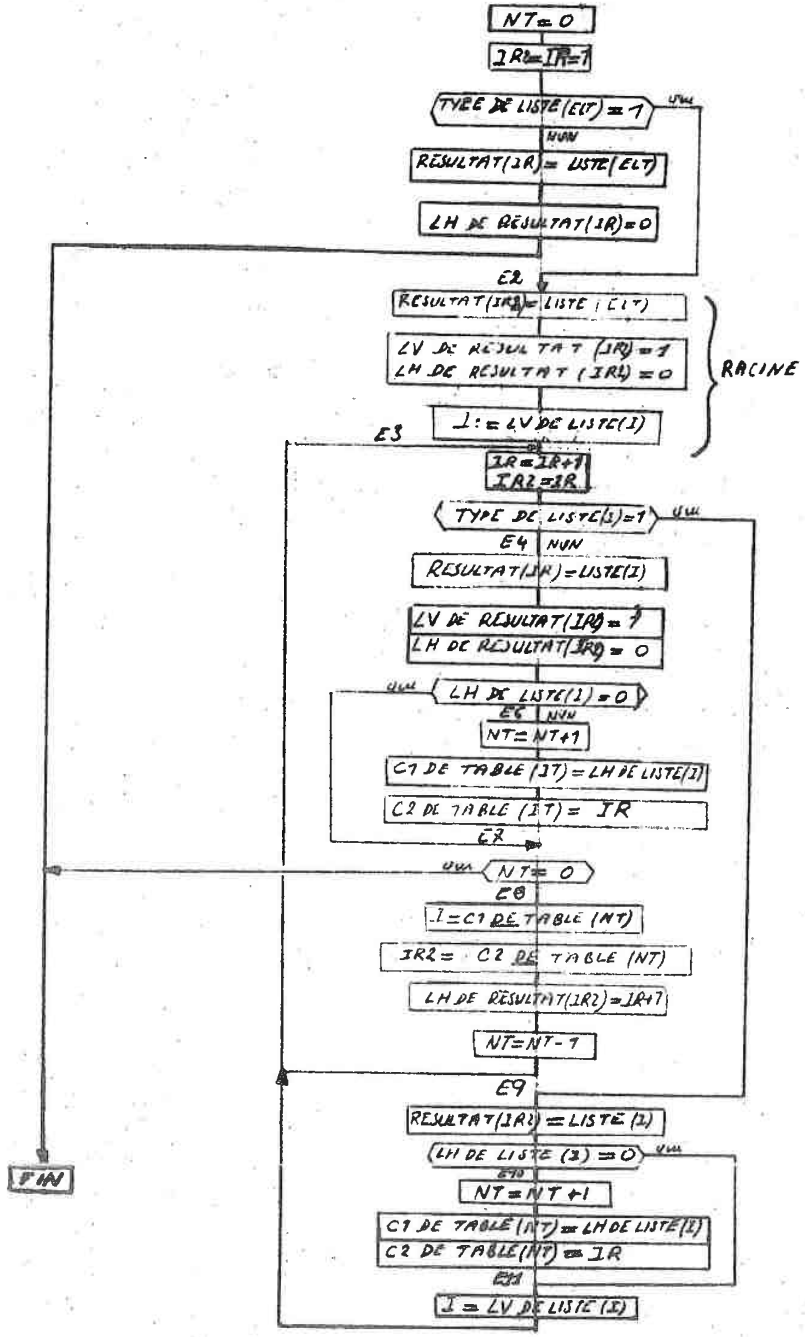
Cette procédure peut être définie récursivement en changeant le statut de RESULTAT, et en ajoutant aux arguments mixtes deux entiers ELT2 et NT repérant le dernier élément rangé dans RESULTAT et le nombre d'élément dans TABLE, l'appel de SPAR s'écrit alors

SPAR (LISTE, ELT ; ELT2, RESULTAT, NT ;)

cet appel remplace l'instruction E11 VERS E2 ; de plus ceci entraîne une modification des instructions qui précédent E1.

IR := 0 est remplacé par IR:= ELT2 et NT:=0 supprimé.

4) Organigramme



5) Programme

TYPE SPECIFICATION

STATUT	NATURE	NOM	LONGUEUR	PARTITION		NOM INDEX	PLACE
				NOM	DANS		
DON	FILE	LISTE		ELEMENT	CATAL	IL,IL2	
DON	NAT	ELT					
RES	FILE	RESULTAT		ELEMENT	CATAL	IR2	
AUX	FILE	TABLE		DEUX C	CATAL	IT	
AUX	NAT	IR					
AUX	NAT	NT					
AUX	INDEX	IL					
AUX	INDEX	IL2					
AUX	INDEX	IR2					
AUX	INDEX	IT					

TYPE TRAITEMENT

```
        IL := PREMIER ; AV IL DE ELT ; IR := 0 ; IR2 := PREMIER ; NT := 0 ;
        SI TYPE DE LISTE (IL) = 0 VERS E1 ;
        AV IR2 DE IR ;
        RESULTAT (IR2) := LISTE (IL) ;
        LH DE RESULTAT (IR2) := 0 ;  VERS E12 ;
E1 :    AV IR2 DE IR ; RESULTAT (IR2) := LISTE (IL) ;
        LV DE RESULTAT (IR2) := 1 ; LH DE RESULTAT (IR2) := 0 ;
E2 :    IL2 := PREMIER ; AV IL2 DE LV DE LISTE (IL) ;
        IL := IL2 ;
E3 :    IR := IR+1 ;
        SI TYPE DE LISTE (IL) = 1 VERS E9 ;
        IR2 := PREMIER ; AV IR2 DE IR ; RESULTAT (IR2) := LISTE (IL) ;
        LV DE RESULTAT (IR2) := 1  LH DE RESULTAT (IR2) := 0
        SI LH DE LISTE (IL) = 0 VERS E7 ;
        NT := NT+1 ; IT := PREMIER ; AV IT DE NT ;
        C1 DE TABLE (IT) := LH DE LISTE (IL) ;
        C2 DE TABLE (IT) := IR ;
E7 :    SI NT=0  VERS  E12 ;
        IT := PREMIER ; AV IT DE NT ;
        IL := PREMIER ; AV IL DE C1 DE TABLE (IT)
        IR2 := PREMIER ; AV  IR2 DE  C2 DE TABLE (IT) ; LH DE RESULTAT (IR2)
               := IR+1 ;
        NT := NT-1 ;  VERS E3 ;
E9 :    IR2 := PREMIER ; AV IR2 DE IR ;
        RESULTAT (IR2) := LISTE (IL) ; LV DE RESULTAT (IR2) := 1 ; LH DE RESUL
              TAT (IR2) := 0 ;
        SI LH DE LISTE (IL) =0  VERS E11 ;
        NT := NT+1 ; IT := PREMIER ; AV IT DE NT ;
        C1 DE TABLE (IT) := LH DE LISTE (IL) ; C2 DE TABLE (IR) := IR
E11 :   VERS E2 ;
E12 :   FIN
```


RAMI (LIS ; NBE, REP ;)

1°) Introduction

Soit r_1, r_2, ..., r_n des pseudo-arborescences, RAMI permet la création de la ramification r formée de r_1, r_2, ..., r_n.

2°) Définition

RAMI : admet 1 paramètre donnée

LIS : file représentant une des pseudo-arborescences, composant la ramification à créer

2 paramètres mixtes

REP : file représentant la ramification crée,

NBE : entier égal au nombre d'éléments de REP avant et après l'appel .

Cette procédure crée une ramification, elle est appelée pour chaque pseudo-arborescence composant la ramification.

3°) Généralités sur l'écriture de la procédure

Avant tout appel :

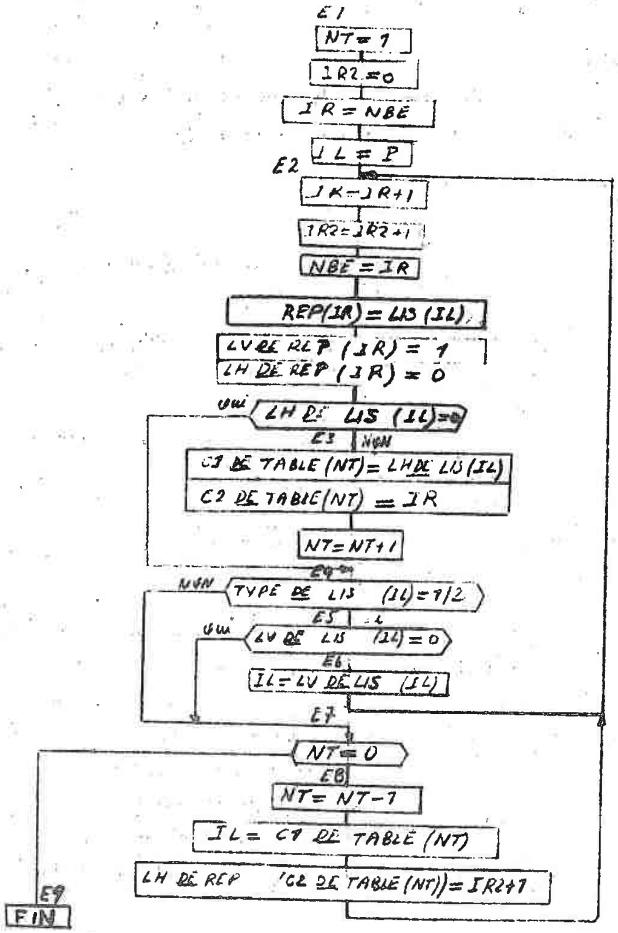
- LV DE REP (1) a pour valeur un entier égal au nombre de pseudo-arborescences composant la ramification.

- LV DE REP (j+1) (j+1 $\leqslant$ LV DE REP (1)) a pour valeur un entier égal à l'ordre de la jième pseudo-arborescence composant la ramification, si cette pseudo-arborescence est vide LV DE REP (j+1) a pour valeur 0.

Le lien vertical de tout élément de type 1 (grammatical) est positionné à 1, de plus les valeurs des liens horizontaux sont exprimées par rapport au premier élément de la file considérée, ceci pour faciliter une utilisation future de REP.

L'ordre d'une pseudo-arborescence est égal au nombre d'articles de la file ou longueur de la file, qui la représente, la longueur d'une file est obtenue par l'utilisation du symbole LONG.

4°) Organigramme



TYPE SPECIFICATION

STATUT	NATURE	NOM	LONGUEUR	PARTITION NOM	PARTITION DANS	INDEX NOM	PLACE
DON	FILE	LIS		ELEMENT	CATAL	IL	
MIXTE	NAT	NBE					
MIXTE	FILE	REP		ELEMENT	CATAL	IRX	
AUX	FILE	TABLE	100	ELEMENT	CATAL	NTX	
AUX	NAT	IR2	4				
AUX	NAT	NBE	4				
AUX	NAT	NT	4				
AUX	NAT	IR	4				
AUX	NAT	NBE	4				
AUX	INDEX	IL					
AUX	INDEX	IRX					
AUX	INDEX	NTX					

TYPE TRAITEMENT

```
E1 :    NT :=1 ; IR2 :=0 ; IR := NBE ; IL := P ;
E2 :    IR = IR+1 ;
        IRX:= IR APRES PREMIER ; NTX := NT APRES PREMIER ;
        IR2:= IR2+1 ; NBE = IR ;
        REP (IRX) := LIS (ILX) ;
        LV DE REP (IRX) := 1 ; LH DE REP (IRX) :=0 ;
        SI LH DE LIS (IL) = 0 VERS E4 ;
E3 :    C1 DE TABLE (NTX) := LH DE LIS (IL) ;
        C2 DE TABLE (NTX) := IR ;
        NT = NT+1 ;
E4 :    SI TYPE DE LIS (IL) = 1  VERS E5 ;
        SI TYPE DE LIS (IL) = 2  VERS E5 ;  VERS E7 ;
E5 :    SI LV DE LIS (IL) = 0  VERS E7 ;
E6 :    IL := LV DE LIS (IL) APRES PREMIER ;  VERS E2 ;
E7 :    SI NT=0  VERS E9 ;
E8 :    NT := NT-1 ;  NTX:= NT  APRES PREMIER ;
        IL := C1 DE TABLE (NTX) APRES PREMIER ;
        IRX := C2 DE TABLE (NTX) APRES PREMIER ; IR2 := IR2+1 ;
        LH DE REP (NTX) := IR2
        VERS E2 ;
E9 :    FIN ;
```

3.2.3 Représentation et déclaration des grammaires

3.2.3.1 Généralités

I - Procédure d'analyse

ANALYSE (G, M, AX ; ; PAR, APP)

1°) Introduction

L'analyse d'un mot α sur le vocabulaire d'une grammaire G se fait pour le couple (G, X) où X est un non terminal, elle a pour résultat une pseudo-arborescence.

2°) Définition

ANALYSE	admet 3 paramètres donnée :
G	file représentant une grammaire,
M	mot représentant une chaîne
AX	mot représentant un non terminal;
	2 paramètres résultat :
PAR	file représentant une pseudo-arborescence
APP	booléen

3°) Généralités

Si M appartient au langage engendré par la grammaire G et l'axiome AX, APP prend la valeur VRAI, PAR aura pour valeur la pseudo-arborescence résultat de l'analyse de M par G si elle est unique, l'une des pseudo-arborescences résultat (la première pseudo-arborescence créée) si la grammaire G est ambigüe.

Si M n'appartient pas au langage, APP prend la valeur FAUX.

Remarque :

Une variante consisterait à donner toutes les pseudo-arborescences résultat de l'analyse sous forme de ramification.

II - Une grammaire est représentée par une file, la file est structurée par une partition conditionnelle décrivant une règle, on différencie éléments terminaux et non terminaux composant une règle en testant la présence de parenthèses ouvrant (non suivi d'un $) et fermant.

Toutes les procédures du programme à l'exception des procédures standard (qui ne peuvent être des étiquettes) sont rangées dans une file (LISTE P). La file des procédures LISTEP correspond en A. T. F. à la description suivante :

TYPE SPECIFICATION

STATUT	NATURE	NOM	LONGUEUR	PARTITION		NOM INDEX	PLACE
				NOM	DANS		
DON	FILE	LISTEP	N	LIDPROC		1	

TYPE PARTITION

PARTITION	CONDITION	JUSQUA	NBFOIS	NATURE	LONGUEUR	NOM
LIDPROC				PROCEDURE		IDP

dans le corps d'une procédure on appelle une procédure du programme en écrivant IDP DE LISTE P (I) (α ; β ; γ).

On remplace donc tout identificateur de procédure associé à une règle par un entier égal au rang de la procédure dans LISTEP.

Une règle possède une représentation externe et une représentation interne. La représentation externe utilise la syntaxe de COGENT (paragraphe 2.4.1.1), la représentation interne de la même règle facilite son emploi en A. T. F.

3.2.3.2 Représentation des règles

1°) Représentation externe d'une règle

La règle est considérée comme une suite de MOT de longueur 1. Cette représentation est utile en entrée.

Sa description en A. T. F. est définie ci-après.

TYPE PARTITION

PARTITION	CONDITION	JUSQUA	NBFOIS	NATURE	LONGUEUR	NOM
REGLEX		".␣␣"		MOT	1	CAR

le drapeau est ".␣␣" et non pas "." pour ne pas confondre le point de fin de règle avec le caractère spécial point.

2°) Représentation interne d'une règle

. . . Le passage de la représentation externe à la représentation interne est défini par la procédure TRANSGR explicitée au paragraphe 3.2.3.4. On obtient la représentation interne en A.T.F., en appliquant la transformation suivante à une règle définie par la syntaxe de COGENT :

- on fait précéder la liste d'étiquettes par le nombre d'étiquettes qu'elle contient (NBT) ;
- on remplace un identificateur de procédure par son rang dans une liste de rangement des procédures (LISTE P) ;
- on fait suivre le signe = par le nombre de mots du second membre de règle (NB2) ;
- on fait précéder chaque mot par un entier ayant pour valeur la longueur du mot (NB3) ;
- on remplace les éléments du vocabulaire par des chaînes de caractères de longueur fixe ;

Cette représentation permet comme en COGENT d'associer une liste de procédures à une règle.

La règle en notation COGENT

P1 / P7 / P3 / (AB) = (BE) b (E) , (A) (B).

s'écrira avec les conventions précédentes :

3 1 7 3 (AB) = 23 (BE) b␣␣␣ (E␣), 2 (A␣) (B␣).

Remarque

Il résulte des conventions précédentes que tout élément du vocabulaire d'une grammaire, est un mot de longueur fixe. Cette restriction aurait pu être évitée en précisant pour chaque élément le nombre de caractères qui le composent.

P1 / P7 / P3 / (AB) = (BE) b (E), (A) (B) s'écrirait

3 1 7 3 4(AB) = 2 3 4 (BE) 1 b 3 (E), 2 3 (A) 3 (B).

cette nouvelle forme modifie très peu la description de la partition REGLE.

Il correspond à la représentation interne d'une règle la description A.T.F. suivante :

TYPE PARTITION

PARTITION	CONDITION	JUSQUA	NBFOIS	NATURE	LONGUEUR	NOM
REGLE				NAT	2	NB1
			NB1	NAT	2	RANG PROC
				MOT	4	P M R
				MOT	1	EGAL
				NAT	2	NB2
			NB2	PARTITION		MOT VOC ;
MOT VOC				NAT	2	NB3
			NB3	MOT	4	ELT VOC ;

On appellera une procédure associée à la Gième règle de la grammaire GRAMPRIM par les instructions :

I:= RANG PROC (R) DE REGLE DE GRAMPRIM (G) ;
IDP DE LISTE P (I) (α ; β ; γ) ;

Passage de la représentation externe d'une règle à sa représentation interne.

On utilise la procédure TRANSGR, on définit pour cela des procédures opérant successivement sur la liste des étiquettes (RECP) et le second membre de règle (RECV).

3.2.3.3 Représentation des grammaires

Une déclaration de grammaire primaire ou secondaire est définie par la description suivante.

TYPE SPECIFICATIONS

STATUT	NATURE	NOM	LONGUEUR	PARTITION		INDEX NOM	PLACE
				NOM	DANS		
	FILE	GRAMPRIM		REGLEX OU REGLE		G	

La représentation externe et la représentation interne d'une grammaire sont respectivement définies par REGLEX et REGLE.

Remarques

1) On rappelle que les grammaires sont également utilisées par les procédures d'analyse et de sortie ;

2) Dans la représentation des structures de longueurs variables l'option JUSQUA d'une feuille de type partition ne s'applique qu'à la description des éléments simples fondamentaux, de plus l'accès au dernier élément n'est pas direct ; cet accès est possible par exemple après transfert de la donnée dans une zone secondaire surdimentionnée ; l'intérêt de JUSQUA apparaît dans la représentation en entrée de l'information considérée comme un tout.

3.2.3.4 Procédures associées

RECP (PX1, M, NBP ; PX2, B ;)

1°) Introduction

Soit un mot α sur un vocabulaire V, un tableau T1 de mots sur V, un tableau entier T2, un entier i et un booléen B. RECP (T1, α, i ; T2, B) range dans T2 (i) le premier entier k s'il existe tel que T1 (k) = α B prend la valeur VRAI, sinon B prend la valeur FAUX ; dans les deux cas la procédure se termine. Cette procédure donne un représentation interne de la liste de procédures associée à une règle.

2°) Définition

RECP admet 3 paramètres donnée :

PX1 tableau de mots représentant les identificateurs de procédure du programme considérés comme des mots, leur ordre de rangement dans PX1 est celui de LISTEP (paragraphe 3.2.3.1)

M mot représentant un identificateur de procédure

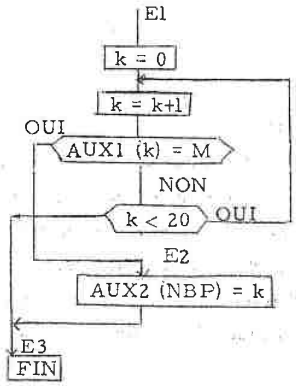
NBP entier représentant une position de PX2,

2 paramètres mixtes :

PX2 article représentant un tableau d'entiers, on rangera dans PX2 (NBP) un entier égal au rang de M dans PX1.

B booléen, il a la valeur VRAI si M est dans PX1, la valeur FAUX sinon.

3°) Organigramme



4° Procédure

TYPE SPECIFICATION

STATUT	NATURE	NOM	LONGUEUR	PARTITION NOM	PARTITION DANS
DON	ARTICLE	PX1		AUX1F	
DON	MOT	M	10		
DON	NAT	NBP	2		
MIXTE	ARTICLE	PX2		AUX2F	
MIXTE	BOOL	B	1		
AUX	NAT	K	2		

TYPE PARTITION

PARTITION	CONDITION	JUSQUA	NBFOIS	NATURE	LONGUEUR	NOM
AUX1F			20	MOT	10	AUX1
AUX2F			20	NAT	2	AUX2

TYPE TRAITEMENT

```
        K:=0 ; B:=FAUX
E1 :    K:=K+1 ;

        SI K < 20  VERS E1 ; VERS E3 ;
E2 :    AUX2 (NBP) := K ;  B:= VRAI  ;
E3 :    FIN ;
```

RECV (PCAR ; PV, I, I2, NV, NET, NE, PX2 ;)

1°) Introduction

Cette procédure permet le passage de la représentation externe à la représentation interne d'un second membre de règle.

2°) Définition

RECV admet 1 paramètre donnée :

PCAR article qui est un tableau de MOTS de longueur 1, il représente un mot sur un vocabulaire V (analogue au vocabulaire défini paragraphe 2.1.1) c'est-à-dire la forme externe d'une règle.

7 paramétres mixtes :

PV article représentant un tableau de mots de longueur 10,
I, I2, NV, NE, NET des entiers,
PX2 article représentant un tableau d'entiers.

La procédure recherche dans PCAR les chaînes de caractères commençant par "(" et terminées par ")" (éléments non terminaux) les chaînes de caractères entre ")" et "(" (terminaux) et range ces chaînes dans les positions successives de PV reperées par NE ; pour chaque virgule ou point non compris entre "(" et ")" NV est incrémenté de 1, NV précise le nombre de mots du second membre de régle, puis NE est rangé dans T2 (NV) et remis à zéro, NE précise la longueur du NEième mot, les entiers I et I2 repérent respectivement une position et la position suivante de PCAR.

4°) Organigramme (page 106)

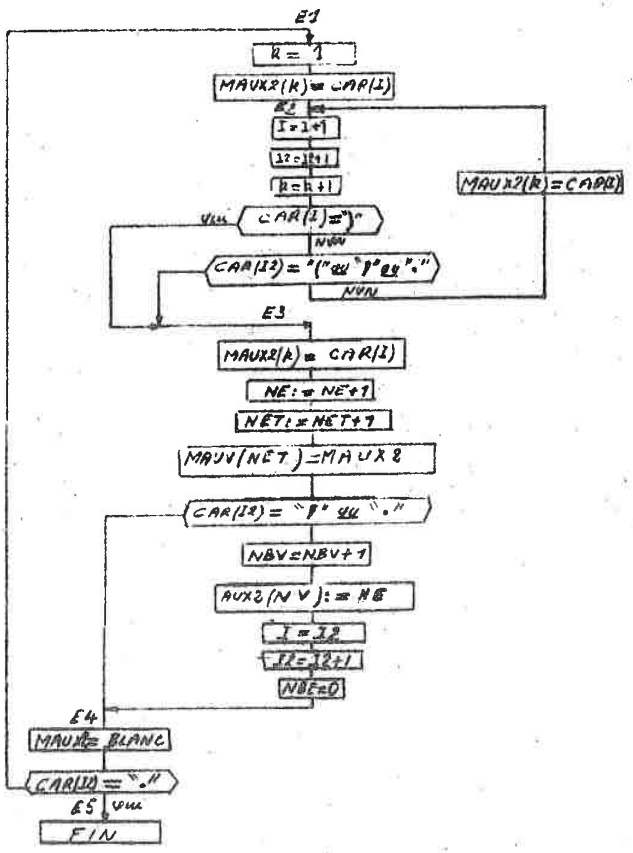
5°) Procédure

TYPE SPECIFICATION

STATUT	NATURE	NOM	LONGUEUR	PARTITION NOM	PARTITION DANS
DONNEE	ARTICLE	PCAR		CARF	
MIXTE	ARTICLE	PV		MAUVF	
MIXTE	NAT	I	2		
MIXTE	NAT	I2	2		
MIXTE	NAT	NV	2		
MIXTE	NAT	NET	2		
MIXTE	NAT	NE	2		
MIXTE	ARTICLE	PX2		AUX2F	
AUX	ARTICLE	PMX2		MAUX2F	
AUX	NAT	K	2		
AUX	MOT	BLANC	10		

TYPE PARTITION

PARTITION	CONDITION	JUSQUA	NBFOIS	NATURE	LONGUEUR	NOM
CARF			200	MOT	1	CAR
MAUVF			20	MOT	10	MAUV
AUX2F			20	NAT	2	AUX2
MAUX2F			10	MOT	1	MAUX2



TYPE DEFINITION

NOM	VALEUR
BLANC	" "

TYPE TRAITEMENT

```
E1 :    K:=1 ;
        MAUX2 (K):= CAR (I) ;
E2 :    I:=I+1 ; I2 :=I2+1 ; K:=K+1 ;
        SI CAR (I) = ")" VERS E3 ;
        SI CAR (I2) = "("VERS E3 ; SI CAR (I2)=")" VERS E3 ;
        SI CAR (I2) = "." VERS E3 ;
        MAUX2 (K) := CAR (I) VERS E2 ;
E3 :    MAUX2 (K) := CAR (I) ; NE := NE+1 ; NET:=NET+1 ;
        MAUV (NET) := MAUX2 ;
        SI CAR (I2) = ")" VERS E4 ; SI CAR (I2) = "." VERS E4 ;
        NV := NV+1 ; AUX2 (NV) := NE ; I:=I2 ; I2:=I2+1 ; NE:=0 ;
E4 :    MAUX2 := BLANC ;
        SI CAR (I2) = "." VERS E5 ;
        RECV (PCAR ; PV, I, I2, NV, NET, NE, PX2 ;) ;
E5 :    FIN ;
```


TRANSG (PCAR, PX1 ; B ; REG)

1°) Introduction

TRANSG permet de passer de la représentation externe d'une règle à sa représentation interne.

2°) Définition

TRANSG admet 2 paramètres donnée :

PCAR article représentant un tableau de mots de longueur 1 c'est-à-dire la représentation externe d'une règle

PX1 article représentant un tableau de mots de longueur 10 dans lequel sont rangés les identificateurs de procédure (considérés comme des mots).

1 paramètre mixte

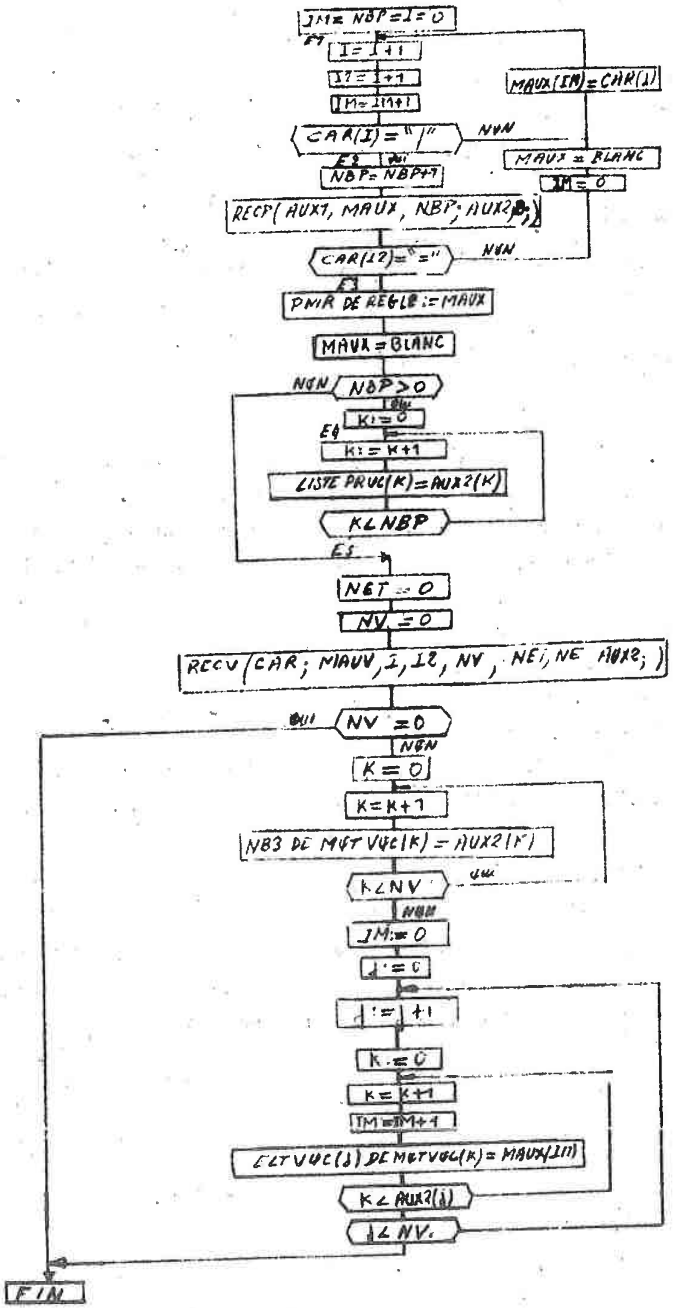
BOOLEEN il a la valeur VRAI si une représentation interne de la règle est possible, la valeur FAUX sinon la procédure se termine alors

1 paramètre résultat

REG article structuré par la partition REGLE (définie paragraphe 3.2.3.2), REG défini la représentation interne de la règle.

3°) Généralités

Les appels successifs de RECP transforment la liste d'étiquettes de la règle en une liste d'entiers et donnent une valeur à NB1, RANG PROC composants de REG. RECV s'appelant récursivement donne une valeur à PMR, NB2, MOT VOC, NB3, ELTVOC composants de REG.



5°) Programme

TYPE REFERENCE

RECV, RECP ;

TYPE SPECIFICATIONS

STATUT	NATURE	NOM	LONGUEUR	PARTITION		INDEX NOM	PLACE
				NOM	DANS		
DON	ARTICLE	PCAR		CARF	CATAL		
DON	ARTICLE	PX1		AUX1F	CATAL		
MIXTE	BOOLEEN	B					
RESULTAT	ARTICLE	REG		REGLE	CATAL		
AUX	NAT	IM	3				
AUX	NAT	NBP	3				
AUX	NAT	I	3				
AUX	NAT	I2	3				
AUX	NAT	K	3				
AUX	NAT	J	3				
AUX	NAT	NE	3				
AUX	NAT	NET	3				
AUX	NAT	NV	3				
AUX	ARTICLE	PX2		AUX2F	CATAL		
AUX	ARTICLE	PMAUX		MAUX2F	CATAL		
AUX	MOT	BLANC	10				

TYPE TRAITEMENT

```
      IM:=0 ; NBP:=0 ; I:=0 ;
E1 :  I:= I+1 ; I2:=I+1 ; IM:=IM+1 ; B:= FAUX ;
      SI CAR (I) = "/" VERS E2 ; MAUX2 (IM) := CAR(I) ;
E2 :  NBP := NBP+1 ;
      RECP (AUX1, PMAUX2, NBP ; PX2, B ;) ; SI B VERS E21 ; VERS E9 ;
E21 : SI CAR (I2) = "=" VERS E3 ; IM:=0 ; MAUX2 := BLANC ; VERS E1 ;
E3 :  PMR DE REG := MAUX2 ; MAUX2 := BLANC ; NB1 DE REG := NBP ;
      SI NBP = 0 VERS E5 ;
      K:=0 ;
E4 :  K:=K+1 ; RANG PROC (K) DE REG := AUX2 (K) ;
      SI K < NBP VERS E4 ;
E5 :  NET :=0 ; NV:=0 ;
      RECV (CAR ; MAUV, I, I2, NV, NET, NE , AUX2 ;)
      SI NET=0 VERS E9 ;
      K:=0 ;
```

```
E6 :  K:=K+1 ;
      NB3 DE MOT VOC (K) DE REG :=  UX2 (K) ;
      SI K < NV  VERS E6 ;
      IM:=0 ; J:=0 ;
E7 :  J:=J+1 ; K:=0 ;
E8 :  K:=K+1 ; IM:=IM+1 ; ELT VOC (J) DE MOT VOC (K) DE REG := MAUV(IM) ;
      SI K < AUX2 (J) VERS E8 ;
      SI J < NV  VERS E7 ;
E9 :  FIN ;
```

3.2.4 Procédures de définition des opérations fondamentales

3.2.4.1 Affectation simple

AFFECTATION (CLAS, CAR, N ; IDE, TAB ;)

1°) Introduction

AFFECTATION exécute l'instruction d'affectation.

2°) Définition

AFFECTATION admet 3 paramètres donnée :

CLAS — entier représentant le type d'un élément de l'alphabet fondamental

CAR — mot représentant une chaîne de caractères associée à un élément numérique ou caténaire

N — entier repère une entrée dans une table ou un numéro de procédure

2 paramètres mixtes :

IDE — article représentant la variable à affecter elle est structurée par la partition ELEMENT définie paragraphe 3.2.2.1

TAB — file d'entrées représentant la table de rangement des chaînes d'éléments numériques et caténaires.

L'affectation donne une valeur aux composants TYPE et REPRESENTANT de IDE ; l'affectation ne concerne que les éléments de type 4 (numérique), 5 (caténaire) et 6 (procédure) de l'alphabet fondamental.
TYPE a la valeur de CLASS, REPRESENTANT la valeur de N.

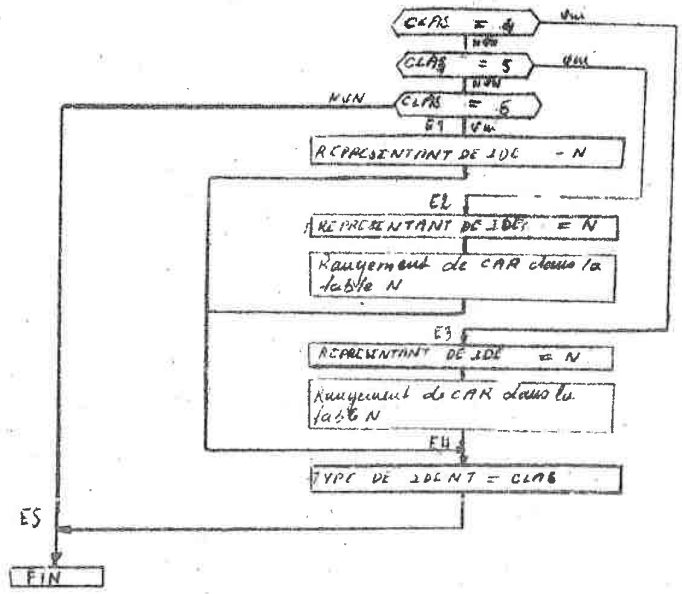
3°) Généralités sur l'écriture de la procédure

Un élément numérique est de type flottant si CAR est une chaîne numérique contenant un point, il est de type entier sinon.

TAB — **est une file à deux entrées : le numéro de table est associé à la première entrée, le rang dans cette table est associé à la deuxième entrée.**

Pour accéder dans la table 5 au rang 123 on donnera à N la valeur 50 123.

4°) Organigramme



5°) Programme

TYPE SPECIFICATION

STATUT	NATURE	NOM	LONGUEUR	PARTITION NOM	PARTITION DANS	INDEX. NOM	PLACE
DON	NAT	CLAS	1				
DON	MOT	CAR	20				
DON	NAT	N	5				
MIXTE	ARTICLE	IDE		ELEMENT	CATAL		
MIXTE	FILE	TAB		NUCA		I J	
AUX	INDEX	I					
AUX	INDEX	J					
AUX	NAT	K	2				
AUX	ARTICLE	PCAR2		ZCAR2			
AUX2	NAT	N2	3				

TYPE PARTITION

PARTITION	CONDITION	JUSQUA	NBFOIS	NATURE	LONGUEUR	NOM
NUCA				MOT	20	NU
				MOT	20	CA
ZCAR2			20	MOT	1	CARAC

TYPE TRAITEMENT

```
        CARAC:=CAR ;
        SI CLAS = 4  VERS E3 ;
        SI CLAS = 5  VERS E2 ;
        SI CLAS = 6  VERS E1 ;  VERS E5 ;
E1 :    REPRESENTANT DE IDE := N ;  VERS E4 ;
E2 :    N2 := N MOD 10 000 ; REPRESENTANT DE IDE :=N2 ;
        I:= PREMIER ; AV I DE N2 ; N2:= N2*10 000 ; N2:=N-N2 ;
        J:= PREMIER ; AV J DE N2 ; CA DE TAB (I,J):= CAR ; VERS E4 ;
E3 :    K:=0 ;
E31:    K:=K+1 ; SI K=21  VERS E23 ;
        SI CARAC (K) := "." VERS E32 ; VERS E31 ;
E32:    REPRESENTANT DE IDE :=1 ;  VERS E34 ;
E33:    REPRESENTANT DE IDE :=0 ;
E34:    N2:=N MOD 10 000 ; I:=PREMIER ; AV I DE N2 ; N2:=N-N2 ;
        J:= PREMIER ; AV J DE N2 ; NU DE TAB (I,J):= CAR ;
E4 :    TYPE DE IDE := CLAS ;
E5 :    FIN ;
```


3.2.4.2 Affectation d'analyse

OP ANALYSE (TEST, MOD, N ; VAR ; REP)

1°) Introduction

Elle exécute l'instruction d'affectation d'analyse définie paragraphe 2.6.5.

2°) Définition

OP ANALYSE admet 2 paramètres donnée :

TEST et MOD qui représentent respectivement les pseudo-arborescences valeur des expressions test et modèle et un entier N dont la valeur est le nombre d'identificateurs de la liste de variable, y compris les éléments vides.

1 paramètre mixte :

VAR file avant éxécution tous les éléments de VAR sont positionnés à zéro, sauf les éléments correspondant à des éléments vides de la liste de variable positionnés à 9999.

VAR (j) correspond au jième élément de < LISTE DE VARIABLES >.

Après éxécution on y range les entiers qui repèrent dans la file TEST les racines des pseudo-arborescences résultat de l'opération de synthèse, on range dans VAR(i) l'entier qui repère la racine de la $i^{ième}$ des pseudo-arborescences qui composent le résultat de l'analyse.

1 paramètre résultat :

REP booléen indique le succés (valeur VRAI) ou l'échec (Valeur FAUX) de l'opération d'analyse, la procédure se termine dés que BOOL à la valeur FAUX.

3°) Généralités sur l'écriture de la procédure

On obtient la $i^{ième}$ pseudo-arborescence résultat de l'analyse par l'appel de la procédure SPA (paragraphe 3.2.2.2) dont les arguments donnée sont la file TEST et l'entier valeur de VAR(i).

On admet que la longueur des files traitées ne sera pas supérieure à 9999. On rappelle qu'au $i^{ième}$ élément non vide de < LISTE DE VARIABLES > doit correspondre un seul index de valeur i, dans le mot des feuilles de la pseudo-arborescence représentée par TEST. On respecte cette condition en vérifiant que l'élément correspondant de VAR auquel on donne une valeur est bien positionné à zéro, la valeur des éléments positionnée à 9999 ne sera pas modifiée.

La procédure OP ANALYSE compare un élément de la pseudo-arborescence modèle (x) à l'élément correspondant de la pseudo-arborescence test (y), la comparaison se poursuit :

a) Si X est un élément caténaire, procédure, ou la ramification vide, Y est le même élément ;

b) Si X est un élément numérique, Y est un élément numérique de même mode et même valeur ;

c) Si X est un élément grammatical, Y est un élément grammatical avec le même numéro de règle, de plus les familles dont ils sont les prédécesseurs doivent être de même longueur.

d) Si X est un élément indexé d'index i, la sous-pseudo-arborescene de test dont l'enracinement est repéré par Y est la valeur de la $i^{ième}$ variable, on range Y dans VAR (i).

Sinon la procédure échoue, on traduit l'échec en donnant à REP la valeur FAUX et en quittant la procédure.

Remarque

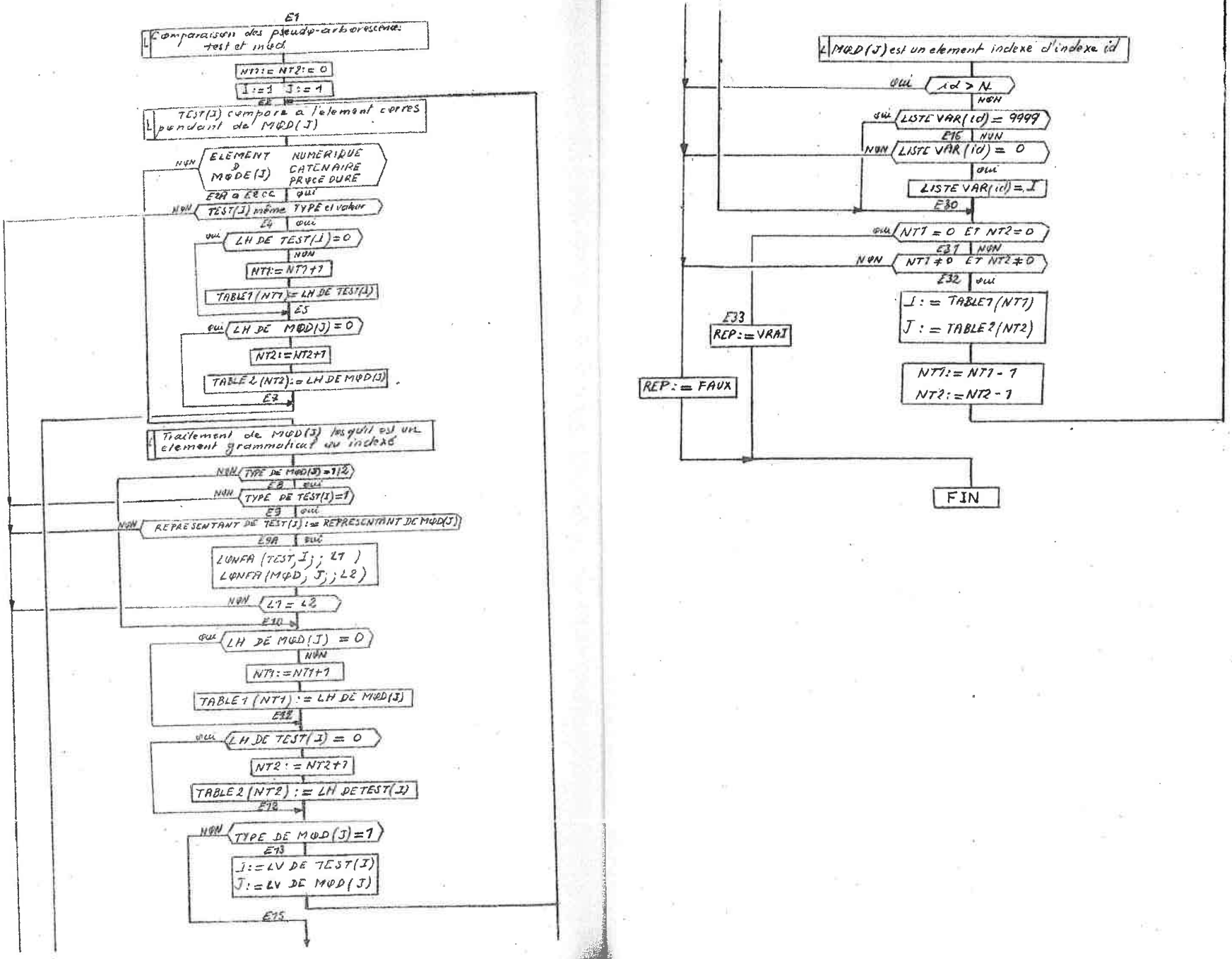
L'instruction de saut conditionnel de COGENT

étiquette IF Expression test = / Expression modèle, Liste de variable sera remplacée en A. T. F. par les deux instructions suivantes :

```
OP ANALYSE    (TEST, MOD, N ; VAR ; REP) ;
SI REP        VERS étiquette ;
```

ou les files TEST et MOD sont respectivement la représentation des pseudo-arborescences valeur des expressions test et modèle. N et VAR jouant le rôle de LISTE DE VARIABLE. . .

4°) Organigramme


E1
Comparaison des pseudo-arborescences test et indd
NT1 := NT2 := 0
I := 1 J := 1
E2
TEST(I) compare a l'element correspondant de MOD(J)
NON
ELEMENT DE MODE(J)
NUMERIQUE
CATENAIRE
PROCEDURE
E2A a E2 cc
OUI
NON
TEST(I) même TYPE et valeur
E4
OUI
OUI
LH DE TEST(I) = 0
NON
NT1 := NT1+1
TABLE1(NT1) := LH DE TEST(I)
E5
OUI
LH DE MOD(J) = 0
NT2 := NT2+1
TABLE2(NT2) := LH DE MOD(J)
E7
Traitement de MOD(J) lorsqu'il est un element grammatical ou indexé
NON
TYPE DE MOD(J) = 1 ou 2
E8
OUI
NON
TYPE DE TEST(I) = T
E9
OUI
NON
REPRESENTANT DE TEST(I) := REPRESENTANT DE MOD(J)
E9A
OUI
LONFA (TEST, I ; L1)
LONFA (MOD, J ; L2)
NON
L1 = L2
E10
OUI
LH DE MOD(J) = 0
NON
NT1 := NT1+1
TABLE1(NT1) := LH DE MOD(J)
E12
OUI
LH DE TEST(I) = 0
NT2 := NT2+1
TABLE2(NT2) := LH DE TEST(I)
E13
NON
TYPE DE MOD(J) = 1
E13
I := LV DE TEST(I)
J := LV DE MOD(J)
E15
MOD(J) est un element indexe d'index id
OUI
id > N
NON
OUI
LISTE VAR(id) = 9999
E16
NON
NON
LISTE VAR(id) = 0
OUI
LISTE VAR(id) := I
E30
OUI
NT1 = 0 ET NT2 = 0
E31
NON
NON
NT1 ≠ 0 ET NT2 ≠ 0
E32
OUI
I := TABLE1(NT1)
J := TABLE2(NT2)
NT1 := NT1 - 1
NT2 := NT2 - 1
E33
REP := VRAI
REP := FAUX
FIN

5°) Procédure

TYPE REFERENCE

LONFA

TYPE SPECIFICATION

STATUT	NATURE	NOM	LONGUEUR	PARTITION NOM	PARTITION DANS	INDEX NOM	PLACE
DON	FILE	TEST		ELEMENT	CATAL	IE	
DON	FILE	MOD		ELEMENT	CATAL	IA	
DON	NAT	N					
MIXTE	FILE	VAR		LIVAR		IV	
RES	BOOL	REP	1				
AUX	NAT	L1	4				
AUX	NAT	L2	4				
AUX	NAT	NT1	4				
AUX	NAT	NT2	4				
AUX	NAT	ELE	4				
AUX	NAT	ELA	4				
AUX	INDEX	IE					
AUX	INDEX	IA					
AUX	NAT	ID	4				
AUX	FILE	TABLE1	100	UN LIEN	CATAL	IT1	
AUX	FILE	TABLE2	100	UN LIEN	CATAL	IT2	
AUX	INDEX	IT1					
AUX	INDEX	IT2					
AUX	INDEX	IV					

TYPE PARTITION

PARTITION	CONDITION	JUSQUA	NBFOIS	NATURE	LONGUEUR	NOM
LIVAR			20	NAT	4	LISTEVAR

TYPE TRAITEMENT

```
E1 :    NT1 :=0 ; NT2 := 0 ;
        ELE :=0 ; ELA :=0 ;
ELA:    IE:= ELE APRES PREMIER ; IT1:= NT1 APRES PREMIER ;
        IA:= ELA APRES PREMIER ; IT2 := NT2 APRES PREMIER ;
E2 :    SI TYPE DE MOD (IA) = 4  VERS E2A ;
        SI TYPE DE MOD (IA) = 5  VERS E2B ;
        SI TYPE DE MOD (IA) = 6  VERS E2C ;
        SI TYPE DE MOD (IA) = 0  VERS E2D ;  VERS E7 ;
```

```
E2A:    SI TYPE DE TEST (IE) = 4  VERS E2AA ;  VERS E34 ;
E2B:    SI TYPE DE TEST (IE) = 5  VERS E2AA ;  VERS E34 ;
E2C:    SI TYPE DE TEST (IE) = 6  VERS E2CC ;  VERS E34 ;
E2D:    SI TYPE DE TEST (IE) = 0  VERS E4 ;  VERS E34 ;
E2AA:   SI LV DE MOD (IA) = LV DE TEST (IE) ET REPRESENTANT DE MOD (IA)
          = REPRESENTANT DE TEST (IE)  VERS E4 ;  VERS E34 ;
E2CC:   SI REPRESENTANT DE MOD (IA) = REPRESENTANT DE TEST (IE)  VERS E4 ;
        VERS E34 ;
E4 :    SI LH DE TEST (IE) = 0  VERS E5 ;
        TABLE1 (IT1) := LH DE TEST (IE) ; NT1:=NT1+1 ;
E5 :    SI LH DE MOD (IA) = 0  VERS E30 ;
        TABLE2 (IT2) := LH DE MOD (IA) ; NT2 :=NT2+1 ;
        VERS E30 ;
E7 :    SI TYPE DE MOD (IA) =1  VERS E8 ;  VERS E10 ;
E8 :    SI TYPE DE TEST (IE) =1  VERS E9 ;  VERS E34 ;
E9 :    SI REPRESENTANT DE MOD (IA) = REPRESENTANT DE TEST (IE)  VERS E9A ;
        VERS E34 ;
E9A:    LONFA (TEST,ELE ; ; L1) ;
        LONFA (MOD, ELA ; ; L2) ;
        SI L1 = L2  VERS E10 ;  VERS E34 ;
E10:    SI LH DE MOD (IA) =0  VERS E12 ;
        TABLE1 (IT1) := LH DE TEST (IE) ; NT1:=NT1+1 ;
        SI LH DE TEST (IE) =0  VERS E12 ;
        TABLE2 (IT2) := LH DE MOD (IA) ; NT2:= NT2+1 ;
E12:    SI TYPE DE MOD (IA) = 1  VERS E14 ;  VERS E15 ;
E14:    ELE:= LV DE TEST (IE) ; ELA := LV DE MOD (IA) ;
        VERS E1A ;
E15:    ID:= LV DE MOD (IA) ;
        SI ID > L  VERS E34 ;
        SI LISTE VAR (ID) = 9999  VERS E30 ;
        IV := ID APRES PREMIER ;
        LISTE VAR (IV) := ELE ;
E30:    SI NT1=0 ET NT2 = 0  VERS E33 ;
        SI NT1≠0 ET NT2 ≠ 0  VERS E32 ; VERS E33 ;
        NT1 := NT1-1 ; NT2 := NT2-1 ;
        IT1:= NT1 APRES PREMIER ; IT2:= NT2  APRES PREMIER ;
        ELE := TABLE1 (IT1) ; ELA := TABLE2 (IT2) ;
        VERS E1A ;
E33:    REP:= VRAI  VERS E35 ;
E34:    REP:= FAUX ;
E35:    FIN ;
```


3.2.4.3 Affectation de synthèse

OP SYNTHESE (MOD, REP ; ; RES, BOL)

1°) Introduction

Elle exécute l'instruction d'affectation de synthèse définie paragraphe 2.6.4.

2°) Définition

OP SYNTHESE admet 2 paramètres données :
les files MOD et REP la file MOD représente la pseudo-arborescence valeur de l'expression modèle, dans la file REP est rangée la ramification valeur des expressions de la liste de synthèse.

2 paramètres résultat :
la file RES et le booléen BOL la pseudo-arborescence résultat de la synthèse est rangée dans RES. La valeur du paramètre résultat BOL indique le succès (valeur VRAI) ou échec (Valeur FAUX) de l'opération de synthèse. La procédure se termine dès que BOL à la valeur FAUX.

3°) Généralités sur l'écriture de la procédure

REP est construit par l'appel de la procédure RAMI définie paragraphe 3.2.2.2.

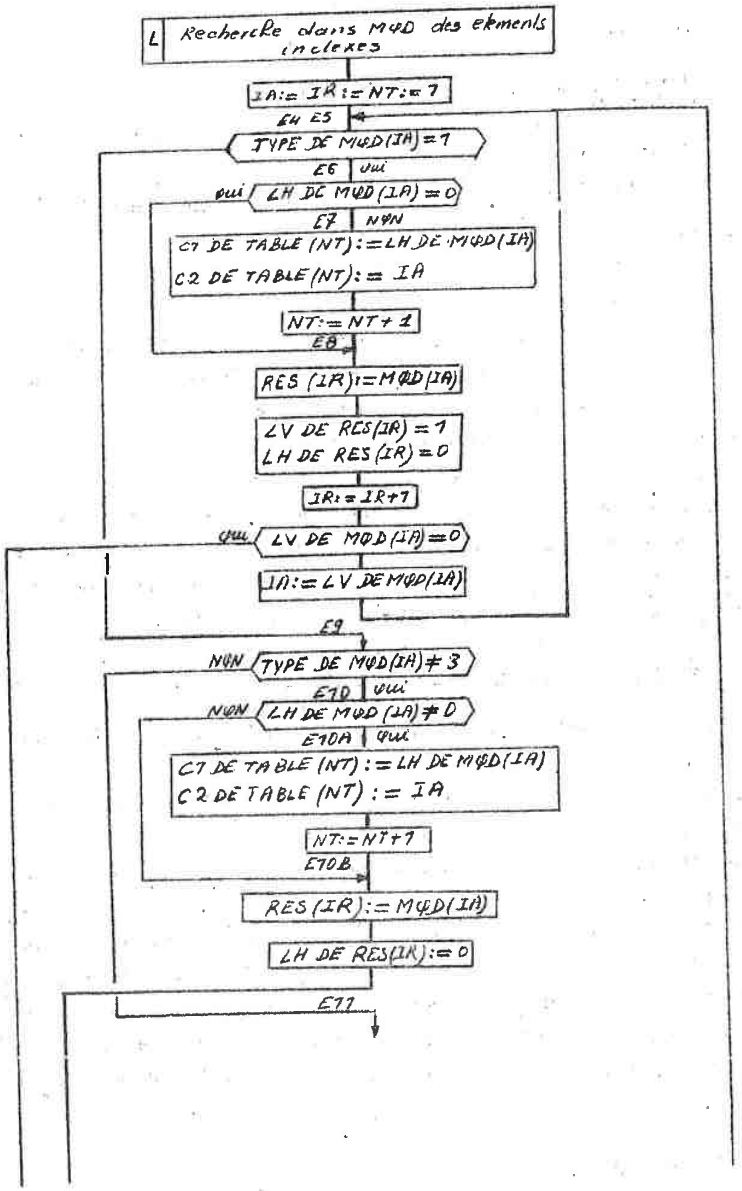
La file RES est obtenue à partir de la file MOD en appliquant les règles suivantes :

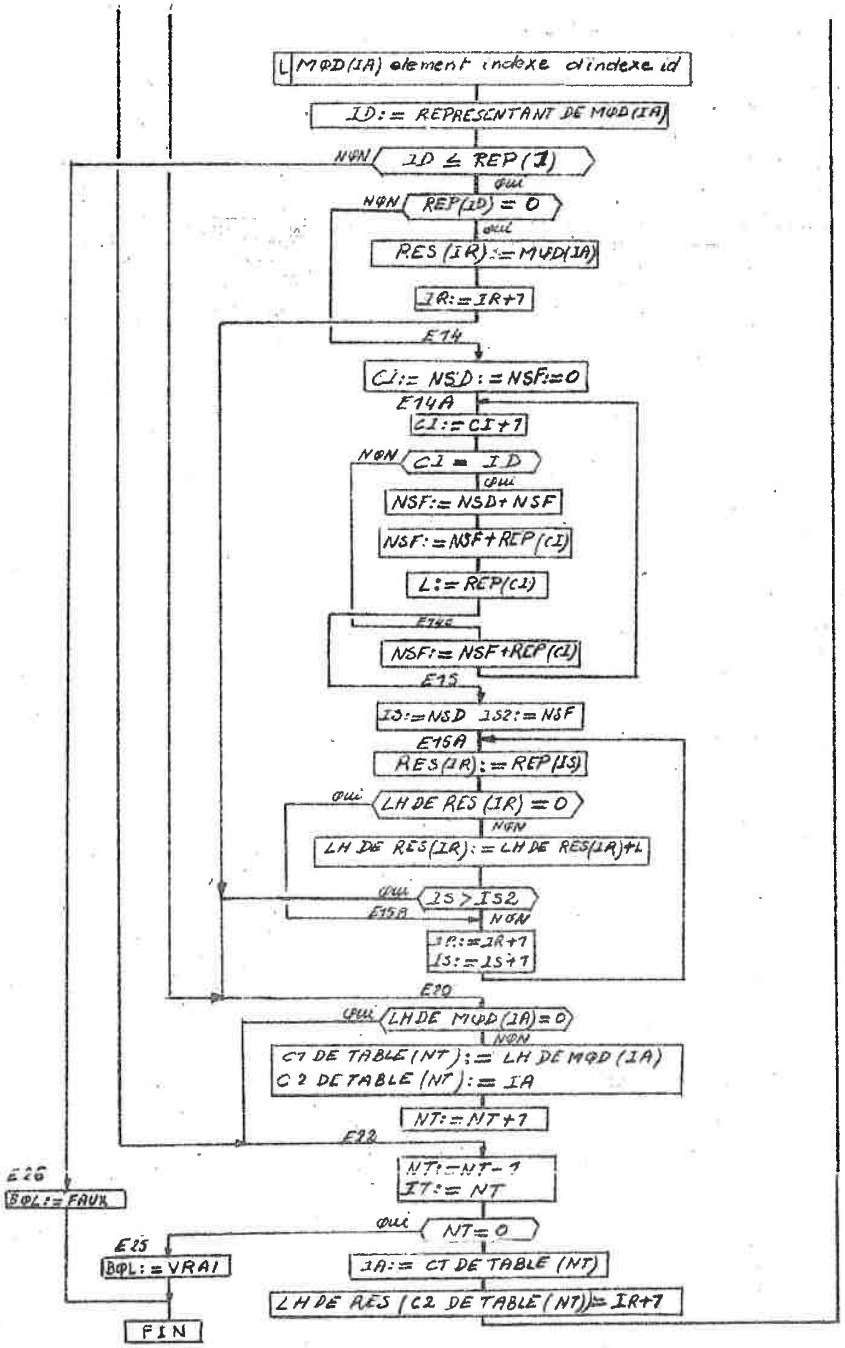
a) Si l'élément n'est pas un élément indexé , il est rangé dans RES ; après modification de ses liens horizontaux et verticaux.

b) Si l'élément est un élément indexé, d'index i, on le range dans RES, si REP (i+1) ≠ 0 on range dans RES la $i^{ième}$ pseudo-arborescence de REP en modifiant les liens horizontaux et verticaux de chacun de ses éléments, on reprend l'étape a, sinon on reprend l'étape a.

Si i > REP (1) où est rangé la longueur de la liste de synthèse il y a erreur.

4°) Organigramme


L Recherche dans MΦD des elements indexes
IA:= IR:= NT:= 7
E4 E5
TYPE DE MΦD(IA) = 7
E6 oui
oui LH DE MΦD(IA) = 0
E7 NON
C1 DE TABLE(NT):= LH DE MΦD(IA)
C2 DE TABLE(NT):= IA
NT:= NT+1
E8
RES(IR):= MΦD(IA)
LV DE RES(IR) = 7
LH DE RES(IR) = 0
IR:= IR+7
oui LV DE MΦD(IA) = 0
IA:= LV DE MΦD(IA)
E9
NON TYPE DE MΦD(IA) ≠ 3
E10 oui
NON LH DE MΦD(IA) ≠ 0
E10A oui
C1 DE TABLE(NT):= LH DE MΦD(IA)
C2 DE TABLE(NT):= IA
NT:= NT+1
E10B
RES(IR):= MΦD(IA)
LH DE RES(IR):= 0
E11


L MΦD(IA) element indexe d'index id
ID:= REPRESENTANT DE MΦD(IA)
NON ID ≤ REP(I)
oui
NON REP(ID) = 0
oui
RES(IR):= MΦD(IA)
IR:= IR+7
E14
CI:= NSD:= NSF:= 0
E14A
CI:= CI+7
NON CI = ID
oui
NSF:= NSD+NSF
NSF:= NSF+REP(CI)
L:= REP(CI)
E14B
NSF:= NSF+REP(CI)
E15
IS:= NSD IS2:= NSF
E15A
RES(IR):= REP(IS)
oui LH DE RES(IR) = 0
NON
LH DE RES(IR):= LH DE RES(IR)+L
oui IS > IS2
E15A
NON
IR:= IR+7
IS:= IS+7
E20
oui LH DE MΦD(IA) = 0
NON
C1 DE TABLE(NT):= LH DE MΦD(IA)
C2 DE TABLE(NT):= IA
NT:= NT+7
E22
NT:= NT-7
IT:= NT
oui NT = 0
E26
BOL:= FAUX
E25
BOL:= VRAI
IA:= C1 DE TABLE(NT)
LH DE RES(C2 DE TABLE(NT)):= IR+7
FIN

5°) Procédure

TYPE 2

STATUT	NATURE	NOM	LONGUEUR	PARTITION NOM	PARTITION DANS	INDEX NOM	PLACE
DON	FILE	MOD		ELEMENT	CATAL	IA	
DON	FILE	REP		ELEMENT	CATAL	IP,IS,IS2	
RES	FILE	RES		ELEMENT	CATAL	IR	
RES	BOOL	BOL					
AUX	FILE	TABLE		DEUX C	CATAL	IT	
AUX	NAT	NT	4				
AUX	NAT	NT2	4				
AUX	NAT	ELR	4				
AUX	NAT	ELR2	4				
AUX	NAT	NT					
AUX	NAT	ELA	4				
AUX	NAT	ELR	4				
AUX	NAT	CI	4				
AUX	NAT	NSD	4				
AUX	NAT	NSF	4				
AUX	NAT	ID	4				
AUX	NAT	L	4				
AUX	INDEX	IT					
AUX	INDEX	IP					
AUX	INDEX	IS					
AUX	INDEX	IR					
AUX	INDEX	IA					
AUX	INDEX	IS2					

TYPE TRAITEMENT

```
E2 :    NT :=0 ; ELR := 0 ; ELA :=0 ; NT2:=0 ;
E4 :    IR := ELR APRES PREMIER ;
        IT := NT APRES PREMIER ;
        IA := ELA APRES PREMIER ;
E5 :    SI TYPE DE MOD (IA) = 1/2  VERS E6 ; VERS E9 ;
E6 :    SI LH DE MOD (IA) ≠ 0  VERS E7 ;  VERS E8 ;
E7 :    C1 DE TABLE (IT) := LH DE MOD (IA) ;
        C2 DE TABLE (IT) := ELR ; NT := NT+1 ;
E8 :    RES (IR) := MOD (IA) ;
        LV DE RES (IR) :=1 ; LH DE RES (IR) :=0 ;
        ELR := ELR+1 ;
        SI LV DE MOD (IA) = 0  VERS E22 ;
```



```
        ELA := LV DE MOD (IA) ;
        VERS E4 ;
E9 :    SI TYPE DE MOD (IA) ≠ 3 VERS E10 ;  VERS E11 ;
E10:    SI LH DE MOD (IA) ≠ 0  VERS E10A ;  VERS E10B ;
E10A:   C1 DE TABLE (IT):= LH DE MOD (IA) ;
        C2 DE TABLE (IT):= ELR ; NT:=NT+1 ;
E10B:   RESULTAT (IR):= MOD (IA) ; LH DE RES (IR) = 0 ;
        VERS E20 ;
E11:    ID := REPRESENTANT DE MOD (IA)
        SI ID < REPERE (PREMIER)  VERS E12 ;  VERS E26 ;
E12:    IP := ID APRES PREMIER ;
        SI REPERE (IP) = 0  VERS E13 ; ELR2 := ELR ; VERS E14 ;
E13:    RES (IR) := MOD (IA) ; ELR := ELR+1 ;
        VERS E20 ;
E14:    CI:=0 ; NSD :=0 ; NSF :=0 ;
E14A:   CI:=CI+1 ; IP:=CI APRES PREMIER ;
        SI CI = ID  VERS E14B ;  VERS E14C ;
E14B:   NSD := NSD+NSF ;
        NSF := NSF+REP (IP) ; L := REP (IP) ;  VERS E15 ;
E14C:   NSF:=NSF+REP (IP) ;  VERS E14A ;
E15 :   IS := NSD APRES PREMIER ; IS2 := NSF APRES PREMIER ;
E15A:   RES (IR) := REP (IS) ;
        SI LH DE RES (IR) = 0  VERS E15B ;
        LH DE RES (IR) := LH DE RES (IR)+L ; ELR := ELR+1 ;
        SI IS < IS2  VERS E20 ;
E15B:   AV IR DE 1 ; AV IS DE 1 ;  VERS E15A ;
E20 :   SI LH DE MOD (IA) = 0  VERS E22 ;
        C1 DE TABLE (IT) := LH DE MOD (IA)
        C2 DE TABLE (IT) := ELR ; NT:=NT+1 ;
E22 :   NT := NT-1 ; IT:= NT APRES PREMIER ;
        SI NT=0  VERS E25 ;
        ELA :=C1 DE TABLE (IT) ; IR := C2 DE TABLE (IT) APRES PREMIER ;
        LH DE RES (IR) := ELR+1 ;
        VERS E4 ;
E25 :   BOL := VRAI ;  VERS E27 ;
E26 :   BOL := FAUX
E27 :   FIN
```

3.3 TRAITEMENT IMMEDIAT ASSOCIE A UN PROGRAMME COGENT

3.3.1 Programme A. T. F.

On passe d'un programme COGENT à un programme A. T. F. en utilisant les procédures précédentes supposées placées dans le catalogue, on exécute pour cela un TRAITEMENT IMMEDIAT aux caractéristiques suivantes : tous les identificateurs des feuilles de type spécification sont des auxiliaires, les feuilles de type traitement comportent exclusivement des instructions d'affectation et des appels de procédure.

Un programme COGENT est défini par un traitement immédiat dont les arguments donnés sont : les grammaires primaire et secondaire, la chaîne lue, la liste des procédures et la suite des axiomes, les feuilles de type traitement représentent la section des procédures, dont chaque procédure est définie par une procédure A. T. F.

3.3.2 Description du traitement immédiat associé à un programme COGENT

a) Informations nécessaires au traitement immédiat

La grammaire primaire (GPE), éventuellement la grammaire secondaire (GSE) sous leur représentation externe structurée par la partition REGLEX, la liste des procédures (LISTEP) associées aux règles, la liste des identificateurs de procédures (PX1), la suite des axiomes utilisés pour l'analyse de la chaîne lue (LIAX), la plupart de ces informations sont stockées sur un ruban. On admet que les procédures associées aux règles ont été précédemment rangées dans le catalogue.

b) Exécution du traitement immédiat

On appelle pour cela 4 procédures :

TRANSG donne de la grammaire primaire et éventuellement de la grammaire secondaire une représentation interne. (article structure par REGLE) ;

ANALYSE éxécute l'analyse de la chaîne lue, son résultat est la pseudo-arborescence PAR ;

EXECUTION éxécute les opérations sur la pseudo-arborescence résultat de ANALYSE ;

SORTIE donne le résultat final du traitement immédiat.

TRAITEMENT IMMEDIAT SYST ANAL

TYPE REFERENCES

REFERENCES PROCEDURES EMPLOYEES
tous les identificateurs de procédures associés aux règles

TYPE SPECIFICATIONS

STATUT	NATURE	NOM	LONGUEUR	PARTITION NOM	PARTITION DANS	INDEX NOM	PLACE
AUX	FILE	GPE	CONSTANTE : NBRE DE REGLES DE LA GRAMMAIRE PRIMAIRE	REGLEX	CATAL	DE	ERUBAN
AUX	FILE	GSE	CONSTANTE: NBRE DE REGLES DE LA GRAMMAIRE SECONDAIRE	REGLEX	CATAL	SE	ERUBAN
AUX	ARTICLE	PX1		AUX1F	CATAL		
AUX	FILE	GP	LONG GPE	REGLE	CATAL	P	
AUX	FILE	GS	LONG GSE	REGLE	CATAL	S	
AUX	MOT	M	200				
AUX	ARTICLE	LIAX		AXTER	CATAL		
AUX	FILE	PA		ELEMENT	CATAL	IP	
AUX	BOOL	B					
AUX	FILE	LISTEP	NBRE DE PROCEDURES ASSOCIEES AUX REGLES	LIDPROC	CATAL	LI	
AUX	FILE	RES		ELEMENT	CATAL	IR	
AUX	FILE	CHLU		CHAINE			ERUBAN
AUX	NAT	RA	2				
AUX	INDEX	PE					
AUX	INDEX	SE					
AUX	INDEX	P					
AUX	INDEX	S					
AUX	INDEX	IP					
AUX	INDEX	LI					
AUX	INDEX	IR					

TYPE PARTITION

PARTITION	CONDITION	JUSQUA	NBFOIS	NATURE	LONGUEUR	NOM
AXTER				MOT	10	AX
				NAT	2	NBTER
			NBTER	MOT	3	TER
CHAINE		DRAP		MOT	3	

TYPE DEFINITIONS

NOM	VALEUR
LISTEP	tous les noms des procédures associées aux règles
PX1	tous les identificateurs de procédure associés aux règles avec le même ordre de rangement que dans LISTEP.
LIAX	tous les axiomes avec les terminaisons qui leurs sont associées.
DRAP	Valeur de DRAP.

TYPE TRAITEMENT

```
M := CHLU ;  RA := 1 ;
TRANSG (GPE, PX1 ; ; GP) ;
TRANSG (GSE, PX1 ; ; GS) ;
AXIO := MOT AXTER DE LIAX (RA) ;
ANALYSE (GP, M, AXIO ; ; PA, B) ;
EXECUTION (PA, LISTEP ; ; RES) ;
SORTIE (RES, G ; ;) ;
```

TRANGR (GE, PX1 ; ; G)

1°) Introduction

Etant donné la représentation externe d'une grammaire G TRANGR (G) a pour résultat la représentation interne de la grammaire G.

2°) Définition

TRANGR admet 2 paramètres donnée :

GE file représentation externe d'une grammaire

PX1 file représentant les identificateirs de procédures associés aux règles.

1 paramètre résultat

G file représentation interne de la grammaire définie par GE

Cette procédure permet le passage de la représentation externe à la représentation interne d'une grammaire.

TYPE REFERENCE

TRANSG

TYPE SPECIFICATIONS

STATUT	NATURE	NOM	LONGUEUR	PARTITION NOM	PARTITION DANS	INDEX NOM	PLACE
DON	FILE	GE		REGLEX	CATAL	PE	ERUBAN
DON	ARTICLE	PX1		AUX1F	CATAL		
RES	FILE	G		REGLE	CATAL	P	
AUX	ARTICLE	PCAR		CARF	CATAL		
AUX	ARTICLE	REG		REGLE	CATAL		
AUX	NAT	I	4				
AUX	INDEX	PE					
AUX	INDEX	P					

TYPE TRAITEMENT

```
        I:=0 ;
E1 :    PE:=I APRES PREMIER ; P:=I APRES PREMIER ;
        PCAR := GE (PE) ; TRANSG ( PCAR, PX1; B ; REG)
        SI NON B VERS E3 ;  G (P) := REG ;
        SI PE = DERNIER VERS FIN ;  I:=I+1 ;  VERS E1 ;
E3 :    ERREUR ;
        FIN
```

EXECUTION (PAR, G, LISTE P ; ; RES)

1°) Introduction

Etant donné une pseudo-arborescence r, résultat de l'analyse de la chaîne lue EXECUTION effectue la transformation ϕ définie par les procédures associées aux règles. (voir définition de ϕ § 3.4.4.3/e)

2°) Définition

EXECUTION a 3 paramètres donnée :

PAR file représentant la pseudo-arborescence à transformer,

G file représentant la grammaire sur laquelle est définie la pseudo-arborescence dans PAR

LISTEP file représentant les procédures associées aux règles de G.

1 paramètre résultat :

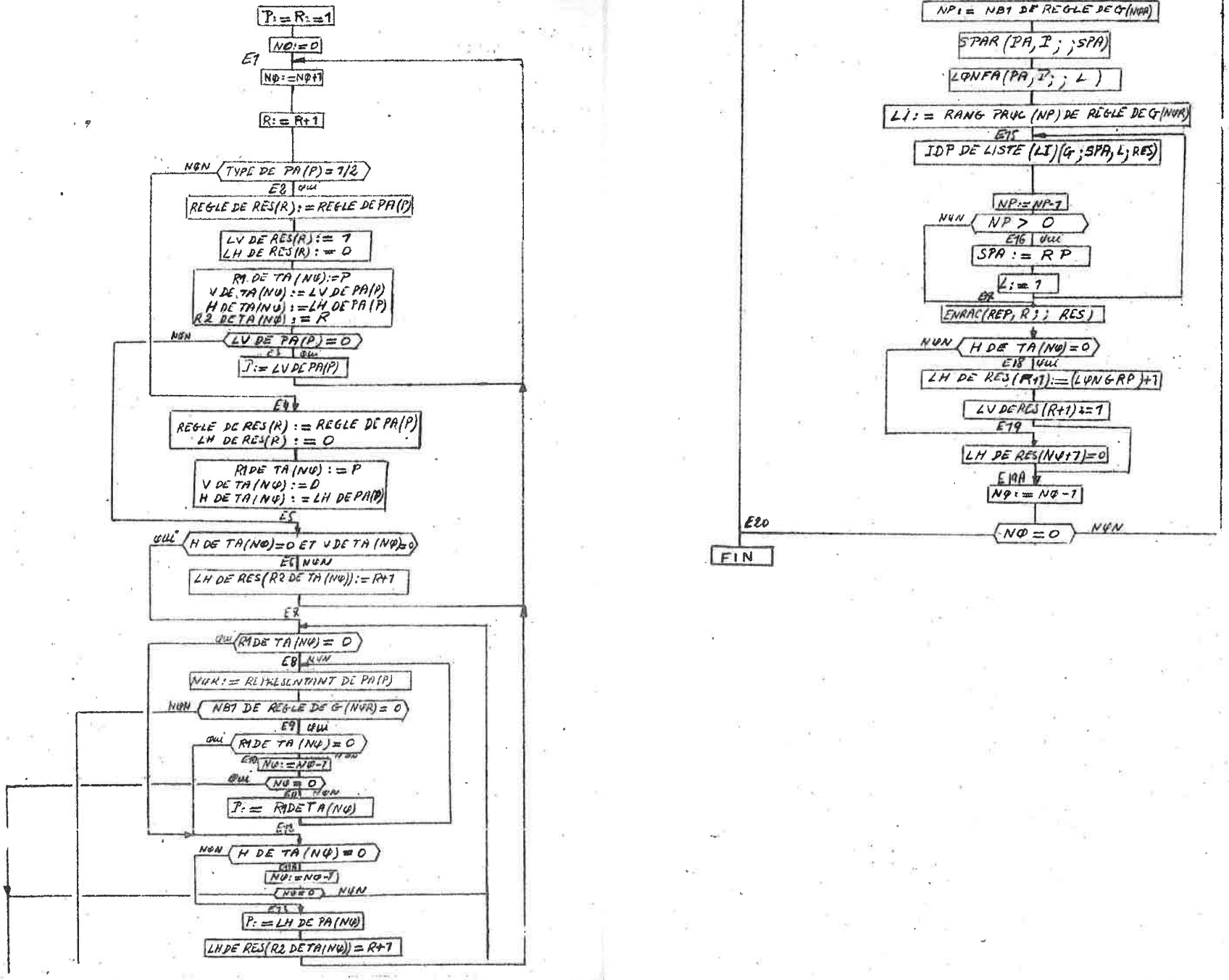
RES file représentant une pseudo-arborescence.

Cette procédure applique à la pseudo-arborescence représentée dans PAR la transformation définie par les procédures associées aux règles de la grammaire représentée par GP.
Pour l'ordre d'appel des procédures on se reportera à la définition de ϕ paragraphe 2.4.4.3.e et à l'exemple de ce paragraphe.

Les procédures associées à une règle ont toujours un seul argument donnée, deux arguments mixtes et un argument résultat. L'argument donnée est une file représentant une grammaire G, elle sera paramètre de la procédure ANALYSE appelée avant toute exécution des procédures OPANALYSE et OPSYNTHESE qui définissent respectivement les instructions d'analyse et de synthèse.

Le premier argument mixte est une file représentant la pseudo-arborescence sur laquelle portera l'opération, le second est un entier qui est toujours égal à un pour toutes les procédures sauf la première. La première procédure appelée a pour argument donnée une pseudo-arborescence $n \uparrow r$ et opère sur les pseudo-arborescences composant r, dans ce cas le second argument mixte a pour valeur le nombre de pseudo-arborescences, composant r. (on rappelle que seule la première procédure appelée peut avoir plusieurs arguments).

L'argument donnée est une file représentant le résultat unique de la procédure.


P:=R:=1
NΦ:=0
E1
NΦ:=NΦ+1
R:=R+1
NON
TYPE DE PA(P)=1/2
E2
OUI
REGLE DE RES(R):=REGLE DE PA(P)
LV DE RES(R):=1
LH DE RES(R):=0
R1 DE TA(NΦ):=P
V DE TA(NΦ):=LV DE PA(P)
H DE TA(NΦ):=LH DE PA(P)
R2 DE TA(NΦ):=R
NON
LV DE PA(P)=0
E3
OUI
P:=LV DE PA(P)
E4
REGLE DE RES(R):=REGLE DE PA(P)
LH DE RES(R):=0
R1 DE TA(NΦ):=P
V DE TA(NΦ):=0
H DE TA(NΦ):=LH DE PA(P)
E5
OUI
H DE TA(NΦ)=0 ET V DE TA(NΦ)≠0
E6
NON
LH DE RES(R2 DE TA(NΦ)):=R+1
E7
OUI
R1 DE TA(NΦ)=0
E8
NON
NUMR:=RETOURSCONTINT DE PA(P)
NON
NBT DE REGLE DE G(NUMR)=0
E9
OUI
R1 DE TA(NΦ)=0
E10
NΦ:=NΦ-1
NON
OUI
NΦ=0
E11
NON
P:=R1 DE TA(NΦ)
E12
NON
H DE TA(NΦ)=0
E13
NΦ:=NΦ-1
NΦ=0
NON
E14
P:=LH DE PA(NΦ)
LH DE RES(R2 DE TA(NΦ))=R+1
E14
NP:=NBT DE REGLE DE G(NUMR)
SPAR(PA,I;;SPA)
LONFA(PA,I;;L)
LI:=RANG PROC(NP) DE REGLE DE G(NUMR)
E15
IDP DE LISTE(LI)(G;SPA,L;RES)
NP:=NP-1
NON
NP>0
E16
OUI
SPA:=RP
L:=1
E17
ENRAC(REP,R;;RES)
NON
H DE TA(NΦ)=0
E18
OUI
LH DE RES(R+1):=(LONGRP)+1
LV DE RES(R+1):=1
E19
LH DE RES(NΦ+1)=0
E19A
NΦ:=NΦ-1
E20
NΦ=0
NON
FIN

TYPE REFERENCE

SPAR, LONFA, ENRAC

TYPE SPECIFICATION

STATUT	NATURE	NOM	LONGUEUR	PARTITION NOM	PARTITION DANS	INDEX NOM	PLACE
DON	FILE	PA		ELEMENT	CATAL	IP	
DON	FILE	G		REGLE	CATAL	IG	
DON	FILE	LISTEP		LIDPROC	CATAL	IL	
RES	FILE	RES		ELEMENT	CATAL	IR	
AUX	NAT	P	4				
AUX	NAT	R	4				
AUX	NAT	R2	4				
AUX	NAT	NO	3				
AUX	NAT	NP	2				
AUX	NAT	L	2				
AUX	NAT	LI	2				
AUX	FILE	SPA		ELEMENT	CATAL	IS	
AUX	FILE	RP		ELEMENT	CATAL	IRP	
AUX	ARTICLE	TA	200	QUATRE			
AUX	NAT	NOR					
AUX	NAT	RX	4				
AUX	NAT	NP	4				
AUX	NAT	GX	4				
AUX	INDEX	IL					
AUX	INDEX	IP					
AUX	INDEX	IG					
AUX	INDEX	IS					
AUX	INDEX	IR					
AUX	INDEX	IRP					

TYPE PARTITION

PARTITION	CONDITION	JUSQUA	NBFOIS	NATURE	LONGUEUR	NOM
QUATRE				NAT	4	R1
				NAT	4	H
				NAT	4	V
				NAT	4	R2

TYPE TRAITEMENT

```
        P:=1 ; R:=1 ; NO:=0 ; IP:= PREMIER ;
E1 :    NO:=NO+1 ; R:=R+1 ; IR:= R APRES PREMIER ; IR:= R APRES PREMIER ;
        SI TYPE DE PA (IP) = 1 OU TYPE DE PA (IP) = 2 VERS E2 ; VERS E4 ;
E2 :    REGLE DE RES (IR) := REGLE DE PA (IP) ;
        LV DE RES (IR) :=1 ; LH DE RES (IR) :=0 ;
        R1 DE TA (NO) :=P ; V DE TA (NO) := LV DE PA (IP) ;
        H DE TA (NO) := LH DE PA (IP) ; R2 DE TA (NO) := R ;
        SI LV DE PA (IP) = 0 VERS E3 ; VERS E4 ;
E3 :     P:= LV DE PA (IP) ; VERS E1 ;
E4 :    REGLE DE RES (IR) := REGLE DE PA (IP) ; LH DE RES (IR) :=0 ;
        R1 DE TA (NO) :=P ; V DE TA (NO) :=0 ; H DE TA (NO) := LH DE PA (IP) ;
E5 :    SI H DE TA (NO) = 0 ET V DE TA (NO) = 0 VERS E7 ;
        RX := R2 DE TA (NO) +1 ; IR:= RX APRES PREMIER ;
        LH DE RES (IR) := R+1 ;
E7 :    SI R1 DE TA (NO) = 0 VERS E12 ;
E8 :    NOR := REPRESENTANT DE PA (IP) ;
        SI NBT DE REGLE DE G (NOR) =0 VERS E9 ; VERS E14 ;
E9 :    SI R1 DE TA (NO) = 0 VERS E12 ;
        NO := NO-1 ; SI NO=0 VERS E20 ; IP:=R1 DE TA (NO) APRES PREMIER ;
        VERS E8 ;
E12 :   SI H DE TA (NO) =0 VERS E12A ; VERS E13 ;
E12A:   NO:=NO-1 ; SI NO:=0 VERS E20 ; VERS E7 ;
E13 :   IP:=LH DE PA (NO) APRES PREMIER ;
        RX:= R2 DE TA (NO) +1 ; IR:= RX APRES PREMIER ; LH DE RES (IR):=R+1 ;
        VERS E1 ;
E14 :   NP:= NB1 DE REGLE DE G (NOR) ;
        SPAR (PA, P ; ; SPA) ;
        LONFA (PA, P ; ; L) ;
        LI:= RANG PROC (NP) DE REGLE DE G (NOR) ;
E15 :   IDP DE LISTE (LI) (G; SPA, L ; RP)
        NP:=NP-1 ;
        SI NP > 0 VERS E16 ; VERS E17 ;
E16 :   SPA (PREMIER JUSQUA DERNIER):= RP (PREMIER JUSQUA DERNIER) ;
        L:=1 ; VERS E15 ;
E17 :   ENRAC (RP ; R; RES) ;
        SI H DE TA(NO) =0 VERS E18 ; VERS E19 ;
E18 :   RX:=R+1 ; IR:=RX APRES PREMIER ; LH DE RES(IR) := LONG RP+1 ;
        LV DE RES(IR) :=1 ; VERS E19A ;
E19 :   RX:=NO+1 ; IR:= RX APRES PREMIER ; LH DE RES (RX) :=0 ;
        NO:=NO-1 ; SI NO=0 VERS E20 ; VERS E7 ;
E20 :   FIN ;
```

ENRAC (RP ; R ; RES)

1°) Introduction

Soit une pseudo-arborescence r et x un élément de r, et une pseudo-arborescence s ENRAC (s, x ; ; r) remplace la sous pseudo-arborescence enracinée en x, par s.

2° Définitions

ENRAC admet 1 paramètre donnée :

RP file représentant une pseudo-arborescence

1 paramètre mixte :

R entier repérant un élément de la pseudo-arborescence RES

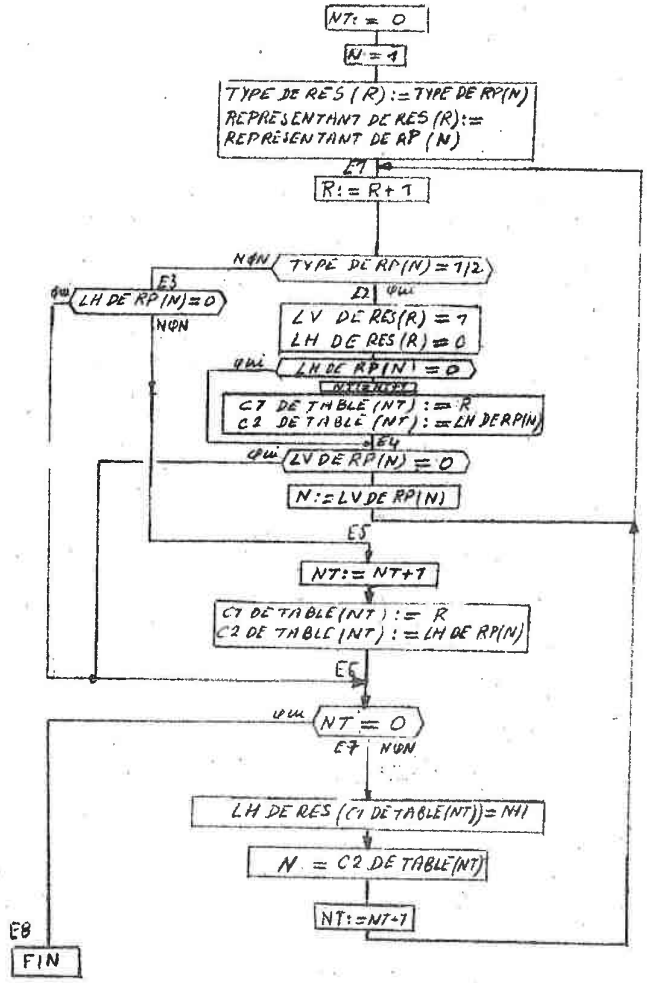
1 paramètre résultat

RES file représentant une pseudo-arborescence

ENRAC crée la pseudo-arborescence représentée par RES dont RP est une sous-pseudo-arborescence enracinée en un élément repréré par R.

TYPE SPECIFICATION

STATUT	NATURE	NOM	LONGUEUR	PARTITION NOM	PARTITION DANS	INDEX NOM	PLACE
DON	FILE	RP		ELEMENT	CATAL	IN	
MIX	NAT	R	4				
RES	FILE	RES		ELEMENT	CATAL	IR	
AUX	FILE	TABLE		DEUX C	CATAL		
AUX	NAT	N	4				
AUX	NAT	NT	4				
AUX	INDEX	IN					
AUX	INDEX	IR					



TYPE TRAITEMENT

```
        NT:=0 ; IN:=PREMIER ; IR :=R APRES PREMIER ;
        TYPE DE RES (IR) := TYPE DE RP (IN) ;
        REPRESENTANT DE RES (IR) := REPRESENTANT DE RP (IN) ; N:=1 ;
E1 :    R:=R+1 ; IN:=N APRES PREMIER ; IR:= R APRES PREMIER ;
        SI TYPE DE RP (IN) =1 OU TYPE DE RP (IN)=2  VERS E2 ;
        SI LH DE RP (IN) = 0  VERS E6 ;  VERS E5 ;
E2 :    LV DE RES (IR) := 1 ; LH DE RES (IR) :=0 ;
        SI LH DE RP (IN) =0  VERS E4 ; NT:= NT+1 ;
        C1 DE TABLE (NT) := R ; C2 DE TABLE (NT) := LH DE RP (IN) ;
E4 :    SI LV DE RP (N) = 0 VERS E6 ; N:=LV DE RP (IN) ; VERS E1 ;
E5 :    NT:=NT+1 ; C1 DE TABLE (NT):=R ; C2 DE TABLE (NT):= LH DE RP (IN) ;
        SI NT=0  VERS E8 ;
        R:= C1 DE TABLE (NT)+1 ; IR:=R.APRES PREMIER ;
        LH DE RES (IR) := R+1 ; N:= C2 DE TABLE (NT) ; VERS E1 ;
E8 :    FIN ;
```

Exemple de passage d'une procédure COGENT à une procédure A.T.F.

3°) Procédure COGENT

```
$GENERATOR PROC NEG ((OPE, X)
   +1 IF  X = / (EXP /(TERME)), X.
      X/ = (TERME/(() (EXP) ())), X.
   1/ X/ = (EXP/-(TERME), X.
      $ RETURN (X).
```

2°) Procédure A.T.F.

```
PROC NEG (G; LID,  L ; RES)
```

TYPE REFERENCE

```
SPAR, LONFA, ANALYSE, OPANALYSE, OPSYNTHESE, RAMI ;
```


TYPE SPECIFICATION

STATUT	NATURE	NOM	LONGUEUR	PARTITION NOM	PARTITION DANS	INDEX NOM	PLACE
DON	FILE	G		REGLE	CATAL		
MIXTE	FILE	LID		ELEMENT	CATAL	IL	
MIXTE	NAT	L	2				
RES	FILE	RES		ELEMENT	CATAL	IR	
AUX	FILE	X		ELEMENT	CATAL	IX	
AUX	FILE	VAR		LIVAR	CATAL	IV	
AUX	FILE	X2		ELEMENT	CATAL	IX2	
AUX	FILE	X3		ELEMENT	CATAL	IX3	
AUX	BOOL	REP	1				
AUX	NAT	V	2				
AUX	NAT	N	2				
AUX	MOT	M	10				
AUX	MOT	AX	10				
AUX	INDEX	IL					
AUX	INDEX	IR					
AUX	INDEX	IX					
AUX	INDEX	IV					
AUX	INDEX	IX2					
AUX	INDEX	IX3					

TYPE TRAITEMENT

```
        IL:= PREMIER ;
        V:= LV DE LID (IL) ; IL:= V APRES PREMIER ;
        V:= LH DE LID (IL) ; SPAR (LID, V ; ; X2) ;
        M:= "TERME" ; AX:= "EXP" ; ANALYSE (GP, M, AX ; ; X, REP) ;
        IV:= PREMIER ; VAR (IV) :=0 ; N:=1 ;
        OP ANALYSE (X2, X,N ; VAR ; REP) ; N:= VAR (PREMIER) ;
        SI REP VERS E1 ; VERS E11 ;
E1 :    SPAR (LID, V ; ; X) ;  VERS E2 ;
E11 :   SPAR (X2, N ; ;X) ;
        AX:= "TERME" ; M:= "((} (EXP) ())" ; ANALYSE (GP, M, AX ; ; X3 REP) ;
        V := 2 ; LV DE X2 (PREMIER):=1 ; IV:=2 APRES PREMIER ;
        LV DE X2 (IV) := LONG X2 ; RAMI (X ; V ; X2) ;
        OP SYNTHESE (X3, X2 ; ; X, REP) ;
E2 :    AX:= "EXP" ; M:= "-TERME" ; ANALYSE (GP,M,AX ; ; X3, REP) ;
        V:=2 ; LV DE X2 (PREMIER) :=1 ; IV:=2 APRES PREMIER ;
        LV DE X2 (IV) := LONG X2 ; RAMI (X ; V, X2 ; ) ;
        OP SYNTHESE (X3, X2 ; ; X, REP) ;
        RES (PREMIER JUSQUA DERNIER) := X (PREMIER JUSQUA DERNIER) ;
        FIN ;
```

BIBLIOGRAPHIE

L. BOLLIET - N. GASTINEL - P. J. LAURENT
ALGOL un nouveau langage scientifique - Harman - Paris (1964)

J. W. BACKUS — Report on the algorithmic language Algol 60 Numerische Mathematik 2 (1960) p. 106-136

C. BERGE — Théorie des graphes et ses applications Dunod - Paris (1962)

N. CHOMSKY — Formal properties of grammars - Handbook of mathematical psychology - (Luce, Bresh, Galauder editors), Wiley and Sons, New York (1963)

K. E. IVERSON — A programming language Wiley and sons New York (1962)

L. NOLIN — Formalisation des notions de machine et de programme - Thèse Institut Blaise Pascal Paris (1968)

L. NOLIN - M. GAMBACH - CH. PICARD - M. RITOUT - P. RIVIERE Y. ROMANO - P. VERROEST - Un langage de programmation pour les besoins de la gestion - l'A. T. F. GESTION Institut Blaise Pascal (Avril 1967)

C. PAIR — Etude de la notion de pile - application à l'analyse syntaxique Thèse publication Faculté des Sciences de l'Université de Nancy - (1966)

C. PAIR - A. QUERE — Définition et étude des bilangages réguliers - Publication - Institut Universitaire de Calcul Automatique de Nancy (1968) (à paraître dans Information and control)

J. C. REYNOLDS — An introduction to the COGENT programming system Argonne National Laboratory

J. C. REYNOLDS — COGENT programming manuel - Argonne National Laboratory - ANL 7022 march 1965

J. C. REYNOLDS — COGENT 1.2 Operation Manual Technical report CS37 - Argonne National Laboratory - april 22, 1966

NOM DE L'ETUDIANT : SCHIVI Henri

Nature de la thèse : Doctorat de Spécialité en mathématiques appliquées

Vu, Approuvé

et Permis d'Imprimer

NANCY le 1 Novembre 1968

LE DOYEN

J. ALBERT