N° 398

2ex Dactyl

UNIVERSITE DE NANCY 1 U.E.R. DE SCIENCES MATHEMATIQUES

A

Sc.N.72/99

FACE

LANGAGE D'ECRITURE DE COMPILATEURS

DEFINITION ET IMPLEMENTATION

UNIVERSITE NANCY BIBLIOTHEQUE

THESE

pour l'obtention du

DOCTORAT DE SPECIALITE MATHEMATIQUES (3ème CYCLE)

soutenue le 26 octobre 1972

par

FRANCOISE BELLEGARDE

JURY : Mr LEGRAS J. Président
Mr DEPAIX M. Examinateur
Mr PAIR C. Examinateur

UNIVERSITE DE NANCY I U.E.R. DE SCIENCES MATHEMATIQUES

FACE

LANGAGE D'ECRITURE DE COMPILATEURS

DEFINITION ET IMPLEMENTATION

THESE

pour l'obtention du

DOCTORAT DE SPECIALITE MATHEMATIQUES (3ème CYCLE)

soutenue le 26 octobre 1972

par

FRANCOISE BELLEGARDE

JURY : Mr LEGRAS J. Président
Mr DEPAIX M. Examinateur
Mr PAIR C. Examinateur

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 a bien voulu me faire en présidant ce jury.

J'adresse toute ma reconnaissance à Monsieur le Professeur PAIR qui a inspiré ce travail et en a assuré la direction avec une attention et une bienveillance constante.

Je remercie Monsieur le Professeur DEPAIX, Président de l'Université d'enseignement et de recherche des Sciences Mathématiques à l'Université de Nancy I, pour l'honneur qu'il me fait en participant au jury.

Je tiens également à remercier J. MAROLDT, H. PISTRE et A.M. BOUCHET pour leur sympathique collaboration, en particulier pour l'implémentation du Langage Face.

Enfin, il me faut remercier le secrétariat qui a assuré la réalisation matérielle de cette thèse.

SOMMAIRE

Introduction.

CHAPITRE 1 . FACE LANGAGE D'ECRITURE DE COMPILATEURS.

1.1. Examen du processus de compilation.

1.2. Le fonctionnement du compilateur écrit en Face.

1.3. Les types d'information traités par Face.

1.4. Description du langage Face.

1.4.1. Eléments de base.
1.4.2. Programme
1.4.3. Définition des grammaires
1.4.4. Liste de déclarations
1.4.5. Déclaration de procédure
1.4.6. Déclaration de rangée de procédures
1.4.7. Expression fermée.

1.5. Exemple de programme Face.

CHAPITRE 2. GENERALITES SUR LE COMPILATEUR FACE.

2.1. Organisation générale du compilateur.

2.1.1. Les passages du compilateur.
2.1.2. Définition du compilateur

2.2. Représentation des informations.

2.2.1. Représentation des grammaires
2.2.2. Représentation des éléments, des repères, des rangées d'éléments ou de repères.
2.2.3. Représentation des rangées de procédures.

2.3. Organisation de la mémoire à l'éxécution.

2.3.1. La zone des grammaires
2.3.2. La zone globale
2.3.3. La zone du programme objet
2.3.4. La zone des rangées de procédures

2.3.5. La zone locale.

2.4. Traduction d'un programme Face

2.4.1. Traduction des grammaires
2.4.2. Traitement des identificateurs
2.4.3. Traitement des informations associées aux appels de procédures.
2.4.4. Traitement des informations associées aux procédures d'une rangée.
2.4.5. Traitement des expressions
2.4.6. Traitement des erreurs.

2.5. Les déclarations globales du compilateur

CHAPITRE 3. L'ETUDE LEXICOGRAPHIQUE : LA RANGEE DE PROCEDURE LEX.

3.1. Eléments de bases.
3.2. Définition des grammaires
3.3. Déclarations globales
3.4. Début d'un bloc
3.5. Programme.

CHAPITRE 4. LA GENERATION : LA RANGEE DE PROCEDURES GEN.

4.1. Eléments de base.
4.2. Définition des grammaires
4.3. Déclarations globales
4.4. Déclarations locales
4.5. Expressions
4.6. Programme.

CHAPITRE 5. PROCEDURES.

Celles ci sont rangées par ordre alphabétique des noms de procédures soit dans cet ordre :

B
B1, B2
COMPAR
CREETI
DEPOINT
DEREP
FINDOP
FILTRE
FILTRE1
FILTRE2
GRG
MRAPPL
OPERER
RECHNIL
REPOUSS
TLACER
TOINTER

Conclusion.

ANNEXE 1

1.1. Restrictions apportées à la définition de Face pour définir le langage Face 0.
1.2. Les procédures standard Face 0

ANNEXE 2.

Grammaire Face.

INTRODUCTION

Face est un langage pour l'écriture de compilateur et ce travail présente les principales idées qui sont à la base de ce langage puis sa définition et son implémentation actuelle.

Comme la plupart des autres langages permettant l'écriture de de compilateurs (voir par exemple [2], [5], [7], [10], [15], [16], [17],) FACE s'applique à des langages dont la syntaxe est définie par une grammaire à contexte libre [1]. Un programme écrit en FACE[1] contient la description d'une telle grammaire G et il peut décrire, grâce à un certain nombre de procédures, une traduction du langage engendré par G ; sa donnée est donc une chaîne de caractères (texte source) et son résultat peut être une traduction de cette chaîne (texte objet) ou un certain nombre de méssages d'erreur si la chaîne donnée n'appartient pas au langage engendré par G ou si elle ne satisfait pas à certaines conditions supplémentaires.

La compilation comporte l'analyse syntaxique du texte source (avec éventuellement messages d'erreur du premier type) et la génération du texte objet et des messages d'erreur du second type ; la génération demande certaines opérations de service, comme la construction de tables d'identificateurs (étude lexicographique). L'analyse syntaxique et la génération peuvent ou non être séparées dans le temps ; de même, la génération peut être effectuée en un ou plusieurs passages, un passage pouvant par exemple être réservé à l'étude lexicographique.

On peut donner à un système d'écriture de compilateurs les objectifs suivants :

a) Le programmeur doit définir son compilateur sans avoir à se préoccuper en aucune façon de l'analyse syntaxique. La première conséquence est que l'analyse doit être effectué automatiquement, par une procédure (l'analyseur) faisant partie du système, dont les données sont la grammaire considérée et la chaîne à analyser. Il en est ainsi pour tous les langages d'écriture de compilateurs, dont chacun est fondé sur un algorithme d'analyse. Mais il semble souhaitable d'aller plus loin : le programmeur ne doit pas être obligé de connaître la

(1) On peut penser que FACE signifie "fabrication automatique de compilateurs efficaces". Mais ce nom ne vient-il pas plutôt du fait que FACE utilise une pile ?

méthode d'analyse, qui pourrait même dépendre du langage étudié et être choisie automatiquement ; en tout cas, la méthode d'analyse doit être sans influence sur la description de la génération du texte objet. Par exemple, il arrive fréquement qu'une analyse syntaxique "ascendante" soit plus efficace, alors que la génération se conçoit plus commodément de manière "descendante". On choisira donc la sortie de l'analyseur la plus commode pour la génération, et cette sortie sera indépendante de l'algorithme d'analyse.

b) La génération doit être décrite aisément et clairement. Pour réalise cet objectif, le plus simple est d'associer des procédures de génératio aux diverses règles de la grammaire ; l'ordre dans lequel ces procédures sont appelées, qui dépend du résultat de l'analyse syntaxique, ne doit pas avoir à être spécifié par le programmeur : le système doit prendre en charge le résultat de l'analyse et les procédur pour commander la succession des appels ; il contient pour cela un programme de commande.

c) Le système doit fournir au programmeur des outils convenables, définis clairement et indépendamment de leur représentation en machin Les chaînes de caractères sont bien entendu nécessaires mais on doit disposer aussi de tables. Une table peut être cnnsidérée comme une fonction à un argument (l'"indicatif") dont la valeur est une"adresse", ou un tableau d'adresses. Pour préciser cette notion d'adresse, nous introduirons en FACE des repères, semblables aux "noms" de [13]. Ces repères permettent aussi le traitement de listes, lorsque c'est nécessaire [16].

d) Une grande souplesse doit être laissée à l'utilisateur. Par exemple, on restreint considérablement le domaine d'emploi en ne décrivant, comme certains systèmes, que les compilations en un seul passage. Au contraire il faut pouvoir décrire un ou plusieurs passages, séparés ou non de l'analyse syntaxique, sans refaire à chacun d'eux cette analyse ou au contraire en utilisant des grammaires différentes aux divers passages : par exemple, l'analyse lexicographique peut être guidée par une grammaire beaucoup plus pauvre que celle qui doit guider la génération.

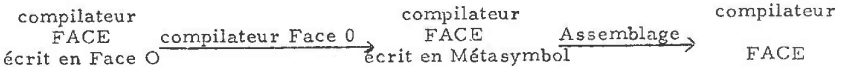
e) Enfin, le compilateur obtenu doit être efficace. Cette efficacité tient notamment à la manière dont est exploité le résultat de l'analyse par le programme de commande (cf. b) et donc à la forme que possède la sortie de l'analyseur. La difficulté est de concilier la souplesse, la clarté et la facilité d'écriture avec l'efficacité ; pour cela, il convient de privilégier les manières de travailler les plus fréquentes.

Après avoir examiné le processus de compilation en général, de manière à pouvoir préciser le rôle du programme de commande (1.1.), le chapître 1 donne la définition du langage FACE.

Le compilateur de Face en assembleur 10070 SYMBOL a été réalisé en utilisant une technique de bootstrapping . Il est écrit dans un langage noyau Face 0 qui comporte des restrictions par rapport au langage Face. Ces restrictions sont écrites en Annexe 1.

Le compilateur de Face 0 est, lui, écrit en Métassembleur 10070 (Métasymbol) [9] et son langage objet est Métasymbol.

On a donc le schéma suivant :



Le chapître 2 donne des généralités sur le compilateur Face. Dans les chapîtres 3 et 4, on trouve le programme Face 0 du compilateur Face sur les pages de gauche, assorti de commentaires sur les pages de droite.

Les compilateurs écrits en Face, comme le compilateur Face lui même sont "dirigés par la syntaxe" ([1], [7], [8], [16], [17], [18]). La grammaire de Face utilisée est donnée en Annexe 2 et les chapîtres 2,3,4 décrivent le compilateur en suivant le plan de la syntaxe donnée au chapître 1.

1.1. EXAMEN DU PROCESSUS DE COMPILATION.

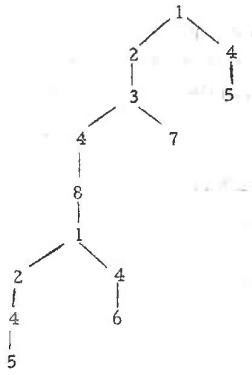
Le résultat de l'analyse syntaxique est une ramification à une racine. Cette notion est définie en [12] [13] ; bornons-nous à la suggérer en disant qu'il s'agit d'une arborescence, orientée de gauche à droite, dont chaque noeud est étiqueté (exemple 1) ; ici, on supposera que les étiquettes sont des numéros de règles de la grammaire pour laquelle est faite l'analyse.

Exemple 1.

Soit la grammaire G dont les règles (numérotées 1, 2 ...) s'écrivent :

E ::= E + T / 1 / T / 2
T ::= T x F / 3 / F / 4
F ::= a / 5 / b / 6 / c / 7 / (E) / 8

avec pour axiome E, elle engendre des expressions arithmétiques employant les opérateurs x et + et les variables a, b , c . Par exemple (a + b) x c + a appartient au langage engendré et la ramification engendrée est alors :



Une telle ramification r est composée d'une racine N numéro de règle et de p ramifications à une racine $r_1 \ldots r_p$

Un passage du compilateur qui utilise la ramification r résultat de l'analyse peut être décrit par une procédure g qui admet r en paramètre [14].

. La procédure g est récursive ; g(r) dépend de $g(r_1), \ldots, g(r_p)$.

Pour l'exemple ci-dessus :
- $g(r_1)$ admet un résultat x_1 : adresse et type du résultat intermédiaire (a + b) x c.
- $g(r_2)$ admet un résultat x_2 : adresse et type de la variable a.

Le calcul g(r) peut être exprimé par :

$$x_1 := g(r_1) ; x_2 := g(r_2) ; \varpi_1(x_1;x_2);$$

où ϖ_1 est une procédure associée à la règle numéro 1 utilisant en paramètre les résultats x_1, x_2 et fournissant le résultat de g(1). Ceci se généralise ; dans la plupart des cas on compile toutes les ramifications $r_1 \ldots r_p$ à une racine composant $r = N \times (r_1 + \ldots + r_p)$, avant d'utiliser leur résultat :

$$g(r) = (x_1 := g(r_1) ; \ldots ; x_p := g(r_p) ; \varpi_N(x_1 \ldots x_p))$$

Il n'en est pas ainsi cependant quand on doit effectuer des actions entre les compilations des ramifications composantes $r_1 \ldots r_p$. Prenons par exemple à la compilation d'une instruction conditionnelle engendrée par la règle :

C ::= si E alors I sinon S/1

Soient r_E, r_I, r_S les ramifications issues de E, I, S.

On a g(r) = (x := g(rE) ; e := h(x) ; g(rI) ; e' := h'(x) ; g(rS) ; h''(e'))

h est une procédure qui teste si E est de type booléen, génère un branchement conditionnel dont on retient l'étiquette e ;
h' génère un saut inconditionnel dont on retient l'étiquette e' et génère la définition de e ;
h'' génère la définition de e'.

On peut se ramener au cas précédent en décomposant la règle 1

C ::= C' sinon S	$g(r_B)$ = (x := $g(r_E)$; h(x))
C'::= B alors I	$g(r_c)$ = (e := $g(r_B)$; $g(r_I)$; h'(x))
B ::= si E	g(r) = (e' := $g(r_c)$; $g(r_S)$; h''(e'))

$g(r_I)$ et $g(r_S)$ sont sans résultat.

On montre [14] que ceci est général et qu'on peut se ramener au cas précédent.

. Pour réaliser le calcul de g(r) de façon pratique, il faut supprimer la récursivité de la procédure g :

Il faut sauvegarder x_1 avant de calculer $g(r_2)$, qui risque de le détruire ; de même, il faut sauvegarder $x_2 \ldots x_{p-1}$. Pour cela on utilise une pile : ϖ_N prendra ses p paramètres sur la pile et y replacera son résultat.

Le calcul de g est alors très facile si on représente la ramification r sous forme post fixée [14]. Bornons nous ici à un exemple, la ramification de l'exemple 1 est représenté sous cette forme par

5 4 2 6 4 1 8 4 7 3 2 5 4 1

et pour calculer g(r), on aura à exécuter successivement

$\varpi_5\ \varpi_4\ \varpi_2\ \varpi_6\ \varpi_4\ \varpi_1\ \varpi_8\ \varpi_4\ \varpi_7\ \varpi_3\ \varpi_2\ \varpi_5\ \varpi_4\ \varpi_1$.

(certaines procédures ϖ_i n'ont aucune action).

. Une difficulté subsiste : les paramètres et le résultat des procédures ϖ_k n'ont pas un format fixe:la compilation d'un opérande d'une expression arithmétique donne un résultat adresse type, celle d'une instruction est sans résultat ... On peut se ramener le plus souvent à un format unique en admettant que chaque ϖ_k peut avoir un nombre quelconque de résultats :

ϖ_k est une procédure prenant ses d_k arguments sur une pile et les y remplaçant par sk résultats. d_k et s_k sont des entiers liés à ϖ_k. En Face arguments et résultats seront tous du même type, éléments : un élément est représenté par le contenu d'un mot de mémoire. Il existe cependant des cas où on ne peut ainsi lier à ϖ_k un nombre fixe d'arguments

Considérons par exemple les déclarations de tableaux d'Algol 60 engendrés par les règles :

< déclaration de tableaux > ::= < type de tableau > < section de tableaux > /1
< déclaration de tableaux > < section de tableaux > /2
< section de tableaux > ::= < suite d'identificateurs > [< paires de bornes >]
< suite d'identificateurs > ::= < identificateur > /4
< suite d'identificateur >, < identificateur > /5

Il faut conserver tous les identificateurs de la suite d'identificateurs avant de connaître à la fois les paires de bornes et le type, renseignements communs à tous les identificateurs.

Or les résultats de la procédure g ayant pour paramètre les ramifications qui engendrent la suite d'identificateurs sont en nombre variable selon le nombre d'identificateurs de la suite :

Le nombre d'arguments des procédures ϖ_1 et ϖ_2 devrait donc varier. On considère donc un autre type d'argument formé d'une liste d'éléments, ou rangée d'éléments : le dernier argument d'une procédure ϖ_N pourra être une rangée d'éléments placée au sommet de la pile et appelée rangée en pile.

1.2. LE FONCTIONNEMENT D'UN COMPILATEUR ECRIT EN FACE.

L'analyseur de FACE est une procédure standard nommée analyse, qui a pour résultat la représentation postfixée d'une ramification, c'est à dire une liste, ou rangée, d'entiers. Les fonctions ϖ_N introduites ci-dessus dépendent de l'entier N : elles forment une rangée de procédures. Il existe une procédure standard nommée appl, qui applique les procédures d'une rangée, dans l'ordre spécifié par une autre rangée formée d'entiers qui sont les indices successifs des procédures, les arguments étant pris et les résultats placés sur une même pile.

Les procédures standard analyse et appl peuvent être combinées en une autre procédure standard anapplyse, qui effectue appl au fur et à mesure de la construction du résultat d'analyse, tout au moins lorsque l'analyse est déterministe.

Il suffit donc à l'auteur du compilateur d'écrire la rangée de procédures ϖ_N associées aux diverses règles ; il n'a pas besoin à ce moment d'avoir présente à l'esprit la manière dont est exécutée appl. Les

procédures analyse, appl, anapplyse peuvent être appelées plusieurs fois dans un même programme pour réaliser plusieurs passages du compilateur. L'écriture est donc très modulaire.

1.3. LES TYPES D'INFORMATION TRAITES PAR FACE.

Le premier type d'information est formé par les entiers ou éléments. Extérieurement, un entier peut être représenté dans diverses bases de numération, par exemple 10, 16, 8, 2, et aussi par une chaîne de caractères : pour une implémentation déterminée, on précisera la correspondance entre chaînes de caractères et entiers, ainsi que la valeur absolue maximum des entiers. Si besoin est, l'entier 0 servira de valeur logique FAUX, tout autre entier servant de valeur logique VRAI. On pourra faire des opérations arithmétiques d'addition, de soustraction, de multiplication, de division, des comparaisons d'éléments et des opérations booléennes ET, OU, NON sur des valeurs logiques.

A côté des éléments existeront les repères (cf la notion de pointeur et les "noms" de [12]) : à chaque instant, un repère R peut repérer un élément : après l'affectation à R d'un élément E, R repère E jusqu'à la prochaine affectation à R.

Face permet l'utilisation de rangée d'éléments et de rangées de repères, qui sont des tableaux unidimentionnels de borne inférieure égale à 1. Une composante d'une rangée t est déterminée par son indice i : t[i] est donc un élément ou un repère selon que t est une rangée d'éléments ou de repères. Une rangée de repères t est dite repérer la rangée d'éléments tl telle que, pour tout indice i, t[i] repère tl[i]. En particulier, la procédure standard analyse affecte une rangée d'éléments à une rangée de repères.

Un programme FACE est essentiellement formé de procédures qui peuvent être sans résultat ou avoir comme résultat un élément ou un repère, ou encore, dans le cas de procédures standard ou écrites en code machine, une rangée d'éléments ou de repères. Le passage d'un compilateur traduisant un langage L en un langage L' à un compilateur traduisant L en un autre langage L" devrait, dans bien des cas, pouvoir être effectué en réécrivant seulement un petit nombre de procédures. Les procédures appl et anapplyse ont pour paramètre une rangée de procédures.

La notion de repère permet de clarifier le passage des paramètres aux procédures [19] notamment "d'appel par nom" [11] : si un paramètre est un repère, la procédure peut modifier l'élément

qu'il repère. En particulier, une procédure ω_N d'une rangée de procédures modifie en général le contenu des cellules au sommet de la pile : ces cellules sont des paramètres de type repère ; au cas où l'un des paramètres est une rangée en pile, il est du type rangée de repères.

Face permet l'utilisation et la construction d'un certain nombre de tables, par exemple de tables d'identificateurs. Dans une telle table les indicatifs sont les identificateurs qui permettent l'accés aux renseignements correspondants : type et adresse

identificateur ⟶ (adresse, type)

Pour gagner du temps à la consultation qui est très fréquente dans ce type de table, on utilise les techniques de hashing pour les stocker :

Un certain nombre de procédures standard appelées procédures table (table, table1 ...) ont pour résultat une rangée de repères qui est associée à un indicatif, qui est une rangée d'éléments. En réalité dans un langage à blocs, les identificateurs peuvent admettre plusieurs définitions chacune valable dans un bloc. Aussi on peut associer plusieurs entrées à un même indicatif, une procédure table à 2 arguments :

- une rangée d'éléments qui est l'indicatif,
- un entier indiquant le rang d'apparition de l'entrée sélectionnée parmi celles qui possèdent cet indicatif.

Dans la pratique, une procédure table sera utilisée comme paramètre effectif d'autres procédures standard permettant d'effectuer des opérations sur les tables : adjonction, recherche, suppression, modification. Ces procédures admettent en paramètre une procédure à résultat booléen qui permet de sélectionner l'entrée voulue (filtre) :

Si l'on considère par exemple un langage à structure de blocs et que l'on utilise la table dans un passage ultérieur à sa construction, on peut numéroter les blocs dans l'ordre de leur apparition et associer à chaque déclaration son numéro de bloc.

A chaque consultation, il faut associer à un identificateur sa dernière déclaration appartenant à un bloc ouvert. Les déclarations seront placées dans la table dans l'ordre inverse de leur apparition ; le rôle de la procédure filtre est de tester si le bloc d'une déclaration est actif.

On va donc chercher dans la table la première occurence de l'identificateur (indicatif) donnant à la procédure filtre la valeur vrai.

Tout ceci est permis par les deux procédures suivantes :

- recherche (table, indic, filtre) a pour résultat la rangée de repères table (indic, i) où i est le premier entier tel que la rangée repérée par table (indic, i) vérifie (c'est-à-dire donne un résultat non nul) à la procédure filtre, s'il existe un tel entier sinon appelle la procédure erreur .
- entrée (table, indic, inf, filtre) est un appel de procédures sans résultat qui :
  - s'il n'existe aucun entier j tel que la rangée repérée par table (indic, i) vérifie la procédure booléenne filtre, affecte la rangée d'éléments inf à table (indic, i) où i est le premier entier tel que table (indic, i) ne repère aucune rangée ;
  - s'il existe un tel entier j et si j_o est le premier d'entre eux, pour $i > j_o$ affecte à table (indic, i) la rangée repérée par table (indic, i-1) et affecte inf à table (indic, j_o).

Les procédures écrites en code Machine permettent de compléter ou d'adapter l'ensemble des procédures standard ; on peut par exemple vouloir décrire des passages plus simples qui n'utilisent pas de grammaire et se contentent de traiter séquentiellement le texte sans employer de pile.

1.4. DESCRIPTION DU LANGAGE FACE.

1.4.1. Eléments de base du langage.

1.4.1.1. Symboles de base.

< 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 octal > ::= 0/1/2/3/4/5/6/7
< chiffre décimal > ::= < chiffre octal > /8/9
< chiffre hexadécimal > ::= < chiffre décimal > /a/b/c/d/e/f
< vide > ::=

1.4.1.2. Chaînes

< chaîne > ::= < caractère > / < chaîne > < caractère >
< caractère > ::= < lettre > / < chiffre décimal > / < caractère spécial >

< caractère spécial > ::= "/:/./,/;/(/)/[/]/+/-/x/ /◁/≤/>/≥/=/≠/#/␣

< chaîne fermée > ::= '< chaîne >'

< commentaire > ::= co < chaîne > co

Sémantique : Les chaînes servent à définir les symboles terminaux des grammaires (1.4.3.) et des quantités de type élément (1.4.1. Elles sont alors mises entre apostrophes, de sorte que ' ne peut être un caractère spécial ; mais " est un caractère spécial. D'autres caractères spéciaux peuvent être ajoutés à volonté pour une implémentation particulière. Le blanc n'est pas significatif dans un programme FACE. Le blanc dans une chaîne sera représenté par le caractère␣. Le changement de ligne dans le texte source est noté $ (1.4.3.). Un commentaire peut être placé n'importe où dans le programme Face.

1.4.1.3. Entiers.

< entier > ::= < entier octal > / < entier décimal >/< entier hexadécima

< entier octal> ::= $ < chiffre octal > /< entier octal > < chiffre octal >

< entier décimal > ::= < chiffre décimal > / < entier décimal> < chiffre décimal >

< entier hexadécimal>::= @ < chiffre hexadécimal > / < entier hexadécimal > < chiffre hexadécimal >

Sémantique : Un entier peut être utilisé pour représenter une constante de type élément (1.4.1.5.), comme indice de rangée (1.4.7.4.), et comme numéro de règle (1.4.3.).

1.4.1.4. Identificateurs.

< ident > ::= < lettre > / < ident > < lettre > / < ident > < chiffre >

< idres > ::= # < chiffre > / < idres > < chiffre >

< identificateur > ::= < ident > / < idres >

Sémantique : Un identificateur peut désigner un élément, une rangée d'éléments, un repère, une rangée de repères (1.4.4.), une procédure (1.4.5.), une rangée de procédures (1.4.6)ou un symbole non terminal d'une grammaire (1.4.3.). Un repère permet d'accéder à une valeur (1.4.6.4.). Cette valeur peut être modifiée par une instruction d'affectation (1.4.7.2.). Certains paramètres formels des procédures formant une rangée de procédures devront être désignés par les identificateurs

particuliers # 1, # 2, ... formés du caractère # suivi d'un entier décimal (1.4.6.).

1.4.1.5. Eléments

< élément > ::= < entier > / < chaîne fermée > / nil

Sémantique :Un élément est représenté par un entier (1.4.1.3.) ou une chaîne fermée (1.4.1.2.) ou la valeur nil (1.4.6.). Pour chaque implémentation du langage, on précisera :

- l'entier maximum et la longueur maximum d'une chaîne pouvant représenter un élément,
- le passage d'une chaîne à l'entier représentant la même valeur.

1.4.2. Programme.

< programme > ::= < définition des grammaires > < liste de déclarations > < expression fermée >

Sémantique :

La définition des grammaires (1.4.3.) sert à décrire la syntaxe des langages étudiés. La liste de déclarations(1.4.4.) sert à déclarer les identificateurs dont la portée est le rpogramme.

L'expression fermée (1.4.7.) qui termine le programme est sans résultat.

1.4.3. Définition des grammaires.

< définition des grammaires > ::= < suite de grammaires>

< suite de grammaires> ::= < vide > / < suite de grammaires > < grammaire >

< grammaire > ::= grammaire < suite de règles >

< suite de règles > ::= < règle > / < suite de règles > < règle >

< règle > ::= < non terminal > : < second membre > ; / : < liste de caractères à sauter >

< non terminal > ::= < identificateur >

< second membre > ::= < membre > / < numéro > / < second membre > < membre > / < numéro >

< numéro > ::= < vide > / < entier décimal >

< membre > ::= < terminal > / < non terminal > / < membre > , < terminal > / < membre > , < non terminal >

<terminal> ::= <chaîne fermée>

<liste de caractères à sauter> ::= <caractère>'/ <numéro> /$/ numéro /
<liste de caractères à sauter> /<caractère>'/ <numéro> /
<liste de caractères à sauter> /$ / <numéro>

Sémantique : Les grammaires sont exprimées en notation de Backus où ::= est remplacé par :, / par / <numéro> / et où la dernière alternative est suivie de / <numéro>. De plus les symboles non terminaux sont des identificateurs, les terminaux sont placés entre guillemets, deux symboles étant séparés par une virgule et chaque règle suivie d'un point virgule. Les numéros servent à repérer les règles. Lorsqu'un numéro est sans importance pour le programme, il peut être laissé vide. Une grammaire est utilisée par la procédure standard analyse, qui possède deux paramètres :

- un élément dont la valeur entière est un numéro de règle ; la grammaire utilisée par l'analyse sera celle qui contient cette règle et l'axiome utilisé sera le premier membre de la règle ;

- Une rangée de repères à laquelle sera affectée le résultat de l'analyse. La procédure effectue l'analyse de la chaîne de caractères lue sur un organe d'entrée fixé, pour la grammaire et l'axiome déterminés par son premier paramètre ; le résultat de l'analyse est une ramification sur l'ense des numéros de règles, qui est mise sous forme post-fixée tout en supprimant les numéros de règle qui ont été laissés vides par le programmeur ; la rangée d'éléments obtenue est affectée au second paramè Tout non terminal ne doit apparaître qu'une fois comme premier membre d'une règle.

Les règles de premier membre vide permettent de transmettre à la procédure analyse la liste des caractères non significatifs du texte source. Pour tout caractère non significatif, pour chaque fin de ligne (4. 1. 2 du texte source, la procédure analyse insère le numéro correspondant dans la ramification résultat. Il n'y a qu'un membre $ pour une grammaire donné

Exemple :

grammaire en notation de Backus.

<exp> ::= <terme> / <exp> + <terme>
<terme> ::= <facteur> / <terme> x <facteur>

section des grammaires

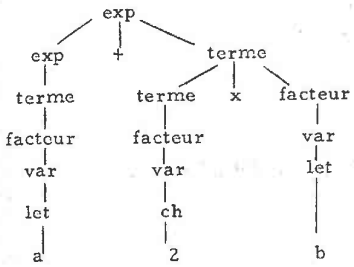
grammaire

```
exp:terme//exp'+', terme/
terme:facteur//terme, 'x',
      facteur/2;
```

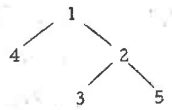

<facteur> ::= <var>
<var> ::= <let> / <ch>
<ch> ::= 2
<let> ::= a/b

```
facteur : var/;
var : let/ / ch/;
ch :'2'/3;
let :'a'/4/'b'/5;
```

Si la chaîne lue est a + 2 x b, l'analyse syntaxique pour cette grammaire a pour résultat :



Mais à cause des numéros vides, la procédure analyse 'simplifie' cette arborescence en



et le résultat est la rangée d'éléments 4, 3, 5, 2, 1

1.4.4. Liste de déclarations.

<liste de déclarations> ::= <vide> /<liste de déclarations><déclaration>
<déclaration> ::= <déclaration d'éléments> ;/<déclaration de repères> ;
<déclaration de rangée d'éléments> ;/ <déclaration de rangée de repères> ;
<déclaration de procédures> ;/ <déclaration de procédure en code> ; /
<déclaration de rangée de procédures> ;
<déclaration d'élément> ::= elt <identificateur> = <élément> /
<déclaration d'élément>, <identificateur> = <élément>
<déclaration de repère> ::= rep <identificateur> <initialisation> /
<déclaration de repère>, <identificateur> <initialisation>
<initialisation> ::= <vide> / := <élément>
<déclaration de rangée d'éléments> ::= rangée(<entier décimal>) elt
<identificateur> =(<liste d'éléments>)/ <déclaration de rangée
d'éléments>, <identificateur> = (<liste d'éléments>)

< liste d'éléments > ::= < élément > / < liste d'éléments >, < élément >

< déclaration de rangée de repères > ::= rangée(< entier décimal >) rep
< identificateur > < rangée initiale > / < déclaration de rangée de repères >, < identificateur > < rangée initiale >

< rangée initiale > ::= < vide > / := (< liste d'éléments >)

Sémantique . Cette partie sert à déclarer les identificateurs dont la portée est le programme. Pour les identificateurs d'éléments ou de rangée d'éléments, elle précise l'élément ou la rangée désigné par l'identificateur. Pour les identificateurs de repère ou de rangée de repères, on peut leur faire repérer une valeur initiale. Le nombre d'éléments ou de repères composant une rangée est indiqué par l'entier qui suit le déclareur rangée.

Exemples : rep mℓ :=1000, n0:=0, nb:=0, niveau:=1, eti:='e000' ;
rangée (6) elt ri = (1, 2, 3, 4, 5, 6) ;
rangée (50) rep ad, cours ;

1.4.5. Déclarations de procédure .

1.4.5.1. Déclaration de procédures.

< déclaration de procédure > ::= proc < résultat > < identificateur > =
< corps de procédure >

< résultat > ::= < vide > / elt / rep

< corps de procédure > ::= (< liste de paramètres formels >) < expression > /
(< vide >) < expression >

< liste de paramètres formels > ::= < liste de paramètres formels > ;
< déclareur > < liste d'identificateurs > / < déclareur >
< liste d'identificateurs >

< déclareur > ::= elt / rep / rangée elt / rangée rep

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

Sémantique : On peut définir deux sortes de procédures :
- Celles qui possèdent un résultat , leur déclaration contient, avant l'identificateur, le déclareur du type du résultat : élément ou repère.
- Celles qui n'admettent aucun résultat.

Selon le cas, l'expression du corps de procédure est sans résultat ou donne le résultat voulu.

Les paramètres formels, s'ils existent, peuvent être des repères, des rangées de repères, des éléments, des rangées d'éléments.

Les procédures peuvent être récursives.

Exemples :

```
proc elt to = (elt type) type = 'réel' :2, 1 ;
proc decrement = (elt ai, type, resint) résint = 0:( ), ai ='*':(mℓ := mℓ -1),
      (mℓ := mℓ - to(type));
proc restaureg = (rep ℓ ; elt k)
      k > ℓ : ( ), (ordre ('ℓw';1;'o';mℓ ; niveau); mℓ := mℓ - 1 ;
      restaureg (ℓ-1;k));
```

1.4.5.2. Déclaration de procédures en code .

< déclaration de procédure en code > ::= code < rescode > < identificateur >
= < corps de pro code >

< rescode > ::= < résultat > / rangée rep / rangée elt

< corps de procode > ::= (< liste des types des paramètres >) / (< vide >)

< liste des types des paramètres > ::= < decode > < liste d'identificateurs > /
< liste des types des paramètres > ; < decode > < liste d'identificateurs

< decode > ::= < déclareur > / proc

Sémantique : On peut déclarer des procédures qui sont directement écrites en code.

Leur écriture dépend directement de l'implémentation du langage Face. En plus des résultats élément ou repère elles peuvent avoir pour résultat une rangée de repères et leurs paramètres peuvent être des procédures.

Exemple :

```
code rangée rep rechtable = (rangée rep ind ; proc filtre).
```

1.4.6. Déclaration de rangée de procédures :

< déclaration de rangée de procédures > ::= rangée (< entier décimal >) proc
< identificateur > ; < liste de procédures de la rangée >

< liste des procédures de la rangée > ::= < identificateur > [< entier décimal >] =
< désignation de la procédure > / < liste de procédures de la rangée > ;
< identificateur > [< entier décimal >] = < désignation de la procédure >

< désignation de la procédure > ::= (< en tête >) < expression > / < identificateu
[< entier décimal >]

< en tête > ::= < entier décimal > , < entier décimal > /
< entier décimal >, rangée, < entier décimal >

Sémantique :

Une déclaration de rangée de procédure définit les procédures de la rangée, notées par l'identificateur de la rangée indicée par un entier, grâc aux désignations de procédures. Une désignation de procédures peut être explicite (en tête et expression sans résultat) ou désigner une procédure appartenant à la même rangée ou à une autre rangée déclarée précédemment. Les indices ne sont pas nécessairement croissants et certains d'entre eux peuvent être absents ; dans ce cas la procédure correspondante n'effectue auc action.

Les rangées de procédures sont formées de procédures sans résul Il n'existe pas d'identificateur indicé de type procédure (1.4.7.4.)

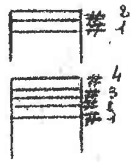
Une rangée de procédures ne peut être un paramètre formel. Une rangée de procédures ne peut donc être utilisée que comme paramètre d'une procédure standard et les seules procédures standard actuellement prévues avec ce type de paramètre sont appl et anapplyse.

Les paramètres formels des procédures d'une rangée sont tous des identificateurs formés du caractère # suivi d'un entier (1.4.1.4.). Si l'en tête est (K, L), ils sont tous de type repère. Si l'en tête est (K, rangée, L ils sont de type repère sauf # k (voir plus bas). Dans les deux cas, leur nombre est le plus grand des entiers K et L.

La procédure appl admet deux paramètres : une rangée d'éléments rel et une rangée de procédures rp. Elle utilise une pile. Elle considère successivement les éléments de la rangée rel et appelle la procédure de la rangée rp dont l'indice est égal à l'élément considéré. Si l'en tête de procédure est (K, L) : les K paramètres # 1, ..., # K désignent, de bas en haut", les K cellules supérieures de la pile ; si L > K, on crée L - K nouvelles cellule au sommet de la pile # K+1, ..., # L ; si L < K, après exécution de la procédure seules les cellules # 1, ..., # L subsistent sur la pile.

Exemples : entrée pile sortie

(4, 2)



(3, 4)

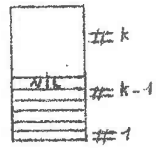
Si l'en-tête de procédure est (K, rangée, L), # K est un paramètr de type particulier qui peut désigner successivement une rangée puis un repère.

La pile doit alors contenir au moins une cellule repérant nil : (1.4.1.5.). Désignons par c la plus haute de ces cellules.

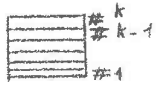
Au départ, # K désigne la rangée formée des cellules situées au dessus de c. Dès qu'une affectation est faite à # K, il désigne un repère qui est la cellule c, et la rangée précédemment désignée par p_k devient donc inaccessible.

Si K > 1, # 1, ..., # K-1 désignent, de bas en haut, les k-1 cellules se trouvant en dessous de c.

Exemple.



à l'entrée de la procédure



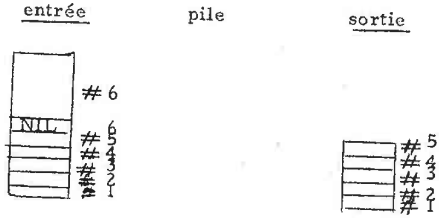
après # K := ....

Si L > K, # K+1 ..., # L désignent les cellules situées au dessus de c, à la sortie de la procédure, # L est le sommet de la pile. Certaines de ces cellules appartiennent donc à la rangée désignée par p_k : tant que p_k désigne une rangée # k+i est synonyme de # k[i], si i est au plus égal à la longueur ℓg de la rangée ;

Lorsque K - L > ℓg, K - L - ℓg cellules sont créés au sommet de la pile.

Exemples :

a)



b)

c)

d)

La procédure anapplyse admet trois paramètres de types respectifs élément, rangée de repères, rangée de procédures ; anapplyse (n;struct;rproc)

équivaut à analyse (n ; struct) ; appl ; rproc) (1.4.3.).

Exemple : voir l'exemple donné au paragraphe 5.

1.4.7. Expression fermée.

< expression fermée > ::= (< liste de déclarations locales >< fin d'expression fermée >)

< fin d'expression fermée > ::= < vide > / < suite d'instructions >

Sémantique : Une expression fermée sert à définir un calcul et à fournir un résultat éventuel qui peut être un élément ou un repère. Les déclarations locales (1.4.7.1.) définissent des repères ou rangées de repères dont la portée est l'expression fermée. Le calcul est défini par la suite des instructions (1.4.7.2.) et seule la dernière "instruction" (1.4.7.2.) peut avoir un résultat ; ce résultat (élément ou repère) est alors celui de l'expression fermée.

. . .

Une expression fermée peut être vide. Elle a alors la forme ( ).

Exemples : (r = 1 : 0, 'erreur')
(anapplyse(1 ; chaîne ; lex) ; appl (chaîne ; gen))
(rangée (4) rep r = recherche (table ; pl ; b) ; rep co ;
r[1] = 'r' : (co := '4d'), (co:='1w');
ordre (co ; '14";0 ; r[2] ; r[3]))

1.4.7.1. Liste de déclarations locales.

< liste de déclarations locales > ::= < vide > / < liste de déclarations locales >; < déclaration locale >

< déclaration locale > ::= < déclaration locale de repères > / < déclaration locale de rangée de repères >

< déclaration locale de repère > ::= rep < identificateur >< valuation > / < déclaration locale de repère >, < identificateur > < valuation >

< valuation > ::= < vide > / = < secondaire > / := < expression simple >

< déclaration locale de rangée de repères > ::= rangée (< entier déc
rep < identificateur > < valrang > / < déclaration locale de rang
repères >, < identificateur > < valrang >

< valrang > ::= < vide > / = < primaire > / := (< liste d'expressions si

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

Sémantique : Les déclarations locales déclarent des identificateurs
repère ou de rangée de repères qui ont pour portée l'expression fer
commençant par ces déclarations.

Si l'identificateur x est déclaré dans une expression fer
contenue dans le domaine de validité d'un même identificateur x, ce
dernier perd sa validité pour la durée de l'expression fermée conte
la nouvelle déclaration de x.

Pour une rangée, l'entier qui suit rangée précise sa tai
Dans le cas où la valuation (resp le valrang) commence par =, le
secondaire (resp. primaire) doit avoir pour valeur un repère (resp.
une rangée de repères dont la taille est celle de la rangée déclarée)
l'identificateur déclaré désigne alors cette valeur pendant toute
l'expression fermée. Dans le cas où la valuation est vide ou comme
par :=, la déclaration crée un nouveau repère (resp. une nouvelle
rangée de repères) qui n'existe que jusqu'à la sortie de l'expression
fermée et y est désignée par l'identificateur ; si la valuation (resp.
le valrang) commence par :=, l'expression simple doit (resp. les
expressions simples doivent) avoir une valeur de type élément et la
déclaration fait repérer cette valeur (resp. la rangée des valeurs de
expressions qui doivent être en nombre convenable) par le repère cr

Comme le nom créé n'existe que dans l'expression ferm
il ne doit pas être utilisé à l'extérieur. C'est par exemple une erreu
de déclarer une procédure par

proc rep pr = (rep x ; ... ; x)

Exemples : rep étiq := creeti + 1 ; rangée (3) rep r = recherche (table
id ;

rep t, cod ;

1.4.7.2. Suite d'instructions :

< suite d'instructions > ::= < vide > / < suite d'instructions > < instruction >

< instruction > ::= ; < instruction d'affectation > / ; < expression >

< instruction d'affectation > ::= < secondaire > := < expression >

Sémantique : Lorsqu'une instruction est une affectation, elle est sans valeur.

Lorsqu'elle est une expression, elle a pour valeur le résultat de l'expression s'il existe ou est sans valeur sinon.

Dans une affectation, l'expression doit avoir un résultat qui soit un élément ; le secondaire doit avoir pour valeur un repère (1.4.7.5).

Exemple : ad := ad+2 ;
to := type = 'réel' : 2, 1 ;
info [3] := adr + 4 x (c-1) ;
2 := compacter ('b' ; 's' ; 'x' ; 0);

1.4.7.3. Expression :

< expression > ::= < expression simple > / < expression simple > :
< expression simple >, < expression >

Sémantique : Soit une telle expression de la forme £1 ; £2, £3 ; alors £1 soit avoir une valeur de type élément ; £2 et £3 doivent toutes les deux être sans valeur ou toutes les deux avoir une valeur de même type élément ou repère. La valeur de cette expression est calculée de la façon suivante : si £1 a une valeur différente de zéro, alors elle est égale à la valeur de £2, sinon elle est égale à la valeur de £3.

Exemples :

# 1 = 'r' :(col:=' ¿d'), (col:='¿w')
resint = 0 :( ) , ai = 1 : (m¿ := m¿ -1), (m¿ := m¿ - to (type))

1.4.7.4. Expression simple.

< expression simple > ::= < terme booléen > / < expression simple > ∨
< terme booléen >

< terme booléen > ::= < facteur booléen > / < terme booléen > ∧
< facteur booléen >

< facteur booléen > ::= < primaire booléen > / ¬ < primaire booléen >

<primaire booléen> ::= <expression arithmétique> <opr> <expression arithmétique> / <expression arithmétique>

<opr> ::= = / < / ≤ / > / ≥ / ≠

<expression arithmétique> ::= <terme> / <expression arithmétique> <opa> <terme> / <opa> <terme>

<opa> ::= + / -

<terme> ::= <facteur> / <terme> <opm> <facteur>

<opm> ::= x / /

<facteur ::= > secondaire> / <élément> / <expression fermée>

Sémantique ;

Les priorités des opérateurs sont par ordre décroissant :

x /

\+ -

=, ≠, <, ≤, >, ≥

¬

∧

∨

De plus, en cas d'égalité de priorité, tous les opérateurs opèrent de gauche à droite.

Une expression qui contient un opérateur a pour valeur un élément. Les opérateurs arithmétiques +, -, x, / ont leur signification habituelle sur les entiers. Pour les opérateurs booléens ¬, ∨, ∧, les opérandes sont d'abord convertis en valeur logique, 0 donne FAUX et tout autre entier VRAI ; le résultat de l'opération booléenne est ensuite converti en un entier : 1 pour VRAI et 0 pour FAUX.

Le résultat d'une expression fermée a été étudiée en (1.4.7.). Lorsqu'une expression simple a pour valeur un repère (c'est alors nécessairement une expression fermée ou un secondaire) et qu'elle se trouve dans un contexte demandant un élément (second membre d'affectation, opérande d'opérateur, paramètre de type élément...) le repère est remplacé par la valeur qu'il repère.

Exemple : (rep X = elt rep (Y) ; X < 2 : X, ML+1) + Z

1.4.7.5. Secondaire .

<secondaire> ::= <primaire> / <identificateur> [<indice>]

<primaire> ::= <identificateur> / <appel de procédure>

<indice> ::= <expression>

<appel de procédures> ::= <identificateur> (<liste de paramètres effectifs>) / <identificateur> (<vide>)

<liste de paramètres effectifs> ::= <paramètre effectif> / <liste de paramètres effectifs> ; <paramètre effectif>

<paramètre effectif> ::= <expression>

Sémantique :

Un identificateur indicé a pour valeur un élément ou un repère selon le type de l'identificateur qui doit, désigner une rangée d'éléments ou de repères ; l'indice détermine l'ordre du résultat (élément ou rangée) dans ce rang.

Le type d'un identificateur est précisé par sa déclaration (1.4.4. et 1.4.7.).

Dans un appel de procédures, l'identificateur est celui d'une procédure déclarée (1.4.5.1), en code (1.4.5.2) ou standard. Les paramètres effectifs doivent correspondre en nombre et en type aux paramètres formels pris dans le même ordre. Ils peuvent être de type élément, repère, rangée d'éléments, rangée de repères dans tous les cas, procédure pour les procédures en code ou standard, rangée de procédure dans le seul cas des procédures standard. Un paramètre qui n'est ni de type élément, ni de type repère est nécessairement réduit à un identificateur.

1.5. EXEMPLE :

Ce paragraphe donne un aperçu du programme FACE décrivant compilateur d'Algol 60 en Symbol 10070. On traite en partie la structure de blocs et les déclarations de types ainsi que l'instruction d'affectation. La compilation est faite en deux passages par appels successifs des procédures anapplyse puis appl, le premier passage réalisant l'analyse syntaxique et l'étude lexicographique par l'intermédiaire d'anapplyse et de la rangée de procédures lex, le second terminant la génération par l'intermédiaire de app et de la rangée de procédures gen.

Grammaire	Commentaire.
bloc non éti : début, partie dec, fin de bloc/5; début :"début"'/6; partie dec : dec// partie dec, ', ', dec ; dec : dec type/7/ decaig // dectab//decproc/; dectype : dectype, ', ', idec/8/type, idec/8; idec : id/ ;	partie de la grammaire Algol 60 engendrant blocs et déclarations de type, décrits en langage FAC
ins de base sans éti:insaf/40/insallera// insproc//insvide/; insaf:partie gauche, ':=', expr/51/ partie gauche, ':=', insaf/52 ;	instruction d'affectation.
rangée(2000) rep chaîne ;	chaîne est la rangée de repères résultat de la procédure analyse ou anapplyse.
rangée (100) rep ad, cours ;	ad est une rangée de repères permettant pour chaque bloc, de repérer l'adresse de la première mémoire libre après les réservations en zone statique. cours est une rangée de repères permettant de connaitre les blocs en cours d'activation.
rep no :=0, nb:=0, mℓ :=0, niveau :=1;	no repère le numéro du bloc en cours d'activation. nb indique le maximum des numé du bloc déjà rencontrés. niveau est le niveau de la procéd en cours.
rangée (3) rep acclib := (1, 1, 1)	acclib contient des booléens indicateurs d'occupation des regi
rangée (3) elt acc = (8, 10, 12)	acc numéro des registres.

ngée (120) proc lex ;	étude lexicographique
x[5] = (2, 0) d[no] := mℓ+1;mℓ := # 1;no := # 2);	fin d'un bloc
x[6] = (0, 2) # 1 :=mℓ ; # 2 := no;nb:= nb+1;no :=nb);	début d'un bloc
x[7] = (1, 0) ( ) ;	fin d'une déclaration de type déclaration d'une variable simple: transport en table d'identificateurs et incrémentation du compteur mℓ.
x[8] = (2, 1)	
rangée (4) rep inf :=(no, niveau,#1, mℓ); ntrée(table;#2;inf; f);mℓ :=mℓ +to(# 1);	(Pour les autres règles citées rien n'est à faire à l'étude lexicographique)
angée (120) proc gen ; en[5] = (1, 0) ours[no] := 0;no:= # 1);	génération : fin d'un bloc.
gen[6] = (0, 1) (# 1 :=no;nb:=nb+1;no:=nb;mℓ :=ad[no]; ours[no] :=1;ordre('ℓw';'4';0;ad[# 1] -1;niveau); ordre('stw';'14';0;ad[no]-1;niveau));	
gen[40] = (2, 0)libérer (# 1);	fin d'une affectation multiple ou simple.
gen[51] = (10, 2)	début d'une affectation multiple ou simple gen[51] doit établir les conversions éventuelles de type entre l expression à affecter et la première partie gauche, puis effectuer l'affectation ce qui permet d'avoir, dans tous les cas, la valeur de l'expression dans un registre. Le numéro de ce registre et le type de l'affectation sont seuls conservés.
gen[52] = (7, 2) (rep co; # 2;# 7;erreur ( ), (co := # 2 ='r':'std', 'stw'; libérer (p3);décrement(# 3; # 2;# 4); # 3 = 0;ordre('co';# 6;0;# 1;# 5), (ordre('ℓw';'14';0;# 1;# 5'; ordre('co';# 6;'*';14;0))));	affectation successives.
proc elt to = (elt type) type = 'r':2, 1;	
proc libérer = (elt adr) testacc(adr)=1:(acclib[adr]:=1), ( );	
proc elt testacc = (elt adr) (adr = 8)∨(adr=10)∨(adr = 12); 1, 0;	

```
proc decrement=(elt ai, type, resint)
resint=0:( ), ai=1:(m↓:=m↓ -1),
(m↓ := m↓  -to(type));

proc elt f=(elt x) no > x:0,
no = x:erreur( ), 1.

(anapplyse(1;chaîne;lex);appl(chaîne;gen))
```

Note :

ordre est une procédure standard plaçant dans un tampon de sortie une partie d'instruction générée. Ses cinq arguments sont, dans l'ordre, pris comme code opération, numéro de registre, marque d'adressage indirect, adresse et numéro de registre d'index.

CHAPITRE 2

GENERALITES SUR LE COMPILATEUR FACE

Le compilateur FACE est un programme écrit en FACE O (définition en Annexe 1) qui a pour donnée un programme FACE qu'il traduit en langage assembleur SYMBOL.

2.1. ORGANISATION GENERALE DU COMPILATEUR.

2.1.2. Les passages du compilateur.

. En FACE O, langage d'écriture de ce compilateur (Annexe 1), la procédure standard ANAPPLYSE (§1.12. et 143) n'existe pas : aussi on est obligé de séparer l'analyse syntaxique du texte source et la génération du texte objet. Un premier passage sera réservé à l'analyse syntaxique et effectué par la procédure standard ANALYSE. (§1.1.2. §1.4.3 et Annexe 1).

. L'étude lexicographique pose un autre problème :

En FACE (§1.4.) dans un corps de procédure, on peut utiliser des identificateurs globaux non encore déclarés.

Exemple : REP PROC F = (REP Y)(X:=G(Y); ...);
REP X ;
ELT PROC G = (REP Z) .... ;

Aussi nous ferons un passage pour l'étude lexicographique des identificateurs globaux.

. On peut alors penser effectuer les réservations pour les identificateurs locaux à ce passage :

Remarquons d'abord qu'aucun problème de définition ne se pose pour ces identificateurs : les déclarations locales (§1.4.7.1.) ne contiennent pas de déclarations de procédures. Considérons cependant l'expression arithmétique suivante :

Y x 3 + (rep X ; ... ; X)

Le repère local X ne peut être alloué que si l'on connait la place prise par le résultat intermédiaire Y x 3. C'est pourquoi objets locaux et résultats intermédiaires doivent être alloués en même temps .

L'étude lexicographique des identificateurs locaux sera faite au 3ème passage, en même temps que les allocations des résultats intermédiaires et la génération du texte objet.

La compilation est donc effectuée en 3 passages :

a) Le premier pour l'analyse syntaxique :

La procédure ANALYSE qui la réalise, utilise une grammaire décrivant la syntaxe du langage FACE. Cette grammaire est codée dans un fichier disque.

Elle est écrite en Annexe 2 en notation de Backus :
- les numéros des règles (§1.4.3.) sont placés entre les règles correspondant à un même non terminal.
- L'axiome <programme> est déterminé par le numéro de règle 120.

Le résultat de l'analyse (§1.1.2. et 1.4.3.) est placé dans la rangée de repères CHAINE (§2.5.).

b) Le 2ème passage réalise essentiellement une étude lexicographique du programme source (sauf celle des identificateurs locaux).

Cette étude lexicographique construit deux tables en adressage dispersé (hashing) gérées par des procédures standard FACE O (Annexe 1.2.).

a) Une table pour les identificateurs : TABLE (§2.4.2.).
b) Une table pour les symboles (terminaux et non terminaux) des grammaires décrites dans le programme source (§1.4.3.).: TABLE 1(§2.4.1.).

Ce passage permet :

1) d'entrer en TABLE les identificateurs globaux et les paramètres formels lors de leur déclaration (déclarations globales §1.4.4.).
2) d'allouer les objets désignés par ces identificateurs.
3) d'entrer en TABLE 1 les symboles (terminaux et non terminaux) des grammaires.
4) de générer les directives SYMBOL qui créeront à l'assemblage du programme objet les constantes et réserveront les places des variables globales (§2.3.2.).

c) Le 3ème passage permet :
1) d'entrer en TABLE les identificateurs locaux et de les allouer (fin de l'étude lexicographique).
2) d'allouer les résultats intermédiaires.
3) de construire les ramifications représentant les grammaires (§2.2.1.) ainsi que les tables représentant les rangées de procédures (§2.2.3.).
4) de générer le texte objet sur disque par la procédure standard ORDRE (Annexe 1.2.).

2.1.2. Définition du compilateur.

Le compilateur FACE est défini :
a) par les déclarations des repères et rangées de repères qu'il utilise (§2.5.).
b) par la rangée de procédures LEX qui décrit le 2ème passage (§3)
c) par la rangée de procédures GEN qui décrit le 3ème passage (§4).
d) par un certain nombre de procédures utilisées dans LEX et GEN (§5).
e) par l'expression finale suivante :
(<affectations initialisant certains repères et rangées de repères globaux>)
ANALYSE (120;CHAINE);
APPL (CHAINE ; LEX);
APPL (CHAINE ; GEN);

2.1.3. Utilisation du compilateur FACE sous BPM.

On l'utilise en tant que processeur du système et on l'appelle par ! FACE.

On obtient alors un programme SYMBOL qui doit être traduit par l'assembleur SYMBOL.

2.2. REPRESENTATION DES INFORMATIONS.

A côté des constantes (éléments et rangées d'éléments) et des noms (repères et rangées de repères), il existe en FACE des informations moins classiques à représenter comme les grammaires (utilisées par les procédures standard ANALYSE et ANAPPLYSE) ou les rangées de procédures

(en liaison avec les procédures APPL ou ANAPPLYSE).

2.2.1. Représentation des grammaires :

Pour représenter une grammaire, il suffit de représenter les symboles (terminaux et non terminaux) et les règles de grammaire.

a) les symboles terminaux et non terminaux reçoivent chacun un numéro d'identification.

- les symboles non terminaux sont numérotés négativement (-1 à -k s'il y a k non terminaux différents dans la grammaire).
- les terminaux sont numérotés positivement. Ils sont rangés consécutivement dans une table (table des terminaux). Dans cette table, les caractères du terminal occupent des octets successifs. Ils sont suivis d'un octet final dont les 8 bits sont à zéro.

  Tout terminal commence au début d'un mot. Le numéro d'identification est le rang de ce mot dans la table des terminaux.

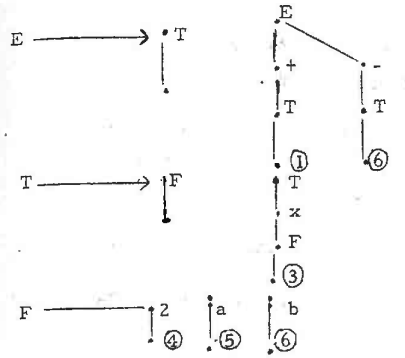
b) Représentation des règles de grammaire :

. La représentation à choisir dépend en principe de la méthode d'Analyse. [14]. Or le choix de cette méthode dépend de la grammaire elle-même. Il existe une procédure standard FACE 0 : FADA (Annexe 1.2.) qui réalise ce choix. A partir de la représentation d'une grammaire par une ramification, FADA cherche une méthode d'analyse déterministe et imprime éventuellement un message d'impossibilité. FADA prépare et complète les données pour cette analyse. On représente donc chaque grammaire pour qu'elle soit une donnée de cette procédure :

. Soit par exemple .

E : T/ /E, '+', T/1/E, '-', T/2;
T : F/ /T, 'x', F/3 ;
F : '2'/4/'a'/5/'b'/6 ;

On représente chaque règle. Chaque symbole non terminal conduit à une ramification dont les noeuds sont étiquetés par les symboles terminaux ou non terminaux.



. On associe à chaque point

- son étiquette
- la liste de ses successeurs : cette liste sera chaînée donc il suffit d'indiquer sa tête.

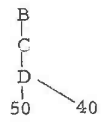
Aussi, pour représenter une règle, il suffit d'associer à chaque point :

- l'entier représentant l'étiquette (numéro d'identification pour un symbole terminal ou non terminal).
- un pointeur vers la liste de ses successeurs (lien vertical)
- un pointeur vers l'élément suivant dans la liste des successeurs à laquelle il appartient (lien horizontal).

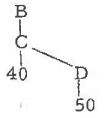
Dans l'exemple ci-dessus, le point étiqueté par + à un point étiqueté par T comme lien vertical et par - comme lien horizontal.

Remarque. On construit la ramification de telle façon que tout noeud fin de règle n'ait pas de lien horizontal. Ceci est exigé par la procédure FADA.

Exemple : La règle A : BC/40/BCD/50 ; sera représentée par



et non par :



Chaque noeud sera représenté effectivement par trois mots mémoire selon la description donnée ci-contre.

c) On regroupe ces ramifications en une seule pour toute la grammaire. Il faut aussi associer à chaque non terminal A la ramification représentant la règle dont il est le premier membre.

Pour cela, il suffit d'un pointeur vers la première racine.

Ces pointeurs sont regroupés dans une table (de taille k) dite table des non terminaux.

Pour un non terminal de numéro d'identification -i, on trouve au rang i dans cette table le pointeur vers le noeud représentant la 1ère racine de la ramification correspondante.

. On définit une table supplémentaire (vecteur de renseignements) qui indique

- les rangs de début des tables, des terminaux, des non terminaux et de la ramification dans la zone mémoire contenant la représentation de la grammaire.
- leurs tailles respectives.
- la liste des caractères non significatifs du texte source (§1.4.

d) Exemple :

Représentons la grammaire donnée en b :

Représentation des noeuds de la ramification :

1er Mot : - bits 1, 2 type du noeud. (terminal 1 0, non terminal 0 0, numéro de règle 0 1)

- bits 3 à 15 à zéro.

- bits 16 à 32 à zéro sauf pour un noeud de fin de règle ; en ce cas ils contiennent le numéro de règle s'il existe sinon -1.

2e Mot : - bits 1 à 15 numéro d'identification pour un symbole, à zéro pour un noeud de fin de règle.

- bits 16 à 32 pointeur vers le lien vertical s'il existe sinon zéro.

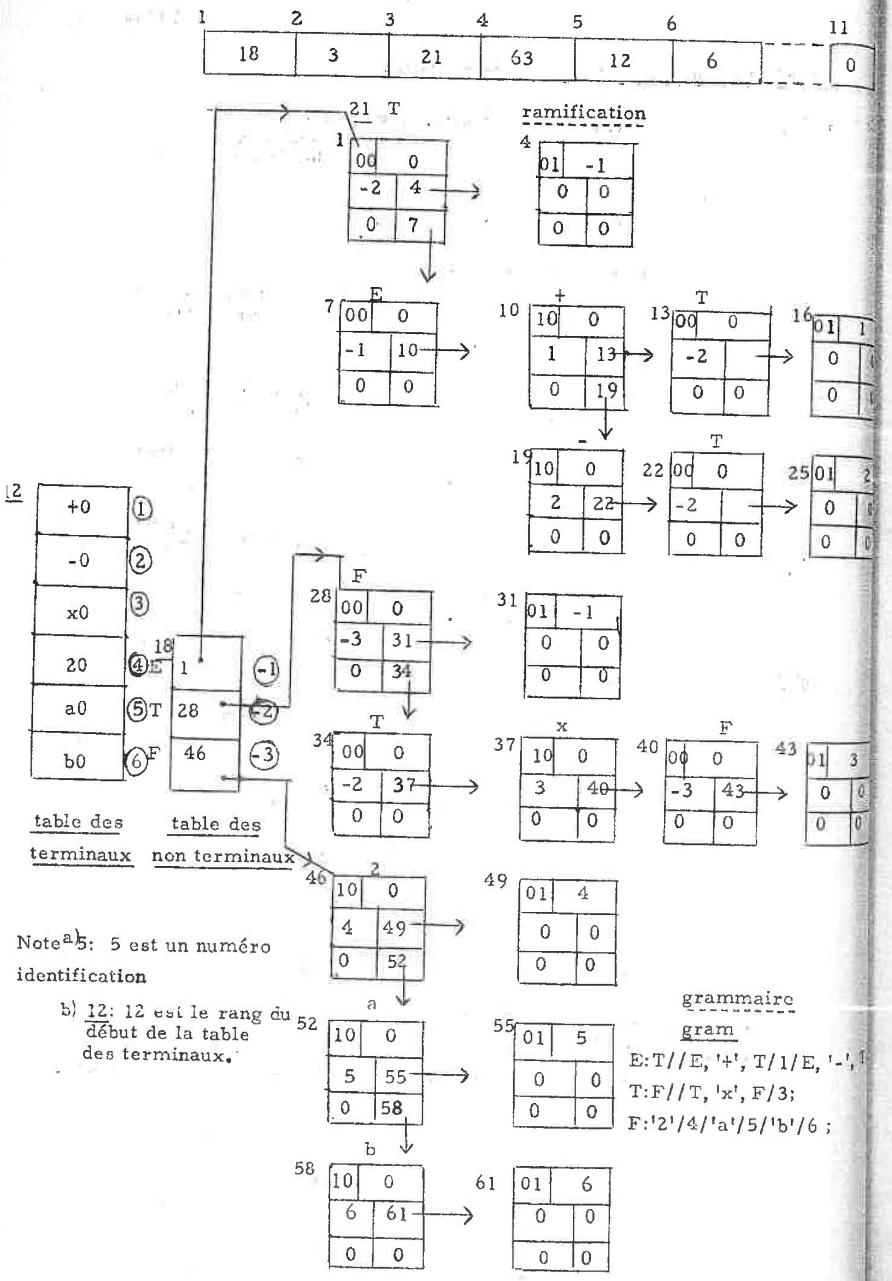
3e Mot : - bits 1 à 15 à zéro ou valeur du compteur de ligne du texte source pour un noeud de fin de règle.

- bits 16 à 32 pointeur vers le lien horizontal s'il existe sinon zéro.

Le vecteur de renseignements contient dans l'ordre :

- le pointeur vers la table des non terminaux
- la taille de cette table
- le pointeur vers la table des terminaux
- la taille de cette table
- le pointeur vers la ramification.
- sa taille
- 4 mots libres pour le préanalyseur
- le nombre des caractères non significatifs du texte source et, s'il y en a,
- leur liste accompagnée du numéro de règle correspondant.

Vecteur de renseignements.



Note a) 5: 5 est un numéro identification

b) 12: 12 est le rang du début de la table des terminaux.

2.2.2. Représentation des éléments, des repères, des rangées d'éléments ou de repères.

a) Les éléments sont représentés par le contenu d'un mot de mémoire.

b) Les repères sont représentés par un mot de mémoire. L'élément repéré est alors représenté par le contenu de ce mot.

Exemple

```
elt X = 1 ;          | 1 |
rep Y ; ... Y := 4 ; | 4 |
```

c) Une rangée d'éléments de taille k est représentée par les contenus de k mots successifs précédés d'un mot contenant k.

Exemple : rangée (4) elt T = (1, 2, 3, 4)

4	1	2	3	4

Une rangée de repères de taille k est représentée par k mots consécutifs précédés d'un mot contenant k.

Exemple : rangée (4) rep R ;

4				

Soit a l'adresse de début de cette zone :

L'adresse a + i permet d'accéder à la i^e composante de la rangée. Cet accès est représenté par chaînage.

Exemple rangée (4) rep R ; ... R[2] := 7 ;

4	7				

2.2.3. Représentation des rangées de procédures.

Seules les procédures APPL ou ANAPPLYSE utilisent les rangées de procédures (§1.4.6.).

1°) Ces procédures doivent réaliser des appels successifs aux procédures de la rangée qu'elles ont en paramètres. Pour cela, elles doivent connaître les adresses de début des corps des procédures de la rangée : chaque rangée de procédure de taille k sera représentée par k+2 mots consécutifs.

- Le 2ème contient k
- Les k suivant contiennent des adresses symboliques de début des corps de procédures de la rangée.

Si a est l'adresse du 2ème mot de la zone et si la procédure d'indice i existe dans la rangée, le mot d'adresse a + i contient l'adresse symbolique du début du corps de la procédure d'indice

2°) Toute procédure d'une rangée, après son exécution doit revenir à la procédure APPL ou ANAPPLYSE qui l'a appelée. Cette adresse retour est communiquée dans le 1er mot de la zone.

Exemple : rangée (4) proc rp

```
rp [1].... ;
rp [4].... ;
rp [3].... ;
rp [2].... ;
```

4	E1	E2	E3	E4

Ei est l'adresse symbolique de début du corps de la procédure d'indice i.

3°) Si la procédure d'indice i n'existe pas, elle doit n'avoir aucune action (§1.4.6.). On a considéré pour chaque rangée une procédure de corps vide. En ce cas, à l'adresse a + i, on trouve l'adresse symbolique de cette procédure vide.

Exemple rangée (5) proc rp

```
rp[1] = ... ;
rp[4] = ... ; RP est l'adresse d'une procédure "vide".
```

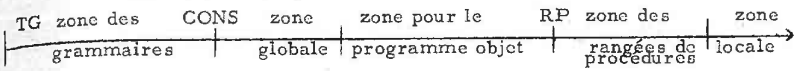
5	E1	RP	RP	E4

2.3. ORGANISATION DE LA MEMOIRE A L'EXECUTION :

. Il s'agit maintenant de savoir où placer en mémoire les diverses informations (allocation mémoire). On regroupe les informations de même nature dans une même zone de mémoire formée de mots contigus. L'allocation est indépendante pour chaque zone : Dans chacune d'elles, un mot est repéré par son rang dans la zone et l'adresse de début de la zone.

. On trouve en mémoire dans l'ordre :
- une zone des grammaires.
- une zone globale contenant les objets permanents (objets globaux) : éléments, rangée d'éléments, repères et rangée de repères.
- une zone pour le programme objet.
- une zone des rangées de procédures.
- une zone locale contenant les résultats intermédiaires, les

objets locaux aux expressions fermées et aux procédures. Cependant les résultats intermédiaires des calculs arithmétiques sont placés en priorité dans 10 registres.



. Toutes les allocations sont statiques: le nombre des objets traités par un programme FACE et leur taille étant connue à la compilation.

Décrivons plus en détail les zones - ce qu'elles contiennent
- la façon dont on les alloue.

2.3.1. La zone des grammaires :

. Cette zone ne contient pas exactement la représentation des grammaires du programme source : celles ci prennent beaucoup de place et seules les procédures standard ANALYSE ou ANAPPLYSE les utilisent. Elles sont dans un fichier à clé sur disque.

Chaque clé permet d'atteindre la représentation d'une grammaire : Connaissant cette clé ANALYSE ou ANAPPLYSE peuvent amener en mémoire la grammaire voulue.

Or ces procédures ne connaissent pas directement la clé mais le numéro d'un règle ayant l'axiome en 1er membre (premier paramètre de ces procédures §1.4.3.).

Les informations nécessaires pour retrouver la clé voulue sont contenues dans la zone des grammaires soit :

par grammaire :
- son numéro de règle le plus élevé ng
- sa clé c

. Exemple :

150	1	200	2	310	3

Il y a 3 grammaires dans le programme source. Si 220 est le premier paramètre d'ANALYSE, la clé de la grammaire utile sera 3.

. La zone des grammaires débute à l'adresse TG.
Le mot d'adresse TG contient sa taille.

Il y a ensuite 2 mots par grammaire contenant successivement les entiers ng et c.

. Elle est allouée au 2ème passage (§2.4.1.). Des directives SYMBOL générées à ce passage la créent à l'assemblage.

2.3.2. La zone globale.

. Elle contient les éléments et rangées d'éléments et aussi les mots représentant les repères et rangées de repères globaux (§2.2.2.).

. Elle est allouée au 2ème passage (§2.1.1.) pour chaque déclaration globale ou pour chaque élément.

Remarque :

On crée ainsi une constante objet par élément noté dans le programme source et non par élément de valeur différente. Mais cela évite d'utiliser une table des éléments.

. Le repère CQG (§2.5.) donne les adresses relatives dans cette zone.

. Elle débute à l'adresse symbolique CONS. Les constantes sont placées et les zones représentant les repères sont réservées à l'assemblage par des directives SYMBOL générées à l'étude lexicographique.

2.3.3. La zone du programme objet comprend la traduction des procédures déclarées dans le programme source et de l'expression finale.

2.3.4. La zone des rangées de procédures.

Contient les tables représentant les rangées de procédures (§2.2.3.)

- Elle est allouée au 2ème passage (§2.1.1.).
- CG repère les adresses relatives dans cette zone (§2.5.).
- Elle débute à l'adresse symbolique RP. Les adresses contenues dans cette zone (§2.2.3.) sont connues à la génération du programme objet. Des directives Symbol générées à ce passage les placent dans cette zone à l'assemblage. Ces directives sont générées derrière le programme objet.

2.3.5. La zone locale.

. Elle contient tous les objets locaux à une expression fermée ou un corps de procédure :

- repères et rangées de repères locaux.

- résultats intermédiaires.

. Ces objets locaux sont alloués comme une pile à la génération (§2.1.) : Pour une expression fermée on alloue les repères et rangées de repères locaux à l'expression fermée, puis les résultats intermédiaires.

Ces objets disparaissent à la fin de l'expression fermée. Le pointeur ML repère toujours la 1ère adresse libre dans cette zone (§2.5.).

. L'allocation est indépendante pour les informations des diverses procédures et de l'expression finale du programme source qui forment dans cette zone des segments indépendants (§2.4.3.).

Pour réaliser l'allocation dynamique des segments de procédures (récursivité des procédures §2.4.3.), on aura un index par segment. Cet index sera placé dans le registre 5.

Bien que ce soit inutile, le segment de l'expression finale sera indexé par le début de la zone locale : cela permet d'avoir le même type d'adresses générées dans cette zone, l'expression finale étant généralement courte.

. Les débordements de la zone locale sont des débordements mémoire et sont donc testés par hardware.

2.4. TRADUCTION D'UN PROGRAMME FACE.

Reprenons les diverses constructions du programme FACE (1.4.) et voyons de façon générale comment le compilateur les traduit .

2.4.1. Traduction des grammaires :

a) Au 2ème passage : (§2.1.2.)

1) On définit la zone des grammaires (§2.3.1.).

- on compte le nombre des grammaires et pour chacune d'elles
  - on attribue une clé numérique
  - on détermine le numéro de règle le plus élevé.
- on génère les directives SYMBOL créant cette zone à l'assemblage.

2) On prépare la représentation de chaque grammaire (§2.2.1.).

. On attribue les numéros d'identification aux symboles termin et non terminaux que l'on conserve dans TABLE1 (§2.1.). Dans cette table l'information a une composante unique qui contient :

- 1er octet:type(terminal (T), non terminal (N))
- 2e : clé de la grammaire contenant le symbole.
- 3e : son numéro d'identification
- 4e : nombre de caractères composant un terminal.

Exemple :

Soit 1 la clé de la grammaire

- pour le terminal "DEBUT" de numéro d'identification 40 :

T	1	40	7

- pour le non terminal A de numéro d'identification -2 :

N	1	2	

. On détermine les tailles des tables des terminaux et non terminaux que l'on conserve dans la rangée TGRAM (§2.5.).

. Afin de connaître la taille du vecteur de renseignements (§2.2. on compte le nombre des caractères non significatifs du texte source. Ces caractères sont indiqués dans la règle de 1er membre vide (§1.4.3.). Cette taille est conservée dans TGRAM.

b) A la génération : Pour chaque grammaire :

1) On construit dans la rangée G (§2) ;

- le vecteur de renseignement
- la table des terminaux
- la table des non terminaux
- la ramification

2) On appelle ensuite la procédure FADA (G, clé) (§2.2.1.b).

FADA considère la grammaire représentée dans la rangée G, cherche une méthode d'analyse déterministe, prépare les données pour cette analyse et écrit ces données dans le fichier des grammaires selon la clé en paramètre.

2.4.2. Traitement des identificateurs.

Lors de la génération, on a besoin d'associer à chaque identificateur l'objet qu'il désigne. Lors des déclarations le compilateur conserve les renseignements à ce sujet dans une table d'identificateurs TABLE (étude lexicographique). Nous avons vu au (§2.1.1.) que cette étude est faite au 2ème passage sauf pour les identificateurs locaux. La construction de la table est liée au fait qu'un identificateur peut admettre plusieurs définitions (langage à structure de blocs).

a) Les identificateurs et la structure de bloc :

. Un identificateur existe dans un bloc qui contient sa déclaration :

En FACE un bloc est une expression fermée (pour les identificateurs locaux (§1.4.7.)), un corps de procédure (pour les paramètres formels), le programme (pour les identificateurs globaux) (§1.4.4.).

. A chaque bloc contenant des déclarations, on associe un numéro (numéro de bloc) comme dans l'expmple suivant.

```
1  2  3        4  5
(  (  (  )  )  (  (  )  )  )
```

Soit dans un bloc donné de numéro k, l'identificateur X déclaré dans un bloc p :

si $p > k$ X n'est pas valide.
si $p = k$ X est valide.
si $p < k$ X n'est valide que si le bloc p contient le bloc k (§1.4.7.1)

Pour vérifier cette dernière condition, on gère une rangée de booléens COURS (§2.5.) telle que dans le bloc k, pour tout $i \leq k$ COURS[i] est vrai si le bloc i contient le bloc k et est faux sinon.

Ceci ne suffit pas si le même identificateur X est déclaré dans plusieurs blocs imbriqués. D'après le (§1.4.7.1), dans le bloc k, l'identificateur X valide est alors celui qui est déclaré dans le bloc dont le numéro est le plus proche de k. Aussi on place dans la table d'identificateurs TABLE, les identificateurs par numéro de bloc croissant (procédure FILTRE Annexe 1). Alors étant donné un identificateur X, l'identificateur de nom X en table est celui du premier bloc p tel que COURS[p] ait la valeur vrai (procédure B Annexe 1).

b) Les renseignements pour un identificateur en TABLE concernent déjà son type ;

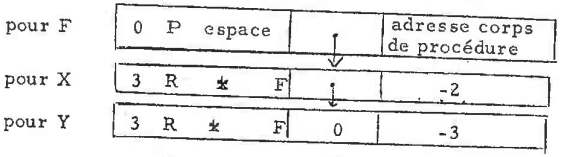
. La plupart des types sont simples :

·repère, élément, rangée de repères, d'éléments ou de procédu

. Pour une procédure, le type comprend non seulement celui de son résultat mais aussi la liste des types de ses paramètres ;

liste qui est représentée par chaînage (§2.4.3.).

Exemple : proc elt F = (rep X, Y) ..... ;

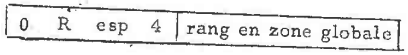


. Pour associer à l'identificateur l'objet interne qu'il désigne, il suffit d'une adresse :

- nom de la zone mémoire
- adresse dans cette zone.

En particulier pour un nom de procédure , c'est l'adresse du début du corps de procédure objet.

Exemple : rep A



Le seul problème vient des identificateurs locaux de repère (ou de rangée de repères) dont la valuation (ou le valrang) commence par = (§1.4.).

Exemple: proc rep f = (elt a, b) .... ; ...
(rep X = f(a, b) + 1 ; ... X:=5 ; ....)

La relation désigné est alors dynamique (tout dépend dans l'exemple du résultat du calcul f(a, b) + 1). En ce cas, on est obligé d'associer à l'objet externe (X dans l'exemple) un mot mémoire en zone locale destiné à contenir le résultat du calcul : soit un pointeur vers le repère désigné par X.

1ère composante : 1er octet — numéro du bloc où est déclaré l'identificateur s'il est local
0 s'il est global

2e octet — type de l'objet désigné : (repère (R), élément (E), rangée de repères (G) ...)

3e octet — si l'objet désigné est atteint par chaînage * sinon espace

4e octet — désignation de la zone mémoire où se trouve l'objet désigné ((4) pour la zone globale (5) pour la zone locale ...
avec une particularité ; la marque (F) pour un paramètre formel.

2° composante : Si c'est une procédure (resp. un paramètre formel): pointeur vers l'information en TABLE du 1er paramètre formel (resp. du paramètre formel suivant).
sinon adresse de l'objet désigné ou adresse de son pointeur.

3e composante : n'existe que pour une procédure ou un paramètre formel : dans le 1er cas elle contient l'adresse symbolique du début de la procédure dans la zone objet;
dans le second, son adresse dans le segment de procédure (§2.4.3.).

On conservera en table l'adresse de ce pointeur ainsi qu'une marque de chaînage *

Exemple :

3	R	*	5	a

2.4.3. Traitement des informations associées aux appels de procédures :

a) Les objets internes d'un appel de procédure sont regroupés dans des segments en zone locale (§2.3.4.).

Les procédures pouvant être récursives (§1.4.5.), chaque appel possède ses propres objets (paramètres effectifs et objets internes) :

• On fait une allocation statique de ces objets indépendants de l'appel. Ils sont alloués à partir d'une adresse fixe et forment un segment de zone locale. Grâce à un index, on effectue à l'appel une translation du segment afin d'utiliser en zone locale une zone mémoire non encore occupée.

• Certains objets locaux sont communs aux segments appelant et appelés :

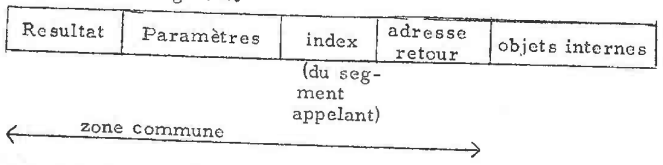
- les paramètres effectifs
- l'adresse de retour
- le résultat

Aussi les segments appelant et appelés enpiètent l'un sur l'autre pour cette zone commune.

Au retour de la procédure, il faudra retrouver l'index du segment appelant : sa valeur est conservée dans la zone commune.

• A chaque instant, il suffit d'avoir accés à un seul ir... car une procédure n'est jamais déclarée à l'intérieur d'une autre procédure.

L'index est placé dans le registre 5. A l'appel, il est initialisé au niveau de la place de l'adresse retour qui se trouve donc à l'adresse 0 du segment.



b) Paramètres effectifs, résultat :

• Les paramètres effectifs ou le résultat de l'appel sont représentés par un mot mémoire qui contient le résultat du calcul du paramètre ou de l'expression du corps de procédure : le problème est analogue aux identificateurs locaux valués par = (§2.4.2.b).

Le mot contient :

- une valeur pour un paramètre ou un résultat élément
- une adresse pour un résultat repère ou un paramètre repère ou rangée (chaînage).

Dans le cas particulier d'un nom de procédure (paramètre effectif d'un appel d'une procédure standard ou écrite en code): le mot contient l'adresse symbolique de début du programme objet correspondant à cette procédure.

• Les vérifications de concordance de type entre paramètres effectifs et formels se font lors de l'appel en suivant en table d'identificateurs (§2.4.2.b) le chaînage des informations des paramètres formels de la procédure appelée.

Remarque : Lors d'un appel d'une procédure à résultat, on doit protéger les registres pouvant contenir des résultats intermédiaires (§2.4.5.). Pour des raisons d'efficacité, on génère au début de la procédure le rangement des 16 registres en zone locale et on les restaure avant le branchement à l'adresse retour.

2.4.4. Traitement des informations associées aux procédures d'une rangée.

. Comme pour les procédures, les objets internes des procédures d'une rangée seront regroupés dans un segment.

Lors d'un appel par APPL ou ANAPPLYSE, le segment de la procédure de la rangée sera placé en zone locale au bon endroit grâce à un index.

. La zone commune aux segments appelant (celui d'APPL ou d'ANAPPLYSE) et appelé comprend uniquement les paramètres, en effet, les procédures d'une rangée :

- sont sans résultat,
- l'adresse retour qui est une adresse fixe dans les programmes APPL ou ANAPPLYSE est placé dans le 1er mot de la zone de la rangée de procédure (§2.2.3.) par les procédures APPL ou ANAPPLYSE.
- Les procédures d'une rangée n'étant pas récursives, il est inutile de conserver la valeur de l'index du segment appelant.

. Les paramètres sont situés sur une pile (§1.4.6.). Les cellules de cette pile sont les objets internes des procédures APPL et ANAPPLYSE. La 1ère cellule de la pile sera à l'adresse 1 dans le segment des procédures APPL ou ANAPPLYSE.

L'évolution de la pile dépend des paramètres effectifs des appels des procédures de la rangée et leur définition dépend de l'en-tête.

a) Pour une en tête (k, ℓ), on génère au début du corps de la procédure de la rangée ayant cet en-tête la diminution de k de l'index. Le premier résultat interne est à l'adresse sup(k, ℓ) + 1 indexé par l'index local.

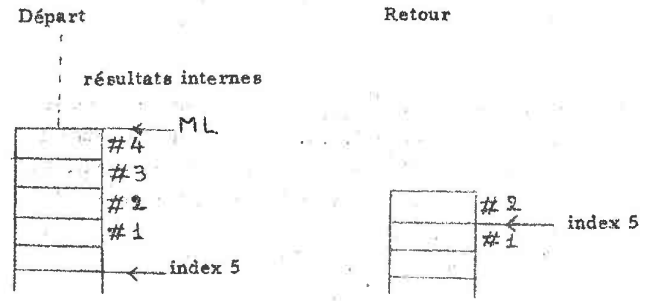
A la fin du corps de procédure de la rangée, on génère l'incrémentation de l'index de ℓ (cf. exemples).

Exemples :

en tête (4, 2) Programme Symbol généré

```
ET:AI, 5      -4      Index := Index-4.
                      étiquette de début du corps.
   :                  corps de procédure
   :
   :
   AI, 5      2       Index := Index + 2
   B          pR      Branchement au point de retour de la rangée.
```

Pile

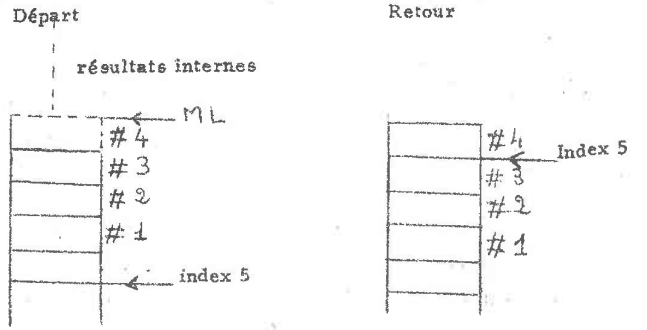


en tête (3, 4)

Programme Symbol généré

```
ET:AI, 5      -3      Index := Index - 3
   :
   AI, 5      4       Index := Index + 4
   B          pR
```

Pile



b) Pour une en-tête (k, rangée, ℓ) , l'identificateur réservé #k désigne au départ une rangée (§1.4.6.) dont la taille ne peut être connue qu'à l'exécution par recherche de la cellule contenant nil. Aussi si on procéde comme en a, on ne sait pas à priori définir le rang du premier résultat interne. On veut de plus accéder aux paramètres #i par i indexé par l'indice local. Aussi on représente les paramètres #i par chaînage en créant à l'entrée de la rangée de procédure n = sup (k, ℓ) cellules fictives pointant sur les cellules #i initiales et on fait pointer l'index local sous ces cellules fictives. Ainsi un paramètre #i est obtenu à partir de la cellule d'adresse relative i et le premier résultat intermédiaire a pour adresse n .

La rangée de repère #k doit être représentée comme les autr rangées de repères aussi la cellule contenant nil doit contenir sa taille.

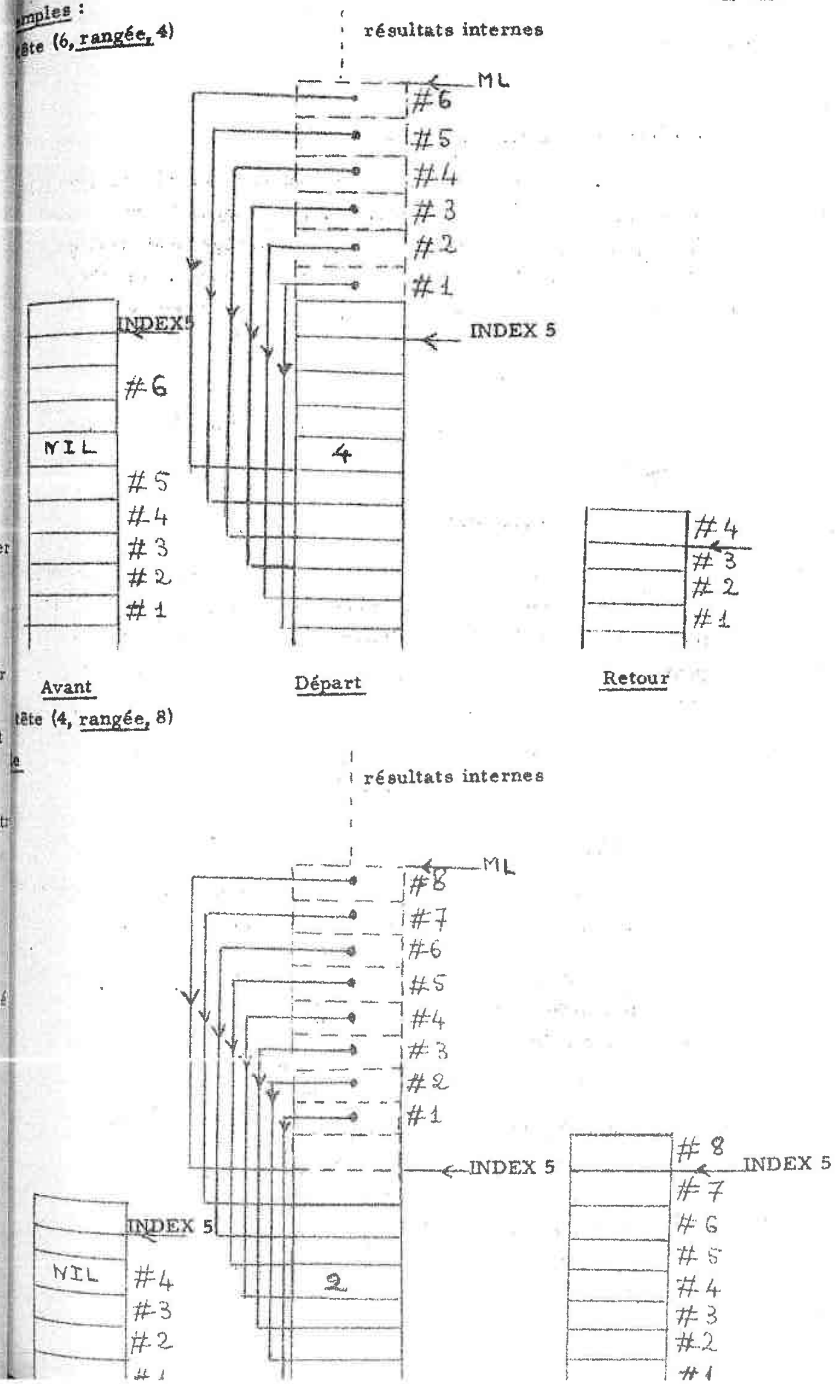
Quand la rangée #k est accessible un drapeau RP est positionné à k(§2.5.). Dès qu'elle devient inaccessible (après une affectation à #k) il est positionné à $\star$. (Ce qui indique en plus que l'on a accés aux repères #i par chaînage car dans le cas a RP est positionné à zéro).

2.4.5. Traitement des expressions :

A chaque expression qui donne un résultat intermédiaire, on associe un mot qui contient

- la valeur de l'expression si c'est un élément
- son adresse si c'est un repère (chaînage).

Ce mot est situé en zone locale (§2.3.5.) ou (uniquement pour les résultats des calculs arithmétiques) est un registre (§2.4.5.3.).



2.4.5.1. Les expressions fermées.

Aux expressions fermées contenant des déclarations lo on associe un numéro de bloc (§2.4.3.). Les objets internes à l'expression, repères, rangées de repères (créés localement ou résultats intermédiaires) sont alloués en zone locale. A la fin de l'expression fermée ils disparaissent grâce au repositionnement du pointeur ML de zone locale à sa valeur initiale du début de l'expression fermée.

Certains problèmes se posent pour le résultat :

a) Considérons l'exemple suivant :

(rep X; ... X := 2;X)

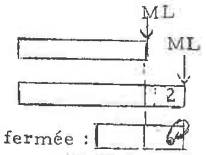
. Cette expression est à résultat repère mais ce repère ne peut être employé que dans un contexte d'élément . Cette conditi ne peut être décelée à la compilation : les paramètres formels peuvent désigner des repères et leur portée locale diffère à chaque appel .

Aussi, On considèrera l'élément repéré en prenant connaissance du contexte (§2.4.5.3).

b) Allocation du résultat :

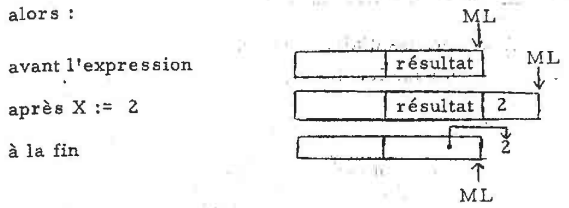
On ne sait qu'il existe un résultat qu'à la fin de l'expre fermée. Il semble donc logique de l'allouer à ce moment.

Si on considère alors l'évolution de la zone locale pour l'exemple ci-dessous :



- ML reprend sa place initiale
- le résultat est alloué en ML
- en ML on établit un pointeur vers le repère résultat et on détruit l 'élément 2 repéré.

Aussi on allouera systématiquement un mot pour le résultat éventuel dès le début de l'expression fermée, ce qui donne alors :



2.4.5.2. Les expressions conditionnelles.

Considérons les expressions de la forme $\mathcal{E}1 : \mathcal{E}2, \mathcal{E}3$ (§1.4.7.3.). On génère le programme objet correspondant selon le schéma suivant :

Calcul de $\mathcal{E}1$
Si $\mathcal{E}1 = 0$ (faux) alors allera e
calcul de $\mathcal{E}2$
allera f
e : calcul de $\mathcal{E}3$
f : suite du programme objet.

Le résultat, s'il existe, est le résultat de $\mathcal{E}2$ ou celui de $\mathcal{E}3$: aussi ces derniers résultats doivent être alloués au même endroit en zone locale. Si $\mathcal{E}2$ et $\mathcal{E}3$ sont l'un de type élément, l'autre de type repère, l'expression est correcte mais le repère est employé dans un contexte d'élément et il faut prendre pour résultat l'élément qu'il repère. Si c'est $\mathcal{E}3$, ce dérépérage se fait normalement après le calcul de $\mathcal{E}3$. Si c'est $\mathcal{E}2$, on ne sait qu'il faut dérépérer qu'après avoir généré la séquence du calcul de $\mathcal{E}3$ aussi dans ce cas le schéma est-il modifié après le calcul de $\mathcal{E}3$ en:

allera g
f : dérépérage de $\mathcal{E}2$
g : suite du programme objet.

Remarque : Un résultat d'expression fermée ou d'expression conditionnelle de type repère est noté 'repère d'expression' pour le compilateur. La grammaire (Annexe 2) autorise des 1er membres d'affectation qui sont des expressions ce que ne permet pas le langage Face (§1.4.7.2). Cette notation permet de vérifier (au 3ème passage) que les 1er

membres d'affectation sont corrects syntaxiquement.

2.4.5.3. Les expressions simples :

a) Les résultats intermédiaires d'un calcul sont placés dans 10 registres ou en zone locale s'ils sont plus nombreux.

b) dérépérage.

Un repère en opérande est dans un contexte d'élément On considère l'élément qu'il repère pour faire le calcul, et ceci le plus tôt possible, soit à la rencontre d'un symbole opératoire. Reprenons l'exemple :

(rep X ; ... X := 2 ; X)x .....

Après ce calcul, la zone locale se présent ainsi :

Si cette expression est opérande, des résultats intermédiaires peuvent être alloués en ML et détruire la valeur 2. Sauf si l'on prend la précaution de "dérépérer" à la rencontre du symbole opératoire x.

c) Si l'expression fermée (rep X ... X := 2; X) est par contre employée dans un contexte de repère, soit comme :
- corps de procédures à résultat repère
- paramètre effectif de type repère.

Alors c'est une erreur.

Aussi pour un résultat employé dans un contexte de repère, on vérifie que ce repère existe dans un bloc plus grand que l'expression fermée.

Il suffit de regarder si l'adresse du repère considéré est bien inférieure à la valeur de ML.

Cette vérification se fait à l'exécution. (à cause de l'utilisation des paramètres formels de type repère).

2.4.6. Traitement des erreurs.

Les erreurs syntaxiques sont décelées par l'analyseur qui imprime un libellé d'erreur correspondant.

Pour les erreurs décelées à l'étude lexicographique et à la génération, la procédure standard erreur (cf. Annexe1.2) imprime le numéro de l'erreur et la valeur des compteurs de caractères CCAR et de ligne CL (§2.5.) du texte source qui sont incrémentés aux 2 passages.

Ces erreurs n'arrêtent pas la compilation mais empêchent l'assemblage du programme objet obtenu.

2.4.6.1. Erreurs décelées à l'étude lexicographique.

1 terminal de plus de 11 caractères ou chaîne notant un élément de plus de 4 caractères ou entier trop grand.

2 Dans la description d'une grammaire, on trouve deux règles de même premier membre.

3 Double déclaration d'un identificateur dans un même bloc

4 Nombre d'éléments définissant (ou initialisant) une rangée d'éléments (ou de repères) globale différent de la taille déclarée.

7 Nom d'une procédure d'une rangée différent de celui de la rangée de procédure correspondante.

2.4.6.2. Erreurs décelées à la génération.

0 Identificateur non déclaré ou non terminal en 2e membre de règle de grammaire qui n'existe pas en 1er membre.

5 Résultat d'un corps de procédure non conforme au résultat déclaré.

6 L'expression simple valuant une déclaration locale de repère (de rangée de repères) commençant par := ne peut être un élément (une rangée d'éléments).

8 Une procédure d'une rangée d'indice supérieur à la taille de la rangée.

9 Type d'un paramètre effectif non conforme à celui du paramètre formel

10 Nombre de paramètres effectifs différent du nombre de paramètres formels

11 Un identificateur réservé utilisé comme nom de procédure.

12 Un identificateur indicé n'est pas un identificateur de rangée.

13 Les expressions formant les 2 branches d'une expression conditionnelle ont des types non conformes.

14 Secondaire valuant une déclaration locale de repère (de rangée de rppère) n'est pas un repère (une rangée de re repère).

15 expression en indice ne pouvant être un élément

16 nombre d'expressions simples initialisant une déclaration locale de rangée de repères non conforme à la taille de la rangée.

17 Une expression à résultat qui n'est pas la dernière dans une expression fermée .

18 Premier membre d'une affectation incorrect syntaxiquemen

19 expression fermée finale à résultat

20 corps d'une procédure d'une rangée ayant un résultat

21 opérande pouvant être élément.

22 expression finissant une expression fermée de type incorre

2.4.6.3. Erreurs décelées à l'exécution :

1 utilisation d'un repère n'existant plus (§2.4.5.3.).

2 indice négatif ou nul.

3 indice supérieur à la taille de la rangée

4 débordement arithmétique lors d'une multiplication.

(Les autres erreurs de débordement arithmétiques sont récupérées par le moniteur BPM par déroutement hardware)

2.5. LES DECLARATIONS GLOBALES DU COMPILATEUR.

REP CCAR,	compteur de caractère du texte source
CETI,	compteur permettant de créer les étiquettes du programme objet
CL,	compteur de ligne du texte source
CG,	pointeur dans la zone des rangées de procédures (§2.3.4.).
CGRAM,	compteur du nombre des grammaires
CQG,	pointeur dans la zone globale (§2.3.2.).
ERR,	booléen indiquant s'il y a une erreur à la compilation.
ETI,	a pour valeur la prochaine étiquette à générer si elle existe sinon 0
NB,	retient à la génération le numéro de bloc le plus élevé (§2.4.2.).
NO,	numéro de bloc actif (2.4.2.).
ML,	pointeur dans la zone locale (§2.3.5.).
RP,	retient la forme de l'en-tête d'une procédure d'une rangée (espace ou entier k ou *) (§2.4.4).
SOMMET;	rang dans TILREG du 1er registre intermédiaire libre (§2.4.5.3.).

RANGEE (?) REP

CHAINE ;	reçoit le résultat de l'analyse (§2.1.2.).
COURS ;	booléens définissant les blocs actifs (§2.4.2.).
G;	sert à construire la représentation d'une grammaire (§2.2.1. et 2.4.1.).
TG;	sert à construire la zone des grammaires (§2.3.1)
TGRAM;	retient les renseignements trouvés à l'étude lexicographique sur les grammaires (§2.1.2. et 2.4.1.).
Z;	sert à construire la zone des rangées de procédures (§2.3.4.).
TILREG;	contient les numéros des registres pouvant contenir des résultats intermédiaires.

REP TRAV, TRAV2, REG, REG1, I, J;

RANGEE(2) REP INF2, INDIC; repères et rangées de travail

RANGEE(3) REP INF3 ;

RANGEE(4) REP INF4 ;

CHAPITRE 3

L'ETUDE LEXICOGRAPHIQUE

Rangée de procédures LEX

3.1. SYMBOLES DE BASE.

3.1.1. Lettres.

```
LEX[131] = (0,1) (P1:='A';CCAR:=CCAR+1);
LEX[132] = (0,1) (P1:='B';CCAR:=CCAR+1);
    .
    .
LEX[156] = (0,1) (P1:='Z';CCAR:=CCAR+1);
LEX[157] = (0,1) (P1:='A';CCAR:=CCAR+1);
    .
    .
LEX[182] = (0,1) (P1:='Z';CCAR:=CCAR+1);
LEX[183] = (0,0) (CCAR:=CCAR+1);
    .
    .
LEX[208] = (0,0) (CCAR:=CCAR+1);
```

3.1.2. Chiffres :

```
LEX[209] = (0,1) (P1:='0';CCAR:=CCAR+1);
    .
    .
LEX[218] = (0,1) (P1:='9';CCAR:=CCAR+1);
LEX[219] = (0,1) (P1:='A';CCAR:=CCAR+1);
    .
    .
LEX[224] = (0,1) (P1:='F';CCAR:=CCAR+1);
LEX[225] = (0,0) (CCAR:=CCAR+1);
    .
    .
LEX[234] = (0,0) (CCAR:=CCAR+1);
LEX[235] = (0,1) (P1:='0';CCAR:=CCAR+1);
    .
    .
LEX[244] = (0,1) (P1:='9';CCAR:=CCAR+1);
```

3.1.3. Symboles :

```
LEX[245] = (0,0) (CCAR:=CCAR+1);
LEX[246] = (0,0) (CCAR:=CCAR+5);
LEX[247] = (0,0) (CCAR:=CCAR+2);
LEX[248] = (0,0) (CCAR:=CCAR+8);
LEX[249] = (0,0) (CCAR:=CCAR+6);
```


On a besoin à l'étude lexicographique de connaître certains caractères du texte source par exemple les lettres (<llex>) et les chiffres (<chlex>) formant les identificateurs des déclarations globales (<idlex>) ou les caractères composant les notations d'éléments. Il faut alors conserver le caractère en question.

Dans tous les cas, on incrémente le compteur CCAR du texte source afin de localiser correctement les erreurs décelées lors de ce passage.

3.1.4. Caractères :

```
LEX[250] = (0,1) (P1:='''';CCAR:=CCAR+2);
LEX[251] = (0,1) (P1:='';CCAR:=CCAR+1);
LEX[252] = (0,1) (P1:='le caractère';CCAR:=CCAR+1);
      .
      .
LEX[273] = (0,1) (P1:='le caractère';CCAR:=CCAR+1);
LEX[118] = (0,1) (P1:='''';CCAR:=CCAR+2);
LEX[274] = (0,1) (P1:='';CCAR:=CCAR+1);
LEX[275] = (0,1) (P1:='le caractère';CCAR:=CCAR+1);
      .
      .
LEX[296] = (0,1) (P1:='le caractère';CCAR:=CCAR+1);
LEX[128] = (0,0 (CCAR:=CCAR+2);
LEX[297] = (0,0) (CCAR:=CCAR+1);
      .
      .
LEX[319] = (0,0) (CCAR:=CCAR+1);
```

3.1.5. Changement de ligne :

```
LEX[39] = (0,0) (CL:=CL+1;CCAR:=1);
```

3.1.6. Chaînes :

```
LEX[2] = (1,4) (P4:=1;P1:=COMPACT(P1;0;0;0);P2:=0;P3:=0);
LEX[3] = (5,4) (P4:=P4+1;P4> 11:ERREUR(1),
                          P4≤ 4:OCTET(P4;P1;P5),
                          P4≤ 8:(I:=P4-4;OCTET(I;P2;P5)),
                                (I:=P4-8;OCTET(I;P3;P5)));
LEX[4] = (0,0) (CCAR:=CCAR+2);
LEX[5] = (1,2) (P2:=1;P1:=COMPACT(0;0;0;P1));
LEX[6] = (3,2) (P2:=P2+1;P2>4:ERREUR(1),
                          (I:=5-P2;OCTET(I;P1;P3)));
LEX[7] = (2,1) (CCAR:=CCAR+2);
```


L'espace représenté en FACE par le caractère ␣ (1.4.1.2) et servant à former les chaînes notant des éléments ou des terminaux (< carlex> ou <caractère>) est utile sous forme du caractère espace Face 0.

L39

La procédure ANALYSE place le numéro de règle 39 dans la ramification résultat (§1.4.1.2.) à chaque changement de ligne du texte source. Aussi on incrémente CL et on repositionne CCAR à 1.

CHAINES

Les <chaînes fermées> sont des notations de terminaux. Dans l'implémentation considérée ils ne peuvent avoir plus de 11 caractères.

On conserve ces caractères dans les octets successifs de 3 mots. L'octet suivant le dernier caractère est à zéro (§2.2.1.). On conserve le nombre de caractères du terminal.

Les <chaînes fermées> sont des notations d'éléments. Dans l'implémentation considérée, ils ont au plus 4 caractères.

On conserve les caractères, de droite à gauche, dans les octets d'un mot. Ils sont complétés par des zéros à gauche.

3.1.7. Entiers :

```
LEX[ 8] = (1, 4) (P4:=1;OCTET(P4;P1;P1));
LEX[ 9] = (5, 4) (P4:=P4+1;P4 > 10:ERREUR(1),
                           P4≤ 4:OCTET(P4;P1;P5),
                           P4 ≤ 8:(I:=P4-4;OCTET(I;P2;P5)),
                                  (I:=P4-8;OCTET(I;P3;P5)));
LEX[33] = LEX[ 8];
LEX[34] = LEX[ 9];
LEX[124] = (1, 5) (P5:=1;P1:=COMPACT('O';'''';P1;0));
LEX[125] = (6, 5) (P5:=P5+1;P5 > 11:ERREUR(1),
                            P5=2:OCTET(4;P1;P6),
                            P5 ≤ 6:(I:=P5-2;OCTET(I;P2;P6)),
                            P5 ≤ 10:(I:=P5-6;OCTET(I;P3;P6)),
                                    OCTET(1;P4;P6));
LEX[122] = (1, 4) (P4:=1;P1:=COMPACT('X';'''';P1;0));
LEX[123] = (5, 4) (P4:=P4+1;P4 > 9:ERREUR(1),
                            P4=2:OCTET(4;P1;P5),
                            P4 ≤ 6:(I:=P4-2;OCTET(I;P2;P5));
                                   (I:=P4-6;OCTET(I;P3;P5)));
LEX[ 12] = (4, 0) (INF3[ 1]:=P1;INF3[ 2]:=P2;INF3[ 3]:=P3;
                  GLOB('DATA';INF3;P4);CQG:=CQG+1);
LEX[126] = (5, 0) (P5=1:OCTET(4;P1;''''),
                   P5 < 6:(I:=P5-1;OCTET(I;P2;'''')),
                   P5 < 10:(I:=P5-5;OCTET(I;P3;'''')),
                          (I:=P5-9;OCTET(I;P3;'''')));
                   INF4[ 1]:=P1;INF4[ 2]:=P2;INF4[ 3]:=P3;INF4[ 4]:=P4;
                   I:=P5+3;
                   GLOB('DATA';INF4;I);CQG:=CQG+1);
LEX[127] = (4, 0) (P4=1:OCTET(4;P1;''''),
                   P4 < 6:(I:=P4-1:OCTET(I;P2;'''')),
                          (I:=P4-5;OCTET(I;P3;'''')));
                   INF3[ 1]:=P1;INF3[ 2]:=P2;INF3[ 3]:=P3;
                   I:=P4+3;
                   GLOB('DATA';INF3;I);CQG:=CQG+1);
```


Les entiers décimaux, octaux et hexadécimaux sont des notations d'éléments. Dans l'implémentation considérée, ils doivent respectivement avoir au plus 10, 11 et 9 chiffres significatifs. Ils sont conservés dans les octets successifs de 3 ou 4 mots sous la forme des notations des constantes SYMBOL de même type.

On garde aussi leur nombre de chiffres significatifs.

Les <décim> servent à numéroter les règles de grammaire ou à indiquer la taille d'une rangée globale, à l'étude lexicographique, ils sont implémentés comme les entiers décimaux.

L12, 126, 127.

La procédure standard GLOB (annexe 1.2.) génère la pseudo-instruction Symbol créant la constante correspondant à l'entier à l'assemblage dans la zone globale (§2.3.2) et on incrémente CQG (§2.5.).

3.1.8. Eléments :

```
LEX[13] = (1, 0) (ORDRE('DATA';0;'';0;P1;0);
                  CQG:=CQG+1);
LEX[1] = (0, 0) (CCAR:=CCAR+5;
                 ORDRE('DATA';0;'';0;'NIL';0);
                 CQG:=CQG+1);
```

3.1.9. Identificateurs.

```
LEX[14] = (1, 3) (P3:=1;P2:=0;P1:=COMPACT(P1;0;0;0));
LEX[15] = (4, 3) (P3:=P3+1;P3 >8:( ),
                          P3 ≤4:OCTET(P3;P1;P4),
                               (I:=P3-4;OCTET(I;P2;P4)));
LEX[16] = à LEX[14];
LEX[17] = à LEX[15];
```

3.2. DEFINITION DES GRAMMAIRES.:

3.2.1. Terminal, non terminal, en premier membre numéro de règle.:

```
LEX[23] = (6, 2) (INF3[1]:=P3;INF3[2]:=P4;INF3[3]:=P5;
                  INF2[1]:=COMPACT(terminal;CGRAM;P1;P6);
                  TRAV:=P1;
                  ENTREE(TABLE1;INF3;INF2;FILTRE2);
                  I:=P6 <4:1, P6 <8:2, 3;
                  P1:=TRAV+I);
LEX[25] = (4, 1) (INDIC[1]:=P2;INDIC[2]:=P3;
                  P1:=P1+1;
                  INF2[1]:=COMPACT(nonterminal;CGRAM;P1;0);
                  ENTREE(TABLE1;INDIC;INF2;FILTRE1));
LEX[28] = (8, 4) (INF3[1]:=P5;INF3[2]:=P6;INF3[3]:=P7;
                  TRAV:=DEC(INF3;P8);
                  P1:=P1 >TRAV:P1, TRAV);
```

3.2.2. Règle:

```
LEX[26] = (3,3) (CCAR:=CCAR+2;P1:=P1+1);
LEX[130] = (3, 3) (P1:=P1+1);
```


L13

On génère la directive Symbol créant la constante correspondant à la chaîne dans la zone globale. Incrémentation de CQG (§2.3.2. et 2.5.).

L1

Chaque élément de valeur nil est créé à l'assemblage par la directive Symbol appropriée en zone globale. Incrémentation de CQG (§2.5.1. et 2.2.2.).

Identificateurs.

Les identificateurs (<idlex>, <id>) des déclarations globales et les non terminaux des grammaires sont seuls utiles à l'étude lexicographique pour les tables et donc sont construits à ce passage (§2.1.2.).

Dans l'implémentation considérée seuls les 8 premiers caractères sont pris en compte.

On place les caractères constituant l'identificateur dans les octets successifs de 2 mots que l'on conserve, avec le compte de caractères.

L23

On entre en TABLE1 (§2.1.1. et 2.4.1.) la chaîne formée du terminal s'il n'y est pas déjà (procédure FILTRE2) avec comme information le numéro de la grammaire et son numéro d'identification (P1). Le pointeur en table des terminaux (P1) est incrémenté de la place prise par le terminal.

L25

On entre le non terminal en TABLE1 avec comme information le numéro de la grammaire et son rang dans la table des non terminaux (P1) (§2.4.1.).

L28

On retient le numéro de règle le plus élevé de la grammaire (§2.4.1.).

L26, 130

On compte le nombre des caractères non significatifs du texte source indiqué dans la règle de 1er membre vide (§2.2.1.).

3.2.3. Grammaire:

```
LEX[22] = (0,4) (CGRAM:=CGRAM+1;P1:=0;P2:=0;P3:=0;P4:=0;CCAR:=CC
LEX[31] = (4,0) (TRAV:=CGRAM+CGRAM+CGRAM;
                 TGRAM[TRAV]:=P3;
                 I:=TRAV-1;TGRAM[I]:=P4;
                 I:=TRAV+1;TGRAM[I]:=P2;
                 TRAV:=CGRAM+CGRAM;
                 TG[TRAV]:=P1;
                 I:=TRAV+1;TG[I]:=CGRAM);
LEX[32] = (0,0) (TG[1]:=CGRAM+CGRAM+1;
                 TGRAM[1]:=CGRAM;
                 CGRAM=0:( ),
                          (ETI:='TG';
                           ORDRE('EQU';0;'';0;'$';0);
                           ECRIRANG(TG;TG[1]);
                 ETI:='CONS';ORDRE('EQU';0;'';0;'$';0) );
```

3.3. DECLARATIONS GLOBALES:

3.3.1. Déclarations d'éléments, de repères:

```
LEX[35] = (3,0) (INDIC[1]:=P1;INDIC[2]:=P2;
                 INF2[1]:=COMPACT(0;élément;'';'4');
                 INF2[2]:=CQG-1;
                 ENTREE(TABLE;INDIC;INF2;FILTRE) );
LEX[36] = (3,0) (INDIC[1]:=P1;INDIC[2]:=P2;
                 INF2[1]:=COMPACT(0;repère;'';'4');
                 INF2[2]:=CQG;
                 ENTREE(TABLE;INDIC;INF2;FILTRE);
                 ORDRE('RES';0;'';0;'1';0) );
LEX[37] = (3,0) (INDIC[1]:=P1;INDIC[2]:=P2;
                 INF2[1]:=COMPACT(0;repère;'';'4');
                 INF2[2]:=CQG-1;
                 ENTREE(TABLE;INDIC;INF2;FILTRE) );
```


L22 au début d'une grammaire :
incrémentation de CGRAM. On positionne à zéro :

- un compteur retenant le numéro de règle le plus élevé
- un compteur des caractères non significatifs du texte source
- un pointeur dans la table des terminaux
- un pointeur dans la table des non terminaux

L31 à la fin d'une grammaire :
On conserve dans TG et TGRAM (§2.5. et 2.4.1.) les renseignements trouvés à savoir la taille des tables des terminaux et non terminaux et le nombre de caractères non significatifs dans TGRAM, le numéro de règle le plus élevé et la clé dans TG.

L32 à la fin des grammaires :
on conserve le nombre des grammaires dans TGRAM[1]. On connait donc la taille de la zone grammaire que l'on place en TG[1] (§2.3.1. et 2.4.1.). On génère l'étiquette TG et les directives Symbol créant la zone grammaire à l'assemblage (§2.3.1.), toutes les constantes la composant étant dans la rangée TG (procédure standard ECRIRANG).

On génère l'étiquette CONS qui marque le début de la zone globale (§2.3.2.).
Remarque :La zone grammaire ne peut être générée qu'une fois la taille connue (car sa taille doit la précéder (§2.3.1.)). C'est pourquoi on stocke au fur et à mesure son image dans la rangée TG.

L35 élément déclaré :
on entre en table l'identificateur désignant l'élément (§2.4.2).

L36 déclaration de repère non initialisé :
on entre l'identificateur en table (§2.4.2.) et on génère la directive Symbol réservant une place en zone globale (§2.3.2.).Incrémentation de CQG.

L37 déclaration de repère initialisé :
on entre en table l'identificateur désignant le repère (§2.4.2.).

3.3.2. Déclaration globale de rangée d'éléments ou de repères.

```
LEX[40] = (4,2) (INF3[1]:=P1;INF3[2]:=P2;INF3[3]:=P3;
                 P1:=P4;P2:=DEC(INF3;P4) ) ;
LEX[39] = (2,2) (GLOB('DATA';INF3;P1);CQG:=CQG+1);
LEX[41] = (0,1) (P1:=1);
LEX[42] = (1,1) (P1:=P1+1);
LEX[38] = (6,2) (INDIC[1]:=P3;INDIC[2]:=P4;
                 INF2[1]:=COMPACT(0;rangée d'éléments;'';'4');
                 INF2[2]:=CQG-P6-1;
                 P6≠P2:ERREUR(4),
                       ENTREE(TABLE;INDIC;INF2;FILTRE) );
LEX[44] identique à LEX[38] avec comme type rangée de repères.
LEX[43] = (5,2) (INDIC[1]:=P3;INDIC[2]:=P4;
                 INF2[1]:=COMPACT(0;rangée de repères;'';'4');
                 INF2[2]:=CQG-1;
                 ENTREE(TABLE;INDIC;INF2;FILTRE);
                 GLOB('RES';INF3;P1) );
```

3.3.3. Déclarations de procédures, de procédures écrites en code :

```
LEX[45] = (0,2) (P1:=procédure élément;P2:='');
LEX[46] = (0,2) (P1:=procédure repère;P2:='*');
LEX[47] = (3,6) (P3:=P1;P4:=P2;
                 P1:=procédure sans résultat;P2:='';
                 P5:=0;P6:=0;NO:=NO+1);
LEX[48] = (1,2) (P1:=0;P2:=0:NO:=NO+1);
LEX[49] = (6,0) (INDIC[1]:=P3;INDIC[2]:=P4;
                 INF4[1]:=COMPACT(0;P1;P2;0);
                 INF4[2]:=P6;INF4[3]:=CREETI( );
                 INF4[4]:=0-P5-1;
                 ENTREE(TABLE;INDIC;INF4;FILTRE) );
LEX[51] = (0,0) (INF3[1]:=COMPACT(NO;élément;'';formel) );
LEX[52] = (0,0) (INF3[1]:=COMPACT(NO;repère;'*';formel) );
LEX[53] identiques à LEX[52] avec rangée d'éléments
LEX[54]                          et rangée de repères.
```


L40 : taille d'une rangée : on conserve la valeur de cette taille.

L39 : début d'une rangée :

On génère la directive SYMBOL plaçant en zone globale (§2.2.2. et 2.3.2.) la taille de la rangée. Incrémentation de CQG.

L41, L42 : On compte le nombre d'éléments de la liste définissant ou initialisant la rangée.

L38, 44 : rangée d'éléments ou de repères initialisés:

On entre l'identificateur en Table (§2.4.2.) et on regarde si le nombre d'éléments de la liste est conforme à la taille déclarée (§2.4.6.).

L43 : rangée de repères non initialisée :

On entre l'identificateur en table (§2.4.2.). On génère la directive Symbol réservant en zone globale la place de la rangée (§2.3.2. et 2.2.2.).

L45, 46 : On conserve le type du résultat avec la marque de chaînage qui lui est associée.

L47, 48 : identificateur de procédure :

On le conserve. On positionne à zéro :

- un compteur du nombre des paramètres formels
- un pointeur vers les renseignements en TABLE du paramètre formel qui suit (on commence par examiner le dernier paramètre) (§2.4.2.et 2.4.3.).

On incrémente NO car on commence un bloc où sont valides les paramètres formels.

L49 : fin de la procédure:

On entre en TABLE (§II.4.2.) le nom de la procédure (§2.4.2. et 2.4.3.).

L51 à 54 : on prépare la première zone de l'information du paramètre formel. (§2.4.2.).

```
LEX[50] = (5, 2) (P1:=P1+1;
                  INDIC[1]:=P3;INDIC[2]:=P4;
                  INF3[2]:=P2;INF3[3]:=0-P1-1;
                  (.REP POINT=ENTREE2(TABLE;INDIC;INF3;FILTRE);
                  P2:=REPELT(POINT) ) );
LEX[55] = (0, 2) (P1:=procédure rangée de repères;P2:='*');
LEX[56] = (3, 5) (P3:=P1;P4:=P2;
                  P1:=procédure sans résultat;P2:='';
                  P5:=0;NO:=NO+1);
LEX[57] = (1, 1) (P1:=0;NO:=NO+1);
LEX[58] = (5, 0) P5 > 4:ERREUR(1),
                        (INDIC[1]:=P3;INDIC[2]:=P4;
                         INF3[1]:=COMPACT(0;P1;P2;0);
                         INF3[2]:=P5;INF3[3]:=P3;
                         ENTREE(TABLE;INDIC;INF3;FILTRE);
                         ORDRE('REF';0;'';0;P3;0) );
LEX[59] = (4, 1) (INDIC[1]:=P1;INDIC[2]:=P2;
                  INF2[2]:=P1;
                        (.REP POINT=ENTREE2(TABLE;INDIC;INF2;FILTR
                         P1:=REPELT(POINT) ) );

LEX[60]
   }= (0, 0) (INF2[1]:=COMPACT(NO;marque de type;0;0) );
LEX[64]                                  appropriée
```


L50. Paramètre formel :

Incrémenter le compteur du nombre de paramètres.

Entrée en table d'identificateur du nom du paramètre (§2.4.2.). On retient l'adresse des renseignements de ce paramètre en TABLE pour le chaînage des renseignements des paramètres (§2.4.3.).

L55. Résultat rangée pour une procédure écrite en code.

L56.57. Identificateur de procédure écrite en code:

Comme pour L47,48, on conserve le nom de la procédure. On incrémente NO pour isoler en table les paramètres fictifs permettant de décrire la procédure. Par contre, on ne les compte pas car on n'a pas besoin de l'adresse des paramètres et du résultat interne au segment.

L58. A la fin de la déclaration de la procédure écrite en code :

On entre en table le nom de la procédure.

On génère la directive Symbol définissant ce nom comme une référence externe pour l'éditeur de liens.

Dans cette implémentation le nom d'une telle procédure ne peut avoir plus de 4 caractères.

L59. Les indicatifs des paramètres fictifs servent à entrer en table leur type et à chaîner les renseignements comme pour une procédure normale.

L60 à 64. On prépare la 1ère zone de l'information du paramètre fictif.

3.3.4. Déclaration de rangée de procédures:

```
LEX[65] = (5,2) (INDIC[1]:=P3;INDIC[2]:=P4;
                 TRAV:=CREETI(1;Z[CG]:=TRAV;CG:=CG+1;
                 Z[CG]:=P2;
                 INF2[1]:=COMPACT(0;rangée de procédures;'';'1');
                 INF2[2]:=CG;CG:=CG+1;
                 ENTREE(TABLE;INDIC;INF2;FILTRE);
                 MRAPPL(P2;TRAV);
                 P1:=P3;P2:=P4 );
LEX[66] = (5,2) P1≠ P3:ERREUR(7),
                P2≠ P4:ERREUR(7) , ( );
LEX[71] = (2,0) ( );
```

3.3.5. Liste de déclarations:

```
LEX[121] = (2,0) ( );
```

3.4. DEBUT D'UN BLOC :

```
LEX[113] = (0,0) (NO:=NO+1);
```

3.5. PROGRAMME :

```
LEX[120] = (0,0) (NO:=0;CQG:=0;CGRAM:=0;CG:=1;
                  CCAR:=0;CL:=0);
```


L65. Déclaration d'une rangée de procédure:

On place dans Z (§2.5.) la taille de la rangée (§2.3.) et création de l'étiquette de la procédure vide pour la rangée (§2.2.3.).

On entre en table le nom de la rangée (§2.4.2.).

On positionne dans Z l'étiquette ci-dessus dans un nombre de repères égal à la taille de la rangée (procédure MRAPPL).

On conserve le nom de la rangée.

L66. On vérifie que le nom d'une procédure d'une rangée est identique au nom de la rangée.

L71. On ne conserve plus le nom de la rangée de procédures.

L121. On ne conserve plus les renseignements concernant la taille de la rangée.

L113. Les paramètres formels doivent être entrés sous leur bon numéro de bloc. (§2.4.1.).

L120. Fin de l'étude lexicographique :

On repositionne à leur valeur initiale les repères globaux Face 0 servant aussi à la génération (§2.5.).

CHAPITRE 4

LA GENERATION DU PROGRAMME OBJET

Rangée de procédures GEN

4.1. SYMBOLES DE BASE :

4.1.1. Lettres.

GEN[131] = (0, 1) ⎫ ⎧ LEX[131]
⋮ ⎬ = ⎨ ⋮
GEN[156] = (0, 1) ⎭ ⎩ LEX[156]

GEN[157] = (0, 0) (CCAR:=CCAR+1);
⋮
GEN[182] = (0, 0) (CCAR:=CCAR+1);
GEN[183] = (0, 1) (CCAR:=CCAR+1;P1:='A');
⋮
GEN[208] = (0, 1) (CCAR:=CCAR+1;P1:='Z');

4.1.2. Chiffres :

GEN[209] = (0, 0) (CCAR:=CCAR+1);
⋮
GEN[224] = (0, 0) (CCAR:=CCAR+1);
GEN[225] = (0, 1) (P1:='0';CCAR:=CCAR+1);
⋮
GEN[234] = (0, 1) (P1:='9';CCAR:=CCAR+1);
GEN[235] = (0, 1) (P1:='0';CCAR:=CCAR+1); LEX[235]
⋮ identiques à ⋮
GEN[244] = (0, 1) (P1:='9';CCAR:=CCAR+1); LEX[244]

4.1.3. Symboles :

GEN[245] ⎫ ⎧ LEX[245]
⋮ ⎬ = ⎨ ⋮
GEN[249] ⎭ ⎩ LEX[249]

Certains caractères du texte source doivent être connus à la génération, par exemple les lettres (<lgen >), les chiffres (<chgen>), les caractères (<cargen>) formant :

- les identificateurs des déclarations locales.
- Les identificateurs des primaires d'expression,
- des non terminaux
- les chaînes notant les terminaux.

Dans tous les cas, on incrémente CCAR (compteur de caractères du texte source) afin de localiser correctement les erreurs décelées lors de ce passage.

4.1.4. Caractères :

```
GEN[250] = LEX[250];
   ⋮
GEN[273] = LEX[273];
GEN[118] = (0,0) (CCAR:=CCAR+2);
GEN[274] = (0,0) (CCAR:=CCAR+1);
   ⋮
GEN[296] = (0,0) (CCAR:=CCAR+1);
GEN[128] = (0,1) (P1:='''';CCAR:=CCAR+2);
GEN[297] = (0,1) (P1:='' ;CCAR:=CCAR+1);
GEN[298] = (0,1) (P1:='le caractère';CCAR:=CCAR+1);
   ⋮
GEN[319] = (0,1) (P1:='le caractère';CCAR:=CCAR+1);
```

4.1.5. Changement de ligne :

```
GEN[39] = LEX[39]
```

4.1.6. Chaînes :

```
GEN[2] = LEX[2];
GEN[3] = LEX[3];
GEN[4] = LEX[4];
GEN[7] = (0,0) (CCAR:=CCAR+2);
```

4.1.7. Entiers, éléments:

```
GEN[33] = LEX[33];
GEN[34] = LEX[34];
GEN[10] = LEX[33];
GEN[11] = LEX[34];
GEN[12] = (0,0) (CQG:=CQG+1);
GEN[126] = (0,0) (CQG:=CQG+1);
GEN[127] = (0,0) (CQG:=CQG+1);
GEN[13] = (0,0) (CQG:=CQG+1);
GEN[1] = (0,0) (CQG:=CQG+1);
```


G2.3.4.

Les notations des terminaux (< chaîne fermée >) sont réutilisées à la génération comme à l'étude lexicographique.

G7.

Les notations d'éléments (< chaîne lex fermée >) ne sont pas reconstruites à la génération.

G33, 34.

Les < décim >, entiers décimaux (numérotant les règles de grammaire ou donnant les tailles des rangées globales) sont utilisées à la génération comme à l'étude lexicographique.

G10, 11.

Les < décimal > :

- Entiers décimaux donnant les tailles des rangées locales.
- Entiers i des #i réservés (§1.4.1.3.).
- En enttête des procédures d'une rangée, ne sont utilisés qu'à la génération :

On les construit comme les autres entiers ou éléments.

G12, 126;127, 13, 1.

Pour les diverses notations d'éléments, chaînes ou entiers de toute base, on incrémente le pointeur de zone globale CQG (§2.3.2. et 2.5.).

4.1.8. Identificateurs.

```
GEN[14] = LEX[14];
GEN[15] = LEX[15];
GEN[18] = LEX[18];
GEN[19] = LEX[19];
GEN[20] = (4,2) (INF3[1]:=P1;INF3[2]:=P2:INF3[3]:=P3;
                 P1:=identificateur réservé;P2:=DEC(INF3;P4));
GEN[21] = (1,0) ( );
```

4.2. DEFINITION DES GRAMMAIRES :

3.2.1. Terminal, non terminal, numéro de règle, premier membre :

```
GEN[23] = (4,1) (INF3[1]:=P1;INF3[2]:=P2;INF3[3]:=P3;
                 ( RANGEE. .REP PINF=RECHERCHE(TABLE1;
                                INF3;B2;ERREUR);
                  P1:=DECOMP(3;PINF[1]);
                  TRAV:=G[5]+P1 ;G[TRAV]:=INF3[1];
                  P4 > 3:(I:=TRAV+1;G[I]:=INF3[2];
                        P4 > 7:(I:=TRAV+2;G[I]:=INF3[3]),
                               ( ) ),
                        ( )    ) );
GEN[24] = (3,1) (INDIC[1]:=P1;INDIC[2]:=P2;
                 (RANGEE REP PINF=RECHERCH(TABLE1;INDIC;
                                B1;ERREUR);
                  P1:=0-DECOMP(3;PINF[1]) ) );
GEN[28] = (4,1) (INF3[1]:=P1;INF3[2]:=P2;INF3[3]:=P3;
                 P1:=DEC(INF3;P4) );
GEN[29] = (0,1) (P1:=0-1);
GEN[25] = (6,3) (INDIC[1]:=P4;INDIC[2]:=P5;
                 (RANGEE REP PINF=RECHERCH(TABLE;INDIC;
                                B1;ERREUR);
                  TRAV:=DECOMP(3;PINF[1])+G[11]-1;
                  G[TRAV]:=P1;
                  P2:=P1;P3:=0 ) );
```


G14, 15.

Les identificateurs, des non terminaux des grammaires par exemple, (<id>) sont utilisés à LEX et à GEN et sont donc construits de la même façon aux 2 passages.

G18, 19.

Les identificateurs des déclarations locales ou des primaires d'expression (<idgen> , <ident>) sont construits uniquement à la génération.

G20.

Pour les identificateurs réservés #i (<idres>), on conserve la valeur i précédée de la marque réservée.

G21.

Par homogénéité avec les <idres> on ne conserve pas plus longtemps le compte des caractères de l'identificateur (de type <ident>) servant aux primaires d'expression.

G23. Terminal dans un 2° membre de règle :

Les renseignements sont recherchés en TABLE1 (§2.4.1). On conserve le numéro d'identification (P1). On place le terminal dans la table des terminaux (§2.2.1.) (même s'il y est déjà). Cette table est située dans la rangée G (§2.5 et 2.4.1.).

G24. Non terminal dans un 2° membre de règle:

Les renseignements sont recherchés en TABLE1 (§2.4.1.). On conserve le numéro d'identification (§2.2.1.).

G28, 29. Numéro de règle:

On conserve la valeur du numéro de règle ou à défaut -1 (§2.2.1.).

G25. Non terminal en 1er membre de règle : (§2.2.1.).

Les renseignements sont recherchés en TABLE1 (§2.4.1.). On place au rang convenable dans la table des non terminaux (située dans G) le pointeur vers le début de la ramification correspondant à la règle (P1 pointeur dans la ramification de la grammaire dans G). C'est un début de construction de la ramification de cette règle aussi on positionne

- le pointeur (P2) vers le début de la ramification de la règle
- le pointeur (P3) à zéro afin de savoir que l'on doit copier directement les renseignements des noeuds de la 1ère branche. Ce pointeur servira ensuite à pointer vers le bloc du noeud à comparer (G27, G30).

4.2.2. Règle :

```
GEN[26] = (6,4) (I:=12+P1;G[I]:=P6;
                 DEMIMOT(1;G I];P5);
                 P1:=P1+1);
GEN[130] = (5,4) (I:=12+P1;G I]:=P5;
                 P1:=P1+1);
GEN[27] = (4,3) P3=0:(I:=P1+3;GRG(P1;P4;I;0)),
                 DEBIT(2;G[P3]):REPOUS (P1;P3;P4),
                       COMPAR(P1;P3;P4);
GEN[30] = (4,3) (G[P1]:=P4;
                P4 >0 :BIT(2;G[P1];1),
                      DEMIMOT(1;G[P1];16385);
                P1:=P1+1;G[P1]:=0;
                P1:=P1+1;G[P1]:=CL;
                P1:=P1+1;P3:=P2 );
```

4.2.3. Grammaire :

```
GEN[22] = (0,4) (CGRAM:=CGRAM+1;
                 CCAR:=CCAR+6;P1:=0;
                 TRAV:=CGRAM+CGRAM+CGRAM;
                 I:=TRAV+1;G[11]:=TGRAM[I];
                 G[6]:=TGRAM[TRAV];
                 G[5]:=12+G[11];
                 G[1]:=G[5]+G[6];
                 I:=TRAV-1;G[2]:=TGRAM[I];
                 G[3]:=G[1]+G[2];
                 P2:=G[3]          );
GEN[31] = (4,0) (G[4]:=P2-G[3];
                 FADA(G;CGRAM)    );
```


G26, 130. Règle donnant les caractères non significatifs:

On place le caractère (0 pour $ §1.4.3.) à sa place dans le vecteur de renseignements (situé dans G) ainsi que le numéro de règle (respectivement 1er et 2ème demi-mot).

Incrémentation du compte des caractères non significatifs.

G27. Terminal ou non terminal en 2^{è} membre (§2.2.1.).

On construit le bloc de 3 mots correspondant dans la ramification : si le pointeur de comparaison est à zéro, on construit directement le bloc (procédure GRG) au niveau du pointeur dans la ramification. (C'est le cas de la 1ère branche au départ ou d'un membre commençant à être différente de celle qui le précède et qui crée une nouvelle branche).

Sinon, on regarde le numéro d'identification et celui du noeud situé au rangdu pointeur de comparaison. Si c'est celui d'une fin de règle, on appelle la procédure repous sinon la procédure compar.

G30. fin de règle :

On construit dans la ramification le bloc de 3 mots concernant une fin de règle (§2.2.1.).

Incrémentation du pointeur dans la ramification. Ensuite commence un autre membre donc une autre branche : on positionne le pointeur de comparaison vers le noeud du départ de la règle concernée.

G22. Début d'une grammaire :

On met à jour CGRAM. On remplit le vecteur de renseignements (§2.2.1.) dans G grâce aux renseignements recueillis à l'étude lexicographique et conservés dans TGRAM.

On met à zéro un compte des caractères non significatifs. On positionne un pointeur dans G au début de la ramification.

On met en place 3 cellules supplémentaires par homogénéité avec le 1er membre de règle.

G31. Fin d'une grammaire :

On supprime les 4 cellules mises en place au début (G22) après avoir placé la taille de la ramification dans le vecteur de renseignements. On appelle le préanalyseur FADA pour cette grammaire (§2.4.1.).

4.3. LES DECLARATIONS GLOBALES.

4.3.1. Déclarations d'éléments, de repères.

```
GEN[36] = (0,0) (CQG:=CQG+1);
```

4.3.2. Déclarations de rangées d'éléments, de repères.

```
GEN[40] = (4,1) (INF3[1]:=P1;INF3[2]:=P2;INF3[3]:=P3;
                 P1:=DEC(INF3;P4) );
GEN[39] = (0,0) (CQG:=CQG+1);
GEN[43] = (1,1) (CQG:=CQG+P1);
```

4.3.3. Déclarations de procédures.

```
GEN[47] = (3,2) (INDIC[1]:=P1;INDIC[2]:=P2;
                 (RANGEE REP PINF=RECHERCH(TABLE;INDIC;B;ERREUR);
                 NB:=NB+1;NO:=NB;COURS[NO]:=1;
                 P1:=DECOMP(2;PINF[1]);P2:=PINF[4];
                 ETI:=PINF[3]ORDRE('LCI';0;'';0;'0';0);
                 ORDRE('STM';'0';'';0;'1';'5');
                 ML:=17) );
GEN[48] = GEN[47];
GEN[49] = (7,0) (ORDRE('LCI';0;'';0;'0';0);
                 ORDRE('LM';'0';'';0;'1';'5');
                 P1 = procédure sans résultat :
                     (P3=procédure sans résultat:( ), ERREUR(5)),
                     (P1=procédure élément:
                         (P3=élément:( ),
                          P3=repère:( ),
                          P3=repère d'expression:( ), ERREUR (5);
                        ML:=P2;TLACER(P4;P5;P6;P7) ),
                        P3=repère:( ),
                             P3=repère d'expression:( ), ERREUR(5);
                        ML:=P2;TOINTER(P4;P5;P6) );
                  ORDRE('B';0;'*';0;'5';0);
                  COURS[NO]:=0);
```


Le compteur CQG étant incrémenté au niveau élément, il n'y a rien à faire pour un élément ou un repère initialisé.

G36: pour un repère non initialisé : Incrémentation de CQG.

G40 : début d'une rangée :
on conserve la taille de la rangée.

Le compteur CQG est incrémenté au niveau des éléments composant ou initialisant la rangée sauf pour une rangée de repères non initialisée .

G47, 48 : une procédure déclarée : Recherche en TABLE de ce nom.

Incrémentation de NB pour le bloc des paramètres formels, positionnement du numéro de bloc NO et du booléen correspondant dans COURS. On génère l'étiquette de début du corps , on conserve le type du résultat et sonadresse dans le segment de la procédure.

On positionne ML pour allouer correctement les objets internes au corps de la procédure après avoir généré le rangement des registres (§2.4.3.).

G49 : fin de procédure :

On génère la restauration des registres (§2.4.3.).

On vérifie que le type du résultat du corps de procédure est conforme à celui qui est déclaré.

En particulier si le corps est un repère et le résultat élément, c'est un repère utilisé dans un contexte d'élément (§1.4.7.5.).

On génère le placement du résultat à l'endroit prévu dans le segment de la procédure (§2.4.3.).

On génère le branchement à l'adresse retour.

On déclare valide le bloc créé pour les paramètres formels.

Remarque : Pour les procédures écrites en code, il n'y a rien à faire à ce passage : il n'y a rien à générer pour ces procédures.

4.3.4. Déclaration de rangées de procédures :

```
GEN[65] = (1, 2) (ETI:=CREETI( )ORDRE('LW';'3';'';1;CG;'1');
                  ORDRE('B';0;'*';0;'3';0);
                  CG:=CG+1;P2:=CG;CG:=CG+P1+1);
GEN[66] = (6, 3) (INF3[1]:=P3;INF3[2]:=P4;INF3[3]:=P5;
                  P3:=DEC(INF3;P6);
                  P3 > P1:ERREUR(8), ( ) );
GEN[67] = (11, 3) (INF3[1]:=P4;INF3[2]:=P5;INF3[3]:=P6;
                   TRAV:=DEC(INF3;P7);
                   INF3[1]:=P8;INF3[2]:=P9;INF3[3]:=P10;
                   P3:=DEC(INF3;P11);
                   ETI:=CREETI( );Z[P1+P2]:=ETI;
                   RP:=0
                   ML:=TRAV> P3:TRAV, P3;ML:=ML+1;
                   ORDRE('AI';'5';'';1;0-TRAV;0) );
GEN[68] = (11, 3) (INF3[1]:=P4;INF3[2]:=P5;INF3[3]:=P6.
                   TRAV:=DEC(INF3;P7);
                   INF3[1]:=P8;INF3[2]:=P9;INF3[3]:=P10;
                   P3:=DEC(INF3;P11);
                   ETI:=CREETI( );Z[P2+P3]:=ETI;
                   RP:=TRAV;
                   ML:=TRAV > P3:TRAV, P3;ML:=ML+1;
                   RECHNIL(TRAV;P3) );
GEN[70] = (8, 2) P4 = sans résultat :
                      (ORDRE('AI';'5';'';1;P3;0);
                       RP=0:( ), ORDRE('LW';'5';'*';0;'5';0);
                       I:=P2-1.;
                       ORDRE('B';0;'';0;Z[I];0) );
                       ERREUR(20);
```


G65. Début de la rangée de procédure :

On génère l'étiquette de début de la procédure vide (§2.2.3.):

On génère uniquement un branchement à l'adresse retour (§2.4.4.).

On incrémente CG de la longueur de la table de la rangée de procédures et on conserve l'adresse de début.

G66. Début d'une procédure d'une rangée :

On vérifie que son rang est inférieur à la taille de la rangée.

G67. En tête de la forme (k, ℓ) (§2.4.4.).

On cherche les valeurs de k et ℓ

On crée et l'on génère une étiquette de début du corps que l'on place dans la table des rangées de procédures dans Z.

On positionne le drapeau RP à zéro (§2.5. et 2.4.4.).

ML pointe vers le 1er résultat interne du segment situé au-dessus des #i, ML est donc égal à 1+sup(k, ℓ).

On génère le déplacement de l'index local sous les # i et on conserve la valeur ℓ.

G68. En tête de la forme (k, rangée, ℓ) (§2.4.4.).

On procède comme ci-dessus en positionnant cette fois RP à k.

On génère par RECHNIL les ordres calculant la longueur de la rangée et recherchent la position de la cellule NIL, établissant sup(k, ℓ) cellules fictives pointant sur les #i et laissant l'index local sous les cellules fictives.

G70. Fin du corps de procédure d'une rangée :

On vérifie que l'expression du corps de procédure est sans résultat.

On génère les ordres amenant l'index local sous les # i résultats.

On génère le branchement à l'adresse retour (§2.4.4.).

```
GEN[69] = (10,2) (INDIC[1]:=P4;INDIC[2]:=P5;
                  (RANGEE REP PINF=RECHERCH(TABLE;INDIC;B;ERREUR);
                  INF3[1]:=P7;INF3[2]:=P8;INF3[3]:=P9;
                  TRAV:=DEC(INF3;P10);
                  I:=P2+P3;J:=PINF[2]+TRAV;
                  Z[I]:=Z[J] ) );
GEN[71] = (2,0)( );
```

4.3.3. Liste de déclarations.

```
GEN[121] = (1,0) ( );
GEN[119] = (0,0) (ML:=0;NO:=0;
                  ETI:='DEB';ORDRE('M:GP';0;'0';0;'256';0);
                  I:=Z-1;
                  ORDRE('LI';'1';'';0;I;0);
                  ORDRE('LI';'4';'';0;'CONS';0);
                  ORDRE('LW';'5';'';0;'9';0) );
```

4.4. LES DECLARATIONS LOCALES.

4.4.1. Début de bloc :

```
GEN[113] = (0,2) (P1:=ML;ML:=ML+1;
                  P2:=NO;NB:=NB+1;NO:=NB;
                  COURS[NO]:=1);
```

4.4.2. Déclarations locales de repères.

```
GEN[103] = (3,0) (INDIC[1]:=P1;INDIC[2]:=P2;
                  INF2[1]:=COMPACT(NO;repère;'';'5');
                  INF2[2]:=ML;ML:=ML+1;
                  ENTREE(TABLE;INDIC;INF2;FILTRE) );
GEN[104] = (8,0) P4= repère:ERREUR(14),
                  (INDIC[1]:=P1;INDIC[2]:=P2;
                   INF2[1]:=COMPACT(NO;repère;P5;P7);
                   INF2[2]:=P6;
                   ENTREE(TABLE;INDIC;INF2;FILTRE) );
```


G69. Cas où la procédure d'une rangée est égale à une procédure d'une rangée déjà déclarée :

On recherche le nom en TABLE de la procédure de la rangée synonyme ce qui permet d'atteindre sa taille dans Z.

Le rang de la procédure synonyme qu'on calcule permet de trouver son étiquette. On la place dans la table de la rangée (située dans Z) au rang de la procédure considérée.

G71. Fin d'une rangée de procédures :

On ne conserve plus l'adresse de la table dans Z et sa taille.

G121. Fin de la déclaration d'une suite de rangée :

On ne conserve plus la taille de la rangée.

G119. Fin des déclarations globales :

On positionne ML à 0 pour l'expression finale ainsi que NO.

On génère les 1ers ordres à exécuter du programme objet: demande du reste de la mémoire pour la zone locale (§2.3.5.) et initialisation des index de début de zone.

G113. Début d'expression fermée avec des déclarations locales : donc d'un nouveau bloc (§2.4.2.). On conserve la valeur de ML et de NO.

On incrémente ML pour allouer le résultat (§2.4.5.).

On incrémente NB . On actualise NO et on positionne le booléen de COURS.

G103. Création d'un repère local (valuation vide).

On incrémente ML et on entre en table le nom du repère (§2.4.2. et 2.3.5.).

G104. Déclaration d'un identificateur de repère (valuation commençant par =) (§2.4.2.).Cet identificateur designe un repère deja créé. Le secondaire le valuant doit avoir pour valeur un repère.

On entre en table l'identificateur local.

```
GEN[105] = (8,0) (P4=repère:( ),
                  P4 = repère d'expression:( ), P4=élément:( ), ERREUR(6);
                 INDIC[1]:=P1;INDIC[2]:=P2;
                 INF2[1]:=COMPACT(NO;repère;'';'5');
                 P8=1:DEPOINT(P5;P6;P7), TLACER(P5;P6;P7;P8);
                 INF2[2]:=ML-1;ENTREE(TABLE;INDIC;INF2;FILTRE) );
```

4.4.3. Déclaration locale de rangées de repères :

```
GEN[107] = (4,1) (INF3[1]:=P1;INF3[2]:=P2;INF3[3]:=P3;
                 P1:=DEC(INF3;P4) );
GEN[108] = (4,1) (ORDRE('LI';'3';'';1;P1;0);
                 ORDRE('STW';'3';'';1;ML;'5');
                 INDIC[1]:=P2;INDIC[2]:=P3;
                 INF2[1]:=COMPACT(NO;rangée de repères;'';'5');
                 INF2[2]:=ML;ML:=ML+P1+1;
                 ENTREE(TABLE;INDIC;INF2;FILTRE) );
GEN[109] = (9,1) (P5=rangée de repères:( ),
                    P5=rangée en pile:( ), ERREUR(14);
                 INDIC[1]:=P1;INDIC[2]:=P2;
                 INF2[1]:=COMPACT(NO;rangée de repères;P6;P8);
                 INF2[2]:=P7;
                 ENTREE(TABLE;INDIC;INF2;FILTRE) );
GEN[106] = (6,1) (P1:=P1+1;
                 P2=élément:( ), P2=repère:( ),
                    P2=repère d'expression:( ), ERREUR(6);
                 P6=1:DEPOINT(P3;P4;P5),
                      TLACER(P5;P6;P7;P8) );
GEN[110] = (2,1) P2=P1:( ), ERREUR(16);
GEN[76] = (1,0) ( );
```


G105. Création d'un repère local (valuation commençant par:=).

L'expression simple valuant doit être de type élément (ou repère dans un contexte d'élément). Cet élément doit être repéré par le repère qu'on crée. On génère les ordres plaçant cet élément en zone locale.

- s'il y est déjà (marque de résultat intermédiaire P8 à 1) on génère les ordres supprimant l'adressage indirect s'il existe (DEPOINT)
- sinon on le place (TLACER).

On met en TABLE l'identificateur désignant le repère créé.

G107. Taille de la rangée :

On calcule et on conserve sa valeur.

G108. Création d'une rangée de repères (valrang vide).

On génère les ordres plaçant la taille en zone locale (§2.2.2.).

On entre en table l'identificateur et on incrémente ML de la place prise par la rangée.

G109. Déclaration d'un identificateur de rangée (valrang commençant par =). L'identificateur désigne une rangée déjà citée (§2.4.2.).

Ce primaire en valrang doit être une rangée de repères. On entre en TABLE l'identificateur.

G106. Expression simple initialisant un repère d'une rangée :

On incrémente le compte du nombre d'expressions (cf G11). Cette expression doit être élément (ou repère dans un contexte d'élément). On génère les ordres plaçant l'élément en zone locale (TLACER) ou, s'il y est déjà, se débarassant de l'adressage indirect (DEPOINT).

G110. Fin de la liste d'expressions initialisant la rangée :

On regarde si leur nombre est égal à la taille de la rangée (§2.4.6.).

G76. Fin d'une déclaration locale de rangée de repères :

On ne conserve plus la taille de la rangée.

```
GEN[111] = (4,2) (ORDRE('LI';'3';'';1;P1;0);
                  ORDRE('STW';'3';'';1;ML;'5');
                  INDIC[1]:=P2;INDIC[2]:=P3;
                  INF2[2]:=ML;ML:=ML+1;
                  ENTREE(TABLE;INDIC;INF2;FILTRE);
                  P2:=0);
```

3.5. EXPRESSIONS.

3.5.1. Appel de procédure.

```
GEN[72] = (2,4) P1=identificateur réservé:ERREUR(11),
                  INDIC[1]:=P1;INDIC[2]:=P2;
                   (RANGEE REP PINF=RECHERCH(TABLE;INDIC;B;ERREU
                   P1:=DECOMP(2;PINF[1]);
                   ML:=ML+1;P2:=ML;
                   P3:=PINF[3];
                   P4:=PINF[2].) );

GEN[73] = (6,1) P1=0:ERREUR(10),
                   (REF POINT=ELTREP(P1);I:=P1+1;P1:=ELTREP(I);
                   TRAV:=DECOMP(2;POINT);
                   TRAV=repère:
                          (P2=repère:( ), P2=repère d'expression:( ),
                           P2=rangée en pile:(RP:='*'), ERREUR(9)),
                         TRAV=élément:
                           (P2=élément:( ), P2=repère:( ),
                            P2=repère d'expression:( ), ERREUR(9)),
                         TRAV=rangée de repères:
                           (P2=rangée de repères:( ), P2=rangée en pile;( ),
                            ERREUR(9)),
                         TRAV=rangée d'éléments:
                           (P2=rangée de repères:( ), P2=rangée en pile:( ),
                            P2=rangée d'éléments:( ), ERREUR(9)),
                         TRAV=P2:( ), ERREUR(9);
                   TRAV=élément:
                        (P6=1:DEPOINT(P3;P4;P5),
                             TLACER(P3;P4;P5;P6)),
```


G111. Création d'une rangée de repères locale (valrang commençant par :=):

On génère les ordres plaçant sa taille en zone locale (§2.2.2.).

On entre en table l'identificateur désignant cette rangée.

On positionne à zéro le nombre d'expressions l'initialisant.

G72. Début d'appel de procédures :

On recherche en TABLE l'identificateur de procédure (ce ne doit pas être un identificateur réservé : erreur syntaxique à vérifier, la grammaire en annexe en laissant la possibilité).

On conserve le type du résultat, l'adresse du résultat dans le segment, l'adresse en TABLE d'identificateursdes renseignements du 1er paramètre formel.

On positionne ML au-dessous de la place du résultat en zone locale (§2.4.3.).

G73. Paramètre effectif :

Si le pointeur dans TABLE vers les renseignements du paramètre formel vaut zéro, c'est que le nombre de paramètres effectifs est supérieur au nombre de paramètres formels (§2.4.6. et 2.4.3.).

On désigne par POINT l'adresse des renseignements du paramètre formel correspondant et on conserve l'adresse en TABLE des renseignements du paramètre formel suivant.

On vérifie la concordance de type entre le paramètre effectif et le paramètre formel correspondant (§2.4.3.).

Puis on place le paramètre effectif dans le segment (§2.4.6.).

Si c'est un élément, on met sa valeur dans le segment;

```
                (P6=1:( ), TOINTER(P3;P4;P5);
                P2=repère d'expression:
                          (ORDRE('LI';'3';'';1;ML;0);
                           ORDRE('AW';'3';'';0;'5';0);
                           ORDRE('CW';'3';'';1;P4;P5);
                           ORDRE('BG';0;''; 0;'$+3';0);
                           ORDRE('BAL';'2';'';0;'ERR';0);
                           ORDRE('DATA';0;'';0;'1';0), ( )));

GEN[ 74] = (4, 5) P4 ≠ 0:ERREUR(10),
                          (ORDRE('STW';'5';'';1;ML;'5');rangement
                                                        d'index
                          ORDRE('LI';'3';'';0;'$+6';0);  } ranger
                          I:=ML+1;                       } l'adresse retour
                          ORDRE('STW';'3';'';1;I;'5');   }
                          ORDRE('LI';'3';'';1;I;0);      } initialiser
                          ORDRE('AW';'3';'';0;'5';0);    } l'index
                          ORDRE('B';0;'';0;P3;0);        } branchement au
                                                           corps de procédure
                          ORDRE('LW';'5';'';0;'-1';'5'); } restaurer
                                                           l'index
                          ML:=P2;
                          P1=procédure sans résultat:
                              (P1:=sans résultat;ML:=ML-1),
                              (P3:=ML-1;P4:='5';P5:=1;
                          P1=procédure élément:
                              (P1:=élément;P2:=''),
                              (P2:='*';
                          P1=procédure repère:
                              (P1:=repère;
                          ORDRE('LI';'3';'';1;ML;0);
                          ORDRE('AW';'3';'';0;'5';0);
                          I:=ML-1;
                          ORDRE('CW';'3';'';1;I;'5');
                          ORDRE('BG';0;'';0;'$+3';0);
                          ORDRE('BAL';'2';'';0;'ERR';0);
                          ORDRE('DATA';0;'';0;'1';0)),
                              (P1:=rangée de repères))));
```


sinon (repère, rangée de repères d'éléments, procédure ou rangée de procédures), on représente le paramètre effectif par chaînage : on génère les ordres établissant le chaînage (TOINTER) si besoin est.
Si le paramètre effectif est un repère d'expression, on génère les ordres vérifiant que ce repère existe toujours (§2.4.5.3.).

G74. Fin des paramètres effectifs (§2.4.6.):

On vérifie qu'il ne reste plus de paramètres formels (§2.4.6.).
On génère les ordres qui:
- rangent l'index du segment appelant en zone locale
- mettent dans le registre 5 la valeur de l'index du segment appelé
- se branchent au début du corps de la procédure et,

au retour
- remettent dans le registre 5 la valeur d'index du segment appelant.

On remet ML au dessus du résultat.

Dans le cas d'un résultat repère, on génère les ordres vérifiant que ce repère existe toujours (§2.4.5.3.).

On conserve les 5 renseignements caractérisant le résultat.

4.5.2. Secondaire :

```
GEN[ 75] = (2, 5) P1=identificateur réservé:
                    (P1:=RP=P2:rangée en pile, repère ;
                     P3:=P2;
                     P2:=RP≠0:'*';'';
                     P4:='5';P5:=0),
                    (INDIC[ 1]:=P1;INDIC[ 2]:=P2;
                    (RANGEE REP PINF=RECHERCH(TABLE;INDIC;
                                              B;ERREUR);
                     P1:=DECOMP(2;PINF[ 1]);
                     P2:=DECOMP(3;PINF[ 1]);
                     TRAV:=DECOMP(4;PINF[ 1]);
                     I:=PINF[ 3];
                     P3:=TRAV=formel:I, P2=procédure élément:I,
                         P2= procédure sans résultat:I, P2=procédure repère,
                         P2=procédure rangée:I, PINF[ 2];
                     P4:=TRAV=formel:'5', TRAV;
                     P5:=0) );
GEN[ 78] = (0, 5) (P1:=élément;P2:='';P3:=COG-1;P4:='4';P5:=0);

GEN[ 77] = (7, 5) (P3=repère:( ), P3=repère d'expression:( ), P3=élément:( ),
                                                    ERREUR(15);
                   P7 <0:(REG:=P5;SOMMET:=SOMMET-1),
                         (REG:='3';OPERER('LW';REG;P4;P5;P6);
                    P7=0:( ), (ML:=ML-1));
                  ORDRE('BCS';'2';'';0;'$+3'0);
                  ORDRE('BAL';'2';'';0;'ERR';0);
                  ORDRE('DATA';0;'';'2';0);
                  P1=identificateur réservé:
                         (RP≠P2:ERREUR(12),
                     (P1:=repère;
                     ORDRE('LI';'2';'';1;P2;0);
                     ORDRE('AW';'2';'';0;'5';0))),
                  (INDIC[ 1]:=P1;INDIC[ 2]:=P2;
                   (RANGEE REP PINF=RECHERCH(TABLE;INDIC;B;ERREUR);
                   TRAV:=DECOMP(2;PINF[ 1]);
                   TRAV=rangée de repères:(P1:=repère);
                   TRAV=rangée d'éléments:(P1:=élément), ERREUR(12);
```


G75. Identificateur en primaire :

Si c'est un #i, ce peut être un repère ou une rangée en pile, il peut être représenté par chaînage ou non (§2.4.4.). On regarde pour cela le drapeau RP. L'adresse est i indexé par l'index local et ce n'est pas un résultat intermédiaire (P5 à zéro).

Sinon on recherche l'identificateur en TABLE (§2.4.2.) et on conserve les renseignements voulus : type, marque de chaînage (* si chaînage, sinon espace), adresse dans la zone, index de zone; ce n'est pas un résultat intermédiaire.

G78. Notation d'élément en primaire :

On conserve en pile les caractéristiques : type, pas de chaînage, rang dans la zone globale, index de zone globale ; ce n'est pas un résultat intermédiaire.

G77. Identificateur indicé en secondaire :

- On vérifie le type de l'expression en indice (§2.4.6.).

On génère les ordres SYMBOL plaçant l'élément i valeur de l'indice dans le registre de travail 3 sauf s'il est déjà dans un registre (P7=-1).

On génère les ordres vérifiant que l'indice i est strictement positif (§2.4.6.2.). On libère éventuellement le registre contenant l'indice ou on décrémente ML en zone locale.

- On regarde alors l'identificateur :

Si c'est un #i, ce doit être une rangée en pile (RP=i §2.4.4.), sinon on recherche en TABLE l'identificateur qui doit désigner une rangée. Dans les 2 cas on génère les ordres plaçant dans le registre 2 l'adresse a du début de la rangée.

```
                    TRAV:=DECOMP(4;PINF[1]);
                    ADR:=TRAV=formel:PINF[3];PINF[2];
                    DECOMP(3;PINF[1]='':
                     ORDRE('LW';'2';'';1;ADR;TRAV),
                     (ORDRE('LI';'2';'';1;ADR;0);
                     ORDRE('AW';'2';'';0;TRAV;0))));
                ORDRE('CW';REG;'*';0;'2';0);
                ORDRE('BLE';'';'';0;'$+3';0);
                ORDRE('BAL';'2';'';0;'ERR';0);
                ORDRE('DATA';0;'';0;'3';0);
                ORDRE('AW';'2';'';0;REG;0);
                ORDRE('STW';'2';'';1;ML;'5');
                P2:='*';P3:=ML;ML:=ML+1;P4:='5';P5:=1);
```

4.5.3. Expression simple :

4.5.3.1. Multiplication, division :

```
GEN[80]=(5, 5) (P1=élément:( ), DEREP(P1;P2;P3;P4;P5);
                P1:='MW';CCAR:=CCAR+1);

GEN[81]=(5, 5) (P1=élément:( ), DEREP(P1;P2;P3;P4;P5);
                P1:='DW';CCAR:=CCAR+1);

GEN[79]=(10, 5) (P6=repère:( ), P6=repère d'expression:( ),
                P6=élément:( ), ERREUR(21);
                P5 <0:(SOMMET:=SOMMET-1;
                       ORDRE('LW';'3';'';0;P3;0)),
                      (OPERER('LW';'3';P2;P3;P4);
                       P5=1:(ML:=ML-1), ( ) );
                P10 <0:(SOMMET:=SOMMET-1;
                        ORDRE(P1;'3';'';0;P8;0)),
                        (OPERER(P1;'3';P7;P8;P9);
                        P10=1:(ML:=ML-1), ( ) );
                P1='MW':(ORDRE('BCR';'4';'';0;'$+3';0);
                        ORDRE('BAL';'2';'';0;'ERR';0);
                        ORDRE('DATA';0;'';0;'4';0) ), ( );
                P1:=élément;P2:='';FINDOP(P3;P4;P5;'3') );
```


On génère les ordres comparant l'indice i à la taille de la rangée (§2.4.6.2.).

On génère les ordres mettant l'adresse a+i (§2.2.2.) en résultat intermédiaire en zone locale. Incrémentation de ML.

On conserve en pile les caractéristiques de cette valeur : type, marque de chaînage, adresse, index local, marque de résultat intermédiaire à 1.

G80, 81. Opérateur x ou / :

Si l'opérande gauche est un repère, on génère les ordres plaçant l'élément qu'il repère dans un registre ou à défaut en zone locale (DEREP) (§2.4.5.3.).

On conserve directement le code opération symbolique correspondant. On incrémente CCAR.

G79. Multiplication, division :

On vérifie le type de l'opérande droit .

On génère les ordres chargeant l'opérande gauche dans le registre 3 et si c'est un résultat intermédiaire on décrémente ML ou SOMMET selon le cas.

On génère les ordres effectuant l'opération avec l'opérande droit, si c'est un résultat intermédiaire, on décrémente ML ou SOMMET selon le cas.

Si c'est une multiplication, on génère un test de débordement (§2.4.6.2.).

On génère les ordres rangeant le résultat situé dans le registre 3 en zone locale (FINDOP) et on conserve les caractéristiques.

4.5.3.2. Addition, soustraction, +, - unaires;

```
GEN[ 82 ] = (0, 1) (P1:='AW';CCAR:=CCAR+1);

GEN[ 83 ] = (0, 1) (P1:='SW';CCAR:=CCAR+1);

GEN[ 84 ] = (6, 5) P1='AW':(P2=élément:( ), DEREP(P2;P3;P4;P5)),
                          (P2=élément:( ), P2=repère:( ),
                           P2=repère d'expression:( ), ERREUR(21);
                           P6 <0:(REG:=P4;
                                  ORDRE('LCW';REG;P3;0;P4;0)),
                                  (SOMMET=12:(REG:='3'),
                                              (REG:=TILREG[ SOMME
                                               SOMMET:=SOMMET+1);
                                  OPERER('LCW';REG;P3;P4;P5);
                                  P6=0:( ), (ML:=ML-1) );
                          P1:=élément;P2:='';FINDOP(P3;P4;P5;REG));

GEN[ 86 ] = (10, 5) (P6=élément:( ), P6=repère:( ),
                     P6=repère d'expression:( ), ERREUR(21);
                     P5 <0:(REG:=P3),
                           (REG:='3';OPERER('LW';REG;P2;P3;P4),;
                           P5=0:( ), (ML:=ML-1));
                    P10 <0:(ORDRE(P1;REG;P7;0;P8;P9);
                           SOMMET:=SOMMET-1),
                           (OPERER(P1;REG;P7;P8;P9);
                           P10=1:(ML:=ML-1), ( ) );
                    P1:=élément;P2:='';FINDOP(P3;P4;P5;REG) );

GEN[ 85 ] = (6, 5) (P1=élément:( ), DEREP(P1;P2;P3;P4;P5);
                    P1:=P6);

GEN[ 98 ] = (5, 5) (P1=élément:( ), DEREP(P1;P2;P3;P4;P5);
                    P1:='OR';CCAR:=CCAR+1);
```


G82, 83. Opérateur + -(unaire ou non):

Incrémentation de CCAR, on conserve le code symbol correspondant.

G84. Opération + ou - unaires:

Pour le + unaire, on se contente de 'dérépérer' s'il le faut.

Pour le - unaire, on vérifie le type de l'opérande.

On génère l'ordre symbol changeant le complément de l'opérande soit dans son registre ou dans un registre intermédiaire, ou dans le registre 3, et on décrémente ML si besoin est.

On génère les ordres plaçant, si besoin est, le résultat en zone locale ou dans un registre (FINDOP) et on conserve ses caractéristiques.

G86. Addition, soustraction, ou logique :

On vérifie le type de l'opérande droit.

On génère les ordres chargeant la valeur de l'opérande gauche dans le registre 3 s'il n'est pas déjà dans un registre. On décrémente ML si c'est un résultat intermédiaire local.

On génère les ordres effectuant l'opération avec l'opérande droit .

On décrémente SOMMET ou ML, selon le cas si c'est un résultat intermédiaire.

On génère les ordres plaçant, si besoin est, le résultat en zone locale (FINDOP) et on conserve ses caractéristiques.

G85. Opérateur + ou - binaire :

On 'dérépère' s'il le faut l'opérande gauche (§2.4.5.3.).

G98. Opérateur OU logique :

On 'dérépère' s'il le faut l'opérande gauche. On incrémente CCAR et on conserve l'opérateur symbol correspondant.

4.5.3.3. Relations :

```
GEN[87] = (5,5) (P1=élément:( ), DEREP(P1;P2;P3;P4;P5);CCAR:=CCAR+1;
                 P1:='BG');
```

procédures analogues avec affectation à P1 de l'opérateur de branchement symbol correspondant.soit:

- > BG
- < BL
- ≥ BGE
- ≤ BLE
- = BE
- ≠ BNE

```
GEN[92]

GEN[93] = (10,5) (P6=élément:( ), P6=repère:( ), P6=repère d'expression:0,
                                                  ERREUR(21);
          P5 <0:(REG:=P3),
                (REG:='3';OPERER('LW';REG;P2;P3;P4);
                 P5=1:(ML:=ML-1), ( ) );
          P10<0:(ORDRE('CW';REG;P7;0;P8;P9);
                 SOMMET:=SOMMET-1),
                (OPERER('CW';REG;P7;P8;P9);
                 P10=1:(ML:=ML-1), ( ) );
          ORDRE(P1;0;'';0;'$+3';0);
          ORDRE('LI';REG;'';0;'0';0);
          ORDRE('B';0;'';0;'$+2';0);
          ORDRE('LI';REG;'';0;'1';0);
          P1:=élément;P2:='';FINDOP(P3;P4;P5;REG));
```

4.5.3.4. NON, ET :

```
GEN[94] = (0,0) (CCAR:=CCAR+1);

GEN[95] = (5,5) (P1=élément:( ), P1=repère:( ), P1=repère d'expression:0,
                                                  ERREUR(21);
          P5 <0:(REG:=P3),
                (SOMMET=11:(REG:='3'),
                           (REG:=TILREG[SOMMET]);
                           SOMMET:=SOMMET+1);
                OPERER('LW';REG;P2;P3;P4);
                P6=0:( ), (ML:=ML-1) );
          ORDRE('BEZ';0;'';0;'$+3';0);
          ORDRE('LI';REG;'';0;'0';0);
          ORDRE('B';0;'';0;'$+2';0);
          ORDRE('LI';REG;'';0;'1';0);
          P1:=élément;P2:='';FINDOP(P3;P4;P5;REG) );
```


G87, 92. Opérateur de relation :

On dérépère l'opérande gauche. Incrémentation de CCAR.

On conserve l'opérateur Symbol de branchement correspondant au sens de la relation.

G93. Rélation :

On vérifie le type de l'opérande droit.

On génère les ordres chargeant l'opérande droit dans le registre 3 s'il n'est pas dans un registre et on décrémente ML si c'est un résultat intermédiaire local.

On génère les ordres de comparaison avec l'opérande gauche. On décrémente ML ou SOMMET, selon le cas, si c'est un résultat intermédiaire.

On génère les ordres chargeant dans le registre qui contenait l'opérande 3 : 0 pour FAUX ou 1 pour VRAI selon le résultat de la comparaison.

On génère, si besoin est, les ordres rangeant les résultats en zone locale et on conserve ses caractéristiques.

G94. Opérateur non :

On incrémente CCAR.

G95. Opération non :

On vérifie le type de l'opérande.

On génère s'il n'y est pas déjà, le chargement de l'opérande dans un registre intermédiaire (alors on incrémente SOMMET) ou à défaut dans le registre 3. On décrémente ML si c'est un résultat intermédiaire local.

On génère un test sur code de condition à zéro (FAUX) et selon le cas, on charge dans le registre qui contenait l'opérande l'entier 0 pour FAUX ou 1 pour VRAI.

On génère, si besoin est, les ordres rangeant le résultat en zone locale (FINDOP) et on conserve ses caractéristiques.

```
GEN[ 96] = (5, 5) (P1=élément:( ), DEREP(P1;P2;P3;P4;P5);
                   CCAR:=CCAR+1);

GEN[ 97] = (10, 5) (P6=élément:( ), P6=repère:( ), P6=repère d'expression:();
                   ERREUR(21);
                   P10 <0: (REG:=P8;SOMMET:=SOMMET-1),
                           (REG:='2';OPERER('LW';REG;P7;P8;P9);
                           P10=1:(ML:=ML-1), ( ) );
                   P5 <0 : (REG1:=P3),
                           (REG1:='3';OPERER('LW';REG;P2;P3;P4);
                           P5=1:(ML:=ML-1), ( ) );
                   ORDRE('LI';REG1;'';0;'0';0);
                   ORDRE('BE';0;'';0;'$+5';0);
                   ORDRE('CI';REG;'';0;'0';0);
                   ORDRE('BE';0;'';0;'$+3';0);
                   ORDRE('LI';REG1;'';0;'1';0);
                   ORDRE('B';0;'';0;'$+2';0);
                   ORDRE('LI';REG1;'';0;'0';0);
                   P1:=élément;P2:='';FINDOP(P3;P4;P5;REG1) );
```

4.5.4. Expression.

```
GEN[ 100] = (5, 1) (P1=élément:( ), P1=repère:( ), P1=repère d'expression:( );
                                                   ERREUR(21);
                   P1:=CREETI( );
                   P5 <0:(SOMMET:=SOMMET-1),
                         (REG:='3';OPERER('LW';REG;P2;P3;P4);
                          P5=1;(ML:=ML-1), ( ) );
                   ORDRE('BEZ';0;'';0;P1;0) );

GEN[ 101] = (6, 2) (P2=sans résultat:( ),
                   P2=élément:(P6=1:DEPOINT(P3;P4;P5),
                                    TLACER(P3;P4;P5;P6) ),
                              (P2=repère:( ), P2=repère d'expression:
                                    (P2:=repère), ERREUR(13);
                              P6=1:( ), TOINTER(P3;P4;P5) );
                         ML:=ML-1);
                   TRAV:=CREETI( );
                   ORDRE('B';0;'';0;TRAV;0);
                   ETI:=P1;P1:=TRAV);
```


G96. Opérateur et :

On 'dérépère' s'il le faut l'opérande gauche. Incrémentation de CCAR.(§2.4.5.3.).

G97. Opération et :

On vérifie le type de l'opérande droit.

On génère le chargement de l'opérande droit dans le registre 2, s'il n'est pas déjà dans un registre. On décrémente ML ou SOMMET selon le cas, si c'est un résultat intermédiaire.

On génère le chargement de l'opérande gauche dans le registre3, s'il n'est pas déjà dans un registre. On décrémente ML si c'est un résultat intermédiaire local.

On génère les ordres regardant si l'opérande gauche a la valeur zéro (FAUX).

On génère les ordres regardant si l'opérande droit a la valeur zéro (FAUX).

Si les deux opérandesvalent VRAI, on génère le chargement de 1 dans le registre qui contenait l'opérande gauche sinon on y charge 0.

On génère les ordres rangeant en zone locale le résultat s'il le faut (FINDOP) et on conserve ses caractéristiques.

G100. Expression simple (£1) précédent le symbole deux points : (2.4.5.2.).

On vérifie son type.

On crée l'étiquette e (2.4.5.2) que l'on conserve.

On génère le chargement du résultat de l'expression dans le registre 3 s'il n'est pas déjà dans un registre. On décrémente ML ou SOMMET si c'est un résultat intermédiaire.

On génère le branchement(si ce résultat vaut zéro (FAUX)) à l'étiquette créée(§2.4.5.2).

G101. Expression simple (£2) avant le symbole virgule : (§2.4.5.2.).

On vérifie son type.

Si c'est un résultat élément, on place l'élément en zone locale s'il n'y est pas déjà et dans ce cas, on supprime l'adressage indirect.

Si c'est un repère, on place en zone locale l'adresse de l'élément qu'il repère sauf si elle y est déjà.

On positionne ML sous ce résultat (§2.4.5.2).

On crée l'étiquette f.

On génère le branchement à l'étiquette e.

On conserve l'étiquette g qu'on génère(2.4.5.2.) ainsi que le type de l'expression simple.

```
GEN[ 102] = (7, 5) P2=sans résultat :
                   (P3=sans résultat:
                         (ETI:=P1;ORDRE('EQU';0;'';0;'$';0);
                          P1:=sans résultat), ERREUR(13) ),
                   P2=élément:
                   (P3=élément:( ), P3=repère:( ), P3=repère d'expression:( ),
                    ERREUR(13);
                    P7=1:DEPOINT(P4;P5;P6), TLACER(P4;P5;P6;P7);
                    ETI:=P1;ORDRE('EQU';0;'';0;'$';0);
                    P1:=élément;P2:='';P3:=ML-1;P4:='5';P5:=1),
                   (P3=élément:
                    (P7=1:DEPOINT(P4;P5;P6), TLACER(P4;P5;P6;P7);
                    TRAV:=CREETI( );
                    ORDRE('B';0;'';0;TRAV;0);
                    ETI:=P1;I:=ML-1;
                    ORDRE('LW';'3';'';1;I ;'5');
                    ORDRE('LW';'3';'*';0;'3';0);
                    ORDRE('STW';'3';'';1;I;'5');
                    ETI:=TRAV;ORDRE('EQU';0;'';0;'$';0);
                    P1:=élément;P2:='';P3:=I;P4:='5';P5:=1),
                   (P3=repère:( ), P3=repère d'expression:( ), ERREUR(13);
                    P7=1:( ), TOINTER(P4;P5;P6);
                    ETI:=P1;ORDRE('EQU';0;'';0;'$';0);
                    P1:=repère d'expression;P2:='*';P3:=ML-1;
                    P4:='5';P5:=1) );
```

4.5.5. Expression fermée :

```
GEN[ 112] = (10, 5) (P1=rangée en pile:(RP:='*'), P1=repère:( ), ERREUR(18);
                    P6=élément:( ), P6=repère:( ), P6=repère d'expression:( ),
                     ERREUR(21);
                    P10 <0:(REG:=P8;SOMMET:=SOMMET-1),
                           (REG :='3';OPERER('LW';REG;P7;P8;P9);
                            P10=1:(ML:=ML-1), ( ) );
                    OPERER('STW';REG;P2;P3;P4);
                    P1:=sans résultat;
                    P5=1:( ), (ML:=ML-1) );
```


G102. Expression ($\mathcal{E}3$) suivant le symbole virgule : (§2.4.5.2).

On vérifie la conformité de type avec l'expression $\mathcal{E}2$ dans chaque cas.

Si l'expression est sans résultat, on génère l'étiquette f (§2.4.5.2.) et on conserve les caractéristiques du résultat.

Si l'expression $\mathcal{E}2$ a pour résultat un élément :

- on place l'élément résultat (ou repéré par le résultat) de l'expression suivant sinon en zone locale.
- On génère l'étiquette f et on conserve les caractéristiques du résultat.

Si l'expression $\mathcal{E}2$ est un repère ;

Si l'expression $\mathcal{E}3$ est un élément :

- on génère la place de l'élément en zone locale si besoin est.
- on crée l'étiquette g (§2.4.5.2.).
- on génère le branchement à l'étiquette g.
- on génère l'étiquette f.
- on prend l'élément repéré par $\mathcal{E}2$ qu'on range en zone locale.
- on génère l'étiquette g.
- on conserve les caractéristiques du résultat élément.

Sinon, ce doit être un repère :

- on place l'adresse de l'élément repéré en zone locale si besoin est
- on génère l'étiquette f
- on conserve les caractéristiques du résultat repère.

Remarque : Considérons l'expression suivante :

$\varepsilon_1 : \beta 1,\ \varepsilon_2 : \beta 2,\ \varepsilon_3,\ \beta 3,\ \beta 4$

En ce cas, on génère 3 étiquettes différentes de type f (§2.4.5.2.) derrière $\beta 4$. C'est pourquoi on génère la directive 'EQU $' derrière les étiquettes f, ce qui donne aux étiquettes différentes le même emplacement à l'assemblage.

G112. Affectation :

Le premier membre doit être un repère (et non un 'repère d'expression') (§2.4.5.). Si c'est une rangée en pile #i, celle-ci disparait (§2.4.4.), on positionne donc RP à '*'; on vérifie le type du 2ème membre.

On génère le chargement dans le registre 3 du 2ème membre s'il n'est pas déjà dans un registre. On décrémente ML ou SOMMET si c'est un résultat intermédiaire.

On génère le rangement à l'adresse du 1er membre.

Si le 1er membre est un repère résultat intermédiaire local (§2.4.5.), on décrémente ML.

```
GEN[114] = (5, 0) P1=sans résultat:( ), ERREUR(17);

GEN[115] = (0, 5) (P1:=sans résultat);

GEN[117] = (1, 0) ( );

GEN[116] = (7, 5) (ML:=P1;COURS[NO]:=P2;P1:=P3;
                   P3=sans résultat:( ),
                         P3=élément:(TLACER(P4;P5;P6;P7);
                                     P2:='';P3:=ML-1;P4:='5';P5:=1);
                                     (P3=repère:P3:=repère d'expression,
                                     P3=repère d'expression:( ),
                                     ERREUR(2);
                                     TOINTER(P4;P5;P6);
                                     P2:='*';P3:=ML-1;P4:='5';P5:=1) );
```

4.6. PROGRAMME :

```
GEN[120] = (5, 0) P1=sans résultat :
                  (ERR=0:( ), ARRT( );
                  ORDRE('CAL1';'9';'';0;'1';0);
                  CG=1:( ),
                       (ETI:='RP';ORDRE('EQU';0;'';0;'$';0);
                        I:=CG-1;
                        ECRIRANG(Z;I) );
                  ORDRE ('END';0;'';0;'DEB';0) ),
                  ERREUR(19);
```


G114. Expression terminant une expression fermée : doit être sans résultat.

G115. Fin vide d'expression fermée :

On donne les caractéristiquec du résultat par homogénéité.

G117. Fin des déclarations locales débutant par une rangée : on ne conserve plus la taille de cette rangée.

G116. Fin d'une expression fermée bloc : (§2.4.2.).

Restauration de ML, NO. Mise à jour de COURS.

Si l'expression fermée a un résultat, on génère les ordres plaçant ce résultat en zone locale à l'endroit prévu : l'élément résultat lui-même ou pour un résultat repère, représentation du repère par chaînage (§2.4.5.1.).

On conserve ses caractéristiques.

G120. Fin de la génération :

On vérifie que l'expression finale est sans résultat.

On génère la fin du rpogramme objet :

- S'il n'y a pas eu d'erreur à la compilation : fermeture du fichier programme objet (ARRT). (En effet, si le fichier programme objet n'est pas fermé, les étapes suivantes du travail ne pourront avoir lieu, donc l'assemblage Symbol n'aura pas lieu).

- ordre d'arrêt d'exécution.

- directives Symbol créant à l'assemblage la zone rangée de procédures si elle existe.

- directive Symbol END.

CHAPITRE 5

PROCEDURES

B.

```
PROC ELT B = (RANGEE REP PX)
          (I:=DECOMP(1;PX[1]);COURS[I]);
```

B1, B2.

```
PROC ELT B1(resp. B2)=(RANGEE REP PX)
          DECOMP(1;PX[1])=terminal(resp. non terminal):0,
          DECOMP(2;PX[1])≠· CGRAM:0, 1;
```

B. Filtre pour rechercher en TABLE un identificateur (§2.4.2.).

Paramètre : rangée de repères contenant les renseignements en table.

Résultat : booléen.

Résultat Faux (0), si le renseignement du synonyme de l'identificateur cherché est un bloc non valide, 1 sinon.

B1, B2. Filtres pour rechercher en TABLE1 (§2.4.1.) un terminal (resp. un non terminal)

Paramètre : rangée de repères contenant les renseignements en table.

Résultat : booléen.

Résultat faux (0), si le renseignement est celui d'un terminal (resp. non terminal) ou d'une autre grammaire, sinon 1.

COMPAR But : Procédure de construction de la ramification d'une grammaire (§2.2.1.).

COMPAR est utilisée dans le cas ou un terminal ou un non terminal d'un 2ème membre de règle est comparé dans la ramification avec un bloc d'un terminal ou d'un non terminal d'une alternative précédente de la même règle.

Exemple : A:B,D|20|B,C,E|30;

Cas 1 :

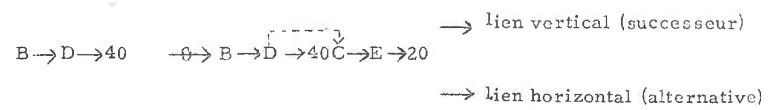
Pour le non terminal B de la règle 30, on compare avec le non terminal B de la règle 20.

Dans ce cas, les non terminaux sont identiques. On n'a rien à ajouter à la ramification pour l'instant. La comparaison suivante pour E aura lieu avec le bloc lien vertical de B, soit celui de D.

Cas 2.

Pour le non terminal C de la règle 30, on compare avec le non terminal D de la règle 20.

Dans ce cas, ils sont différents et D n'a pas de bloc lien horizontal. Un bloc pour C est placé dans la ramification et ce bloc est mis en alternative (lien horizontal) avec celui de D.



COMPAR.

```
PROC COMPAR=(REP PRANG, PCOURS, PNDI)
      (TRAV:=PNDI+PNDI;
       I:=PCOURS+1;
       DECOUP(1;G[I])=DECOUP(2;TRAV):
          (PCOURS:=DECOUP[2;G[I]]),
          (I=PCOURS+2;TRAV:=DECOUP(2;G[I];
           TRAV=0:(DEMIMOT(2;G[I];PRANG);
                   PCOURS:=0;
                   I:=PRANG+3;
                   GRG(PRANG;PNDI;I;0)),
                  (PCOURS:=TRAV;
                   DEBIT(2;TRAV):
                         REPOUS (PRANG;PCOURS;PNDI),
                         COMPAR(PRANG;PCOURS;PNDI))));
```


Cas 3. A:B,D |20 |B,C,E| 30 |B,F,G |40;

Pour le non terminal F de la règle 40, on compare avec le non terminal D de la règle 20.

Dans ce cas, il sont différents mais D a une alternative qui est le bloc de C.

On compare alors F avec le non terminal C. On est alors amené à retrouver les trois cas suivants.

Cas a : Bloc en alternative d'un bloc fin de règle, on appelle alors la procédure Repous .

Exemple : A:B,D |20 |B |30 |B,F,G |40;

Pour ces 2 cas il suffit de recommencer la procédure compar avec des paramètres différents.

Cas b : Bloc en alternative différent, on est ramené au cas 2 ou 3.

Exemple : A:B, D/20/B, C/30/B, F, G/40;

Cas c : Bloc en alternative identique, on est ramené au cas 1.

Exemple : A:B, D/20/B, D, E/30/B, F, G/40;

Paramètres:

PRANG Pointeur dans la ramification.

PCOURS. Rang dans la ramification du bloc à comparer.

PNDI. Numéro d'identification du terminal ou du non terminal.

Analyse :

Si le numéro d'identification du terminal ou du non terminal est égal à celui du bloc à comparer alors,

Cas 1 : Le paramètre PCOURS est alors égal au successeur (lien vertical) de ce bloc.

Sinon, s'il n'y a pas d'alternative (lien horizontal nul)

Cas 2 : On construit un bloc pour le non terminal ou terminal au rang FRANG(GRG). Ce bloc est placé en alternative du bloc concerné. Le paramètre PCOURS est mis à zéro afin de copier directement dans le graphe les blocs qui terminent la règle en question.

Sinon :

Cas 3 : Si le bloc en alternative est un bloc de fin de règle, on appelle la procédure REPOUS, sinon on appelle récursivement la procédure COMPAR après avoir fait pointer PCOURS sur ce bloc en alternative.

CREETI.

```
PROC ELT CREETI=( ) (RANGEE REP PI=BD(CETI);
                     CETI:=CETI+1;
                     OCTET(1;PI[2];'E');PI[2]);
```

DEPOINT.

```
PROC DEPOINT=(REP PAI, PADR, PRI)
        PAI=' ':( ),
                     (ORDRE('LW';'3';'';1;PADR;PRI);
                      ORDRE('LW';'3';'';0;'3';0);
                      ORDRE('STW';'3';'';1;PADR;PRI));
```

DEREP.

```
PROC DEREP=(REP PTYP, PAI, PADR, PRI, PRINT)
        (PTYP=repère:( ), PTYP=repère d'expression:( ), ERREUR(21);
         SOMMET=11:(REG='3'),
                     (REG:=TILREG[SOMMET];SOMMET:=SOMMET+1);
         OPERER('LW';REG;PAI;PADR;PRI);
         PAI='';PTYP:=élément;
         REG='3':(ORDRE('STW';REG;'';1;ML;'5');
                  PADR:=ML;ML:=ML+1;PRI:='5';PRINT:=1),
                  (PADR:=REG;PRI:=0;PRINT:=0-1) );
```

FINDOP.

```
PROC FINDOP=(REP PADR, PRI, PRINT, PREG)
        PREG='3':(SOMMET=11:
                  (PADR:=ML;ML:=ML+1;PRI='5';PRINT=1
                   ORDRE('STW':PREG;'';1;ML;PRI)),
                  (PADR:=TILREG[SOMMET];SOMMET:=SOMMET+1;
                   PRI:=0;PRINT:=0-1;
                   ORDRE('STW';PREG;'';0;PADR;PRI))),
        (PADR :=PREG;PRI:=0;PRINT:=0-1);
```


CREETI. Création d'adresses symboliques objets (ei) toutes différentes grâce au compteur CETI donnant i.

Résultat l'étiquette ei.

Incrémentation de CETI dont on concatène la valeur décimale avec le caractère E.

DEPOINT. Génération des ordres supprimant l'adressage indirect, s'il existe, pour un résultat intermédiaire en zone locale.

Paramètres marque de chaînage PAI, adresse en zone locale PADR, index PRI.

DEREP. Un repère étant employé dans un contexte d'élément, DEREP génère les ordres plaçant l'élément repéré dans un registre ou en zone locale et affecte aux paramètres ses nouvelles caractéristiques (§2.4.5.1.).

Paramètres marque de type PTYP,

Adresse PADR, marque de chaînage PAI, index PRI, marque de résultat intermédiaire PRINT.

On vérifie le type (repère). On génère les ordres chargeant l'élément dans un registre intermédiaire, (alors on incrémente SOMMET) ou dans le registre 3 (OPERER) (Dans ce cas, on génère les ordres rangeant l'élément en zone locale et on incrémente ML.)

On affecte aux paramètres les nouvelles caractéristiques de l'élément.

FINDOP. Après un calcul laissant un élément dans un registre intermédiaire, ou dans le registre 3 : attribuer aux paramètres les nouvelles caractéristiques après avoir généré les ordres Symbol replaçant le résultat soit en zone locale, soit si possible dans un registre intermédiaire sauf s'il y est déjà.

Incrémentation de ML ou de SOMMET s'il le faut.

Paramètres adresse PADR, index PRI, marque de résultat intermédiaire PRINT, registre PREG.

FILTRE.

```
PROC ELT FILTRE=(RANGEE REP PX)
      (TRAV:=DECOMP(1;PX[1]);
       NO < TRAV:0, NO=TRAV:(ERREUR(B);1), 1);
```

FILTRE1.

```
PROC ELT FILTRE1=(RANGEE REP PX)
      DECOMP(2;PX[1])≠ CGRAM:0, DECOMP(1;PX[1])=terminal :0,
            (ERREUR(2);1);
```

FILTRE2.

```
PROC ELT FILTRE2=(RANGEE REP PX)
      DECOMP(2;PX[1])≠ CGRAM:0, DECOMP(1;PX[1])=non terminal:0,
            (TRAV2:=DECOMP(4;PX[1]);
             I:=TRAV2 <4:1, TRAV2 <8:2, 3;
             TRAV:=TRAV-I;RETOUR( );1);
```

GRG.

```
PROC GRG=(REP PRANG, PNDI, PSUC, PALT)
      (G[PRANG]:=0;
       PNDI <0:( ), BIT(1;G[PRANG];1);
       PRANG:=PRANG+1;
       G[PRANG]:=PSUC;
       TRAV:=PNDI+PNDI;
       DEMIMOT(1;G[PRANG];TRAV);
       PRANG:=PRANG+1;
       G[PRANG]:=PALT;
       PRANG:=PRANG+1 );
```

MRAPPL.

```
PROC MRAPPL=(ELT PN, PET)
      (Z[CG]:=PET;CG:=CG+1;PN=1:( ), (I:=PN-1;MRAPPL(I;PET)));
```


FILTRE. Filtre pour adjoindre à TABLE un identificateur (§2.4.2.).

Paramètre : rangée de repères des renseignements en table.

Résultat booléen.

Si le renseignement provient d'un identificateur d'un bloc plus grand que le bloc actif alors 0 ;

en cas d'égalité, c'est une double déclaration (erreur) ;

sinon 1.

FILTRE1. Filtre pour adjoindreen TABLE 1 un non terminal (§2.4.1.)

Paramètre : rangée de repères des renseignements en TABLE1

Résultat: booléen.

Si le renseignement du synonyme est d'une autre grammaire : faux, ou de la même grammaire pour un terminal : faux, sinon c'est qu'il y a dans la grammaire 2 non terminaux identiques au 1er membre : c'est une erreur (si on ne veut pas faire cette vérification FILTRE1 est une tautologie).

FILTRE 2. Filtre pour adjoindre à TABLE 1 un terminal . (§2.4.1.)

Paramètre : rangée de repères des renseignements d'un synonyme en TABLE1.

Résultat : booléen.

Si le renseignement du synonyme est celui d'une autre grammaire faux, ou d'un non terminal de la même grammaire : faux, sinon le même terminal a déjà été trouvé dans la grammaire : il faut diminuer TRAV qui contient le rang dans la table des terminaux de la longueur du terminal (car on l'a incrémenté avant (§2.2.1.: L23). Il suffit alors de revenir dans L 23 (RETOUR) pour ne pas entrer une 2ème fois ce terminal en TABLE 1.

GRG . Placer un noeud dans la ramification d'une grammaire située dans G (§2.2.1.).

Paramètres : rang dans la ramification PRANG, numéro d'identification PNDI, rang du noeud lien vertical PSUC, rang du noeud lienhorizontal PALT.

Mise à zéro du 1er mot et

mise à 1 du 1er bit si c'est un terminal.

Placer le rang lien vertical dans le 2ème mot ainsi que le numéro d'identification

Placer le rang du lien horizontal dans le 3ème mot

Incrémenter PRANG

MRAPPL. Au début d'une rangée de procédures :

On place dans la table décrivant la rangée située dans Z, l'adresse symbolique débutant la rangée dans le Programme objet (§2.2.3.).

Paramètres: PN taille de la rangée donc de la table.

PET étiquette.

```
OPERER
PROC OPERER = (REP , PREG, PAI, PADR, PRI)
     PAI = '*' : (ORDRE ('LW';'2';'';1;PADR;PRI);
                  ORDRE (PCO;PREG;PAI;0;'2';0));
                  ORDRE (PCO;PREG;PAI;1;PADR;PAI);
```

RECHNIL

On génère la suite des ordres ci-dessous :

```
    LI, 3   0        3 calcule long
    LW, 2   0, 5     au rang 0 dans le segment est la cellule sommet de # k,
                     2 contiendra la valeur d'une cellule de la rangée, 5
    CI, 2   NIL      pointera sur une cellule de # k.
    BE      $+4      la cellule considérée contient nil.
    AI, 5   -1
    AI, 3   1
    B       $-5      On comparera à la cellule suivante.
    STW, 3  0, 5     On placelong dans la cellule contenant nil. 5 pointe
                     sous cette cellule
    LW, 2   3
    AI, 2   PN1      silong+ k > ℓ branchement à ETIO
    AI, 2   PN2
    BG      ETIO
    Cas ℓ > k + long
    AI, 5   PN2-PN1  5 pointe sur # ℓ
    LW, 2   5
    AI, 2   -PN2     2 pointe sous # 1
    LI, 3   PN2      on doit établir ℓ cellules fictives
    AI, 2   1
    AI, 5   1
    STW, 2  *5       boucle mettant dans les cellules fictives les adresses
    BDR     $-3      des # i
    AI, 5   -PN2
    B       RFIN     5 pointe sur # 1.
    Caslong+ k > 1
ETIO LW, 2  5
    AI, 2   -PN1     2 pointe sous # 1
    AW, 5   3        5 pointe le sommet de # k
    LI, 3   PN2
    CI, 3   PN1
    BG      $+2
```


OPERER. Si un opérande n'est pas dans un registre, génération des ordres Symbol effectuant l'opération donné sur cet opérande avec le registre donné.

Paramètres PCO code symbolique,
PREG, registre, PAI, PADR, PRI caractèristiques de chaînage, d'adresse et d'index de l'opérande.

RECHNI IL. Pour une en tête (k, rangée, ℓ) génération des ordres: (§2.4.4.).

- recherchant la longueur lors de la rangée # k et la cellule contenant nil.
- plaçant long à la place de nil
- établissant r = sup(k, ℓ) cellules fictives sur la pile
- laissant l'index au niveau de la 1ère allure fictive (§2.8.2.).

Paramètres : PN1 et PN2 repérant les valeurs respectives k et l.

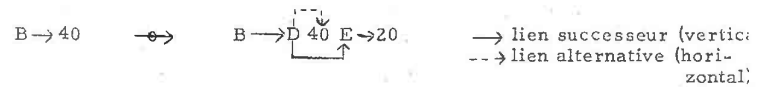
```
      LI, 3    PN1
      AI, 2    1
      AI, 5    1
      STW 2    *5          établissant des cellules fictives pointant vers les #i
      BDR, 3   $-3
      LI, B    PN2
      CI, 3    PN1
      BG       $+3         5 pointe sous les cellules fictives.
      AI, 5    -PN2
      B        $+2
      AI, 5    -PN1
RFIN  EQU      $
```

```
REPOUS.
PROC REPOUS = (REP PRANG, PCOURS, PNDI)
        (G[ PRANG] :=G[ PCOURS] ;
         PRANG:=PRANG+1;
         PCOURS:=PCOURS+1;
         G[ PRANG] :=G[ PCOURS] ;
         PRANG:=PRANG+1;
         PCOURS:=PCOURS+1;
         PRANG:=PRANG+1;
         TRAV:=PCOURS-2;TRAV:=PRANG-3;GRG(TRAV;PNDI;PRANG;TRAV2);
         PCOURS:=0);
```


REPOUS BUT procédure de construction de la ramification représentant la grammaire. Repous est utilisé dans le cas où un terminal ou un non terminal d'un 2ème membre de règle est comparé dans la ramification avec un bloc de fin de règle (§2.2.1.).

Exemple : A:B /40/B,D,E/20;

Le non terminal D provoque une comparaison avec le bloc de fin de règle 40. Son bloc devrait être alternative de celui ci, ce qui est interdit (§2.2.1.). Aussi on place le bloc concernant D à la place du bloc fin de règle 40 afin qu'il soit successeur de B. On recopie le bloc fin de règle 40 derrière celui ci et on le met alternative de D.



(Le bloc E et le bloc de fin de règle 20 seront placés ensuite).

Paramètres PRANG rang dans la ramification, PCOURS, rang dans la ramification du noeud que l'on compare (donc ici d'un bloc fin de règle), PNDI numéro d'identification.

Analyse

On recopie au rang PRANG les 3 mots du bloc de fin de règle situé au rang PCOURS. On incrémente PRANG.

On place dans la rangée G à partir du rang PCOURS, le bloc concernant le terminal ou le non terminal avec le nouveau rang du bloc fin de règle comme alternative et le rang dans la ramification comme successeur supposé. On remet le pointeur PCOURS à zéro afin de placer directement dans la ramification les blocs qui terminent la règle en question (gen 27).

```
TLACER
PROC TLACER = (REP PAI,  PADR, PRI, PRINT)
      PRINT <0 : (ORDRE('STW';PADR;''; 1;ML;'5');
                  SOMMET:=SOMMET-1;ML:=ML+1);
                 (ORDRE('LW';'3';'';1;PADR;PRI);
                  PAI='':( ), ORDRE('LW';'3';'';0;'3';0);
                  ORDRE('STW';'3';'';1;ML;'5');
                  ML:=ML+1);

TOINTER
PROC TOINTER = (REP PAI, PADR, PRI)
      PAI ='*' : (ORDRE('LW';'3';'';1;PADR;PRI);
                 ORDRE('STW';'3';'';1;ML;'5');
                 ML:=ML+1);
                 (ORDRE('L';'3';'';1;PADR;0);
                  ORDRE('AW';'3';'';0;PRI;0);
                  ORDRE('STW';'3';1;ML;'5');ML:=ML+1);
```


TLACER: génère les ordres plaçant un élément à l'adresse ML en zone locale. Incrémentation de ML.

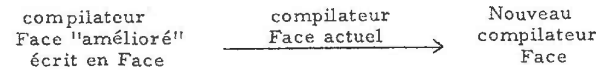
Paramètres : PAI marque de chaînage, PADR, adresse , PRI, index, PRINT marque de résultat intermédiaire.

TOINTER :génère les ordres plaçant un repère en zone locale à l'adresse ML. Incrémentation de ML.

Paramètres : PAI marque de chaînage, PADR, adresse, PRI index.

CONCLUSION

Nous avons décrit un compilateur Face écrit en Face 0. Il serait maintenant utile de réécrire ce compilateur en Face : il ne serait pas essentiellement modifié, mais son efficacité pourrait être améliorée, notamment en effectuant la compilation en deux passages par l'utilisation d'ANAPPLYSE.



L'expression Face finale s'écrirait

(ANAPPLYSE(120;CHAINE;LEX);APPL(CHAINE;GEN))

On pourrait aussi générer directement du MODULEOBJET 10070 et non du SYMBOL grâce à une procédure de génération autre que ORDRE.

L'emploi de Face pour écrire des compilateurs divers devrait conduire à des améliorations de la définition du langage. Actuellement, un compilateur Algol 60 est écrit en FACE.

Pour toute nouvelle définition, la nouvelle version du compilateur Face sera obtenue facilement à partir de la précédente et sa réalisation sera de plus facilitée par la modularité de l'écriture d'un compilateur en FACE.

ANNEXE 1

RESTRICTIONS APPORTEES A LA DEFINITION DE FACE POUR DEFINIR LE LANGAGE FACE 0.

I.1 RESTRICTIONS FACE 0

Le langage noyau Face 0 sert uniquement à implémenter le premier compilateur Face. Nous reprenons le plan de la définition de Face (§1.4.) et nous signalons les restrictions du langage Face 0[9].

Symboles de base (§1.4.1.).

- le caractère espace Face 0 est le blanc.
- Les entiers sont uniquement écrit en base décimale.
- Les identificateurs dont la première lettre est un p sont des identificateurs réservés : en Face les seuls identificateurs réservés ont la forme #i (i entier) et désignent les paramètres des procédures d'une rangée. En Face 0, ils ont la forme p suivie d'un entier. Les autres identificateurs réservés commençant par la lettre p désignent les paramètres formels des procédures ou les identificateurs locaux.

Programme (§1.4.2.).

. Le programme Face 0 ne comporte pas de <définition des grammaires> : la procédure analyse Face 0 qui permet de réaliser l'analyse syntaxique d'un programme Face lit sur disque directement la grammaire définissant la syntaxe du langage Face convenablement codée (§2.2.1.).

Cette grammaire est donnée en annexe 2 sous forme de notation de Backus : la procédure analyse Face 0 suit une méthode d'analyse descendante [13] . Elle accepte ce pendant une grammaire récursive à gauche. (qui comprend des règles de production de la forme A ::= A φ ou A est un non terminal et φ un mot sur la réunion des vocabulaires terminaux et non terminaux [13]). Elle fait ses tests par lecture d'un caractère à l'avance (Langage LL(1)) ce qui impose certaines conditions qui expliquent les différences entre la grammaire exposée au chapître 1 et celle qui est donnée à la procédure analyse (Annexe 2).

Les déclarations : (§1.4.4.).

- Les identificateurs d'éléments ou de rangée d'éléments n'existent pas en Face 0.

On ne peut faire repérer une valeur initiale pour les identificateurs désignant des repères ou des rangées de repères.

Les déclarations de procédures (§1.4.5.).

En face 0, les procédures à résultat repère (§1.4.5.1.) ne sont pas admises.

Signalons que les procédures récursives sont permises en Face 0 comme en Face mais en Face 0 toute procédure doit être déclarée avant son 1er appel.

Rappelons que les paramètres formels Face 0 débutent par la lettre P.

- Les procédures standard Face 0 sont décrites en Annexe 1.2 (§1.4.5.2.).

- Le langage Face 0 ne comporte pas de procédures écrites en code machine (§1.4.5.3.).

Les rangées de procédures (§1.4.6.).

- Les procédures de la liste sont obligatoirement dans l'ordre de leurs indices.

- Une <en tête> ne peut comporter de rangée : les paramètres ne peuvent être de type rangée en pile. Rappelons que ces paramètres sont des identificateurs réservés de forme pi en Face 0 (i entier).

Les expressions fermées (§1.4.7.).

- Une expression fermée ne peut avoir un résultat de type repère.

Déclaration locale (§1.4.7.1.).

. En Face 0, il n'y a pas en réalité de déclaration locale de repères ou de rangées de repères : il est juste permis de déclarer des identificateurs désignant des repères ou des rangées de repères déjà créés (évaluation ou valrang commençant par = sont seules permises).

Les identificateurs ainsi déclarés sont des identificateurs réservés Face 0 commençant par la lettre P.

Dans un même bloc (expression fermée ou corps de procédure) on n'admet pas de double définition d'identificateur réservé Face 0.

Expression (§1.4.7.3.). Elle ne peut avoir un résultat de type repère.

Expression simple : (§1.4.7.4).

Les seuls opérateurs admis sont les opérateurs d'addition (+ primaire) soustraction (- binaire) et les opérateurs de relation.

(=, ==, >, ≥, <, ≤).

Appel de procédures (§1.4.7.5.).

Les paramètres effectifs sont uniquement des identificateurs (et non des expressions).

Ils peuvent être des éléments uniquement pour les appels de procédures standard.

Identificateur indice (§1.4.7.5.).

Un indice ne peut être qu'un identificateur ou un élément (et non une expression).

) Gestion de table :

Nom	Paramètres	Résultat	Rôle
ıble, table 1 ...	rangée de repères : indic	la i^{ème} rangée de repères (information) sélectionnée par l'indicatif indic.	trouve la place de l'information associée à un indicatif donné dans une table, pour l'entrée désignée par l'entier i (1. 3.).
ıble(indic, i)	élément : i		
entrée	procédure : procédure table ci-dessus.	sans résultat pour entrée	adjoint à une table l'information donnée info, pour l'indicatif indic grâce à la procédure table. L'entrée i est sélectionnée grâce à la procédure filtre selon le processus suivant : (rappel de 1. 3.). - s'il n'existe aucun entier i tel que la rangée repère par table (indic, i) vérifie la procédure filtre, entier affecte info à table (indic, j) où j est le premier entier tel que table (indic, j) ne repère aucune rangée. - s'il existe un tel entier j et si i_0 est le premier d'entre eux, pour $i > i_0$ entier affecte à table (indic, i) la rangée repère par table (indic, i-1) et affecte info à table (indic, i_0).
ıtrée (table, indic, ıfo, filtre).	rangée de repères : indic.		
	rangée de repères : info.		
ıtrée 2	procédure : filtre à résultat booléen et à paramètre rangée de repères.	résultat la rangée de repères table (indic, i) sélectionnée (où se fait l'adjonction.)	
recherche	procédure : procédure table ci dessus.	la rangée de repères pointant vers l'information sélectionnée en Table.	recherche dans une table la rangée de repères table(indic, i) où i est le premier entier tel que la rangée repère par table (indic, i) vérifie la procédure filtre s'il existe, sinon appelle la procédure erreur pour le numéro 0 et le résultat est aléatoire (1. 3.).
echerche (table, ıdic, filtre, erreur)	rangée de repères : indic procédure : filtre à résultat booléen et à paramètres rangée de repères. procédure erreur à un paramètre élément.		
retour			la procédure f (filtre) appelant retour est paramètre d'une procédure S standard (entrée ou recherche). Retour commande le retour dans la procédure appelant S(entrée ou recherche) (§5. filtre 2).
etour ( )			

3) Manipulation d'octet, de bit de demi-mot.

Compact compact (e1, e2, e3, e4).	les repères ei	élément	l'élément résultat est composé dans l'ordre de gauche à droite des 4 octets de droite des 4 éléments ei.
decoup découp (i, e)	repère i entier	élément	l'élément résultat est le $i^{ème}$ demi-mot de l'élément e pour decoup (i vaut 1 le demi-mot gauche, i vaut 2 le demi-mot droite).
debit debit (i, e)	repère e		pour debit l'élément résultat est le i^e bit de l'élément e ($1 \leq i \leq 32$), les bits étant numérotés de gauche à droite.
decomp decomp (i, e)			pour decomp l'élément résultat est le i^e octet de l'élément e. ($1 \leq e \leq 4$), les octets étant numérotés de gauche à droite.
octet octet (i, mot, x)	repère i		octet met l'octet droite de l'élément x à la place du i^e ($1 \leq i \leq 4$) octet de la valeur répérée par mot.
bit bit (i, mot, x)	repère mot		bit met le 32e bit de l'élément x à la place du i ($1 \leq i \leq 32$) bit de la valeur repérée par mot.
demi-mot demi-mot (i, mot, x)	repère x		demi-mot met le 2e demi-mot de x à la place du i^e demi-mot de la valeur repérée par mot.

C) Conversion

elt rep elt rep (e)	repère e	repère	l'élément est considéré comme une adresse qui est le repère résultat.
rep elt rep elt (r)	repère r	élément	le repère r est une adresse et cette adresse est l'élément résultat.
bd bd (i)	repère i	rangée de repères	i est un entier décimal bd le convertit sous la forme d'une chaîne de 8 caractères complétés à gauche par des zéros ; elle est repérée par la rangée de 2 repères résultat.
dec dec (r ; n)	r rangée de repères n repère	élément	les n premiers caractères rangés dans les repères de la rangée r sont des chiffres décimaux désignant un entier et dec donne pour résultat l'entier correspond

3) Compilation.

analyse analyse (rp, ramif)	repère entier numéro d'une règle d'une grammaire dont le premier membre est l'axiome rangée de repères à laquelle analyse affecte la ramification sous forme post fixée sur les numéros de règle de grammaire.		analyseur de Face (I 1),(I 2) (I 3) (I 4. 3)
appl appl (ramif, rp)	rangée de repères résultat ci dessus de l'analyse rangée de procédures		programme de commande de Face (I 1) (I 2) (I 3) (I 4 6)
Fada Fada (rg, clé)	rangée de repères contenant les tables définissant une grammaire repère désignant la clé de la description de la grammaire en question dans le fichier des grammaires		préanalyseur cherche une méthode d'analyse déterministe et éventuellement et raye la grammaire dans le fichier des grammaires selon la clé donnée en argument.

E) Génération

ordre ordre (co, reg, ai, m,adr, ri).	les paramètres sont des repères désignant les codes opération registre, marque d'adressage indirect, booléen de conversion d'adresse, adresse et index de l'ordre à générer.		génère un ordre du programme objet en plaçant dans un tampon de sortie la zone étiquette désignée par la variable globale éti, la zone commande grâce au opération et registre, la zone adresse après conversion éventuelle binaire décimale de l'adresse adr pour fermer l'ordre Symbol. l'ordre généré est alors placé dans le fichier SO en séquence et écrit simultanèment sur l'imprimante.
écrirang écrirang (r, n)	r rangée de repères repérant les constantes Symbol. n nombre de pseudo-instruction data à générer. Remarque : cette procédure pourrait être écrite en Face 0 grâce à une procédure récursive utilisant la procédure glob .		génère n directives data avec en zone adresse les valeurs des repères (convertis binaire décimaux) r[1] ... r[n].
glob glob (co, r, i)	co repère désignant le code de la directive à générer r rangée de repères repérant i caractères de la constante symbol i nombre de caractères de la constante symbol.		génère la directive symbol où les i caractères de la zone commande sont placés dans les octets successifs de r[1] r[2] .... de gauche à droite.
erreur erreur (n)	n élément numère l'erreur décelée.		imprime sur l'imprimante le numéro de l'erreur ainsi que les compteurs de lignes et de caractères du texte source.
arrt	sans paramètres sans résultat		fermeture du fichier programme objet sur disque créé par les procédures ci-dessus)
init	sans paramètres sans résultat		génère les ordres de début du programme objet et ouvre le fichier

REFERENCES

1 - R.A. BROOKER, O.MORRIS

A general translation program for phrase structure languages.Journal of the ACM 9 (1962) pl-16.

2 - R.A. BROOKER, I.R. MACCALLUM, D. MORRIS, J.S. ROHL.

The COMPILER-COMPILER Ann. Rev. in AUTO. PROG.(1963) Vol 3, p 229.

3 - J. FELDMANN

A formal semantics for computer languages and its application in a compiler-compiler. CACM, Vol 10(1966), p 316.

4 - J. FELDMANN, D.GRIES.

Translator writing systems.
Communications ACM Vol 11, Num 2 (Février 1968) p 77-113.

5 - P. GILBERT, W.G. MC LELLAN.

Compiler generation using formal specification of procedure-oriented and machine languages.
Proc. AFIPS (1967) Vol 30 p 447-455.

6 - S. GINSBURG.

The mathematical theory of context-free languages.
MC Graw-Hill, New York (1966).

7 - E.T. IRONS.

The structure and use of the syntax directed compiler.
Ann. Rev. in Auto. Prog (1963) Vol 3 p 207.

8 - R. KHALIL.

Essai de formalisation de la description des compilateurs.
Application à Algol 60.
Thèse Docteur Ingénieur, NANCY (1970).

9 - J. MAROLDT.

Thèse de Spécialité à paraître.
Faculté des Sciences - NANCY.

10 - H.H. METCALFE.

A parametrised compiler based on mechanical linguistics.
Annual review in automatic programming (1964) Vol. 4 p 125-165.

11 - P. NAUR.

Revised report on the algorithmc language. Algol 60.
Communications ACM (Janvier 1963).

Sc N 72/99 (B)

12 - C. PAIR et A. QUERE.

Définition et études des bilangages réguliers.
Information and Control, 13 (1968), p565-593.

13 - C. PAIR.

Sur des notions algébriques liées à l'analyse syntaxique
Exposé fait au Centre d'Automatique de l'Ecole des Mines, Fontainebl
(1969) soumis à la Revue Française d'Informatique et de Recherche
Opérationnelle.

14 - C. PAIR.

Cours de l'Ecole d'Eté d'Informatique, Compilation, NEUFCHATEL (1

15 - E.C. RUSSELL

Meta 5 Manual.
Version for XDS SIGMA7 (or CII 10070) UNDERBATCH PROCESSING
MONITOR IRIA (Octobre 1970).

16 - J.C. REYNOLDS.

Cogent Manuel Argonne National Laboratory
Argonne, Illinois (Mars 1965) ANL - 70 22.

17 - F.W. SCHNEIDER et G.D. JOHNSON.

Meta 3 : A syntax directed compiler writing compiler to generate effic
code. Proc ACM 19 ème Nat. Conf (1964) p D1. 5-1.

18 - D.V. SCHORSE.

META II : A syntax oriented compiler writing language
Proc ACM 19 th Natl. Conf (1964). P D1 - 3.

19 - A. VAN WIJNGAARDEN (ed)

Report on the algorithmic language Algol 68 - Mathematisch Centrum
Amsterdam (1969).

20 - SYMBOL METASYMBOL.
Manuel d'utilisation CII
C 900 952.

ANNEXE 2

GRAMMAIRE FACE

<vide> ::=

Symboles de base :

Lettres :

<lettre> ::= A/131/B/132/.....Z/156/

<ℓlex> ::= A/157/B/158/ Z/182/

<ℓgen> ::= A/183/B/184/ Z/208/

Chiffres :

<choctal> ::= 0/209/1/210/2/211/ 9/217/

<chdécimal> ::= <choctal> // 8/217/9/218/

<chhexadécimal> ::= <chdécimal> // A/219/B/220/C/221/D/222/E/223/F/224/

<chgen> ::= 0/225/1/226/2/227/ 9/234/

<chiffre> ::= 0/235/1/236/2/237/ 9/244/

Symboles :

<souvrir> ::= (/245/

<sfermer> ::=)/245/

<crochet ouvert> ::= [/247/

<crochet fermé> ::=]/247/

<deux points> ::= :/245/

<virgule> ::= ,/245/

<point virgule> ::= ;/245/

<slash> ::= / /245/

<égal> ::= =/245/

<deux points égal> ::= :=/247/

<selt> ::= ELT/246/

<srep> ::= REP, /246/

<srangée> ::= RANGEE /248/

<sproc> ::= PROC /249/

<scode> ::= CODE /249/

<sres> ::= # /245/

Caractères :

<caractère> ::= <lettre> // <chiffre> // <caractère spécial> //

<carlex> ::= <ℓlex> // <chdécimal> // <carlex spécial> //

<cargen> ::= <ℓgen> // <chgen> // <cargen spécial> //

<caractère spécial> ::=␣/251/"/250/$/252/ */253/ / /254/+/255/-/256/>/257
</258/ = /259/!/260/¬/261/&/262/ (/263/)/264/:/265/, /266/;/267/.
"/269/%/270/# /271/@/272/?/273/

<carlex spécial> ::=␣/274/)(/118/$/275/ ....?/296/

<cargen spécial> ::=␣/297/)(/128/$/298/....?/ 318/

Caractère non significatif et règle de changement de ligne :

::= / /$/99

Chaînes :

<chaîne> ::= <caractère> /2/ <chaîne> <caractère> /3/

<chaîne lex> ::= <carlex> /5/ <chaînelex> <carlex> /6/

<chaîne fermée> ::= '<chaîne>' /4/

<chaîne lex fermée> ::= '<chaîne lex>' /7/

Entiers :

<entier décimal> ::= <chdécimal> /8/ <entier décimal> <chdécimal> /

<décimal> ::= <chgen> /10/ <décimal> <chgen> /11/

<décim> ::= <chiffre> /33/ <décim> <chiffre> /34/

<entier octal> ::= $ <choctal> /124/ <entier octal> <choctal> /125/

<entier hexadécimal> ::= @ <ch hexadécimal> /122/

<entier hexadécimal> <chhexadécimal> /123/

<entier> ::= <entier décimal> /12/ <entier octal> /126/

<entier hexadécimal> /127/

Eléments :

<élément> ::= <entier> / / <chaine lex fermé> / 13/NIL/1/

Identificateurs :

<ident> ::= <idgen> /21/ <sres> <décimal> /20/

<id> ::= <lettre> /14/ <id> <chiffre> /15/ <id> <lettre> /15/

<idlex> ::= <ℓ lex> /16/ <idlex> <chdécimal> /17/ <idlex> <ℓ lex> /17

<idgen> ::= <ℓgen> /18/ <idgen> <ℓ gen> /19/ <idgen> <chgen> /19/

Définition des grammaires :

Terminal, non terminal, premier membre, numéro.

<terminal> ::= <chaîne fermée> /23/

<non terminal> ::= <idgen> /24/

<premier membre> ::= <id> /25/

<numéro> ::= <vide> /29/ <décim> /28/

Règle de grammaire.

<règle> ::= <premier membre> <deux points> <second membre>

<point virgule> / / <deux points> <membre deux> <point virgule> / /

<second membre> ::= <membre> <slash> <numéro> /30/

<slash> <numéro> /30/

<second membre> <slash> <membre> <slash>

<numéro> /30/ <second membre> <slash> <slash>

<numéro> /30/

<membre> ::= <terminal> /27/ <non terminal> /27/ <membre> <virgule>

<terminal> /27/ <membre> <virgule> <non terminal> /27/

<membre deux> ::= ' <cargen> ' <slash> <numéro> /26/

$ <slash> <numéro> /130.

<membre deux> <slash> ' <cargen> ' <slash>

<numéro> /26/

<membre deux> <slash> $ <slash> <numéro> /130/

Grammaire :

<gram> ::= GRAM/ 22

<grammaire> ::= <gram> <suite de règles> /31/

<suite de règles> ::= <règle> <point virgule> / / <suite de règles> <règle>

<point virgule> / /

<définition des grammaires> ::= <suite de grammaires> /32/

<suite de grammaires> ::= <vide> / / <suite de grammaires> <grammaire> / /

Les déclarations globales :

Déclaration globale d'élément de repère :

<déclaration d'élément> ::= <selt> <idlex> <égal> <élément> /35/

<déclaration d'éléments> <virgule> <idlex> <égal> <élément> /35/

<déclaration de repères> ::= <srep> <idlex> /36/

<srep> <idlex> <deux points égal> <élément> /37/

<déclaration de repères> <virgule> <idlex> /36/

<déclaration de repères> <virgule> <idlex> <deux points égal>

<élément> /37/

Déclaration globale de rangée d'éléments ou de repères :

< fin de déclaration de rangée d'éléments > ::= < elt >< idlex >< égal >< souvrir >
< liste d'éléments >< sfermer > /38/< fin de déclaration de rangée
d'éléments >< seprangée >< idlex >< égal >
< souvrir >< liste d'éléments >< sfermer > /38/

< fin de déclaration de rangée de repères > ::= < rep >< idlex > /43/
< rep >< idlex >< deux points égal >< souvrir >< liste d'éléments >
< sfermer > /44/< fin de déclaration de rangée de repères >
< sep rangée >< idlex > /43/ < fin de déclaration de
rangée de repères >< seprangée >< idlex >< deux points
égal >< souvrir >< liste d'éléments >< sfermer > /44/

< rep > ::= < rep > /39/
< sep rangée > ::= < virgule > /39/
< elt > ::= < selt > /39/
< déclareur de rangée > ::= < srangée >< souvrir >< décim >< sfermer > /40/
< liste d'éléments > ::= < élément > /41/ < liste d'éléments >< virgule >
< élément > /42/

Déclaration de procédures, de procédures écrites en code :

< dec proc > ::= < tête proc >< égal >< corps proc > /49/
< tête proc > ::= < sproc >< id > /47/ < sproc >< résultat >< id > /48/
< résultat > ::= < selt > /45/ < srep > /46/
< corps proc > ::= < souvrir >< liste de paramètres formels >< sfermer >
< expression > / < souvrir >< sfermer >< expression > //
< liste de paramètres formels > ::= < déclareur >< liste d'idlex > //
< déclareur >< liste d'idlex >< point virgule >< liste de paramètres
formels > //
< liste d'idlex > ::= < idlex > /90/ < idlex >< virgule >< liste d'idlex > /50/
< déclareur > ::= < selt > /51/ < srep > /52/ < srangée >< selt > /53/
< srangée >< srep > /54/
< decproccode > ::= < tête code >< égal >< corps code > /58/
< tête code > ::= < scode >< idlex > /56/ < scode >< rescode >< idlex > /57/
< rescode > ::= < résultat > // < srangée >< srep > /55/
< corpscode > ::= < souvrir >< liste des paramètres fictifs >< sfermer > //
< souvrir >< sfermer > //
< liste des paramètres fictifs > ::= < decode >< liste d'idfictifs > //
< decode >< liste d'idfictifs >< point virgule >< liste des paramètres
fictifs > //
< liste d'idfictifs > ::= < idlex > /59/ < idlex >< virgule >
< liste d'idlex > /59/
< decode > ::= < selt > /60/ < srep > /61/ < srangée >< selt > /62/ < srangée >< srep >
/63/ < sproc > /64/

Rangées de procédures :

< fin de rangée de procédures > ::= < début de rangée de proc >< liste de proc / 71
< début de rangée de proc > ::= < sproc >< idlex >< point virgule > /65/

< liste de proc > ::= < debproc >< désignation > //
< liste de proc >< debproc >< désignation > //

< debproc > ::= < idlex >< crochet ouvert >< décimal >< crochet fermé >< égal > /66/

< désignation > ::= < idgen >< crochet ouvert >< décimal >< crochet fermé >
< point virgule > /69/
< entrée >< expression >< point virgule > /70/

< entrée > ::= < souvrir >< décimal >< virgule >< décimal >< sfermer > /67/
< souvrir >< décimal >< virgule >< srangée >< virgule >
< décimal >< sfermer > /68/

Liste de déclarations :

< liste de déclarations > ::= < vide > /119/ < suite de déclarations > /119/

< suite de déclarations > ::= < déclaration >< point virgule > //
< suite de déclarations >< déclaration >
< point virgule > //
< déclaration > ::= < déclaration de repères > // < déclaration d'éléments > //
< déclareur de rangée >< fin de déclaration de
rangée de repères > / 121/ < déclareur de rangée >
< fin de déclaration de rangée d'élément > /121/
< déclaration de procédure > //< décproccode > //< déclareur de
rangée > < fin de déclaration de rangée de procédures > //

Déclarations Locales :

Début de bloc:

< premier repère > ::= < srep > /113/

< première rangée > ::= < srangée > /113/

Déclaration locale de repère :

< déclaration locale de repère > ::= < srep >< fin de decloc de repères > //

< fin de bloc de repères > ::= < idgen >< valuation //
< fin de decloc de repères >< virgule >< idgen >< valuation > //

< valuation > ::= < vide > /103/ < égal >< secondaire > /104/
< deux points égal >< expression simple > /105/

Déclaration locale de rangée de repères :

<déclaration locale de rangée de repères> ::= <srangée>
<fin de décloc de rangée> /76/

<fin decloc de rangée> ::= <déclareur local de rangée><idgen>
<valrang> //<findecloc de rangée><virgule>
<idgen><valrang> //

<déclareur local de rangée> ::= <souvrir><décimal><sfermer> /107/

<valrang> ::= <vide> /108/ <égal><secondaire> /109/
<sinit><souvrir><liste d'expressions><sfermer><srep> /107/

<sinit> ::= <deux points égal> /111/

<liste d'expressions> ::= <expression simple> /106/
<liste d'expressions><virgule><expression simple> /106

Liste de déclarations locales :

<liste de déclarations locales> ::= <vide> //
<liste de déclarations locales><point virgule>
<déclaration locale de repères> //
<liste de déclarations locales><point virgule>
<déclaration locale de rangée de repères> //

Expression.

Appel de procédures :

<début d'appel> ::= souvrir /72/

<fin d'appel> ::= <début d'appel><liste de paramètres effectifs><sfermer> /74/
<début d'appel><sfermer> /74/

<liste de paramètres effectifs> ::= <expression> /73/
<liste de paramètres effectifs><point virgule><expression> /73/

Secondaire :

<secondaire> ::= <ident> /75/ <ident><fin d'appel> //
<ident><crochet ouvert><expression><crochet fermé> /77/
<élément> /78/ <expression fermée> //

Expression simple :

<expression simple> ::= <terme bool> // <expression simple><ou>
<terme bool> /86/

<ou> ::= ∨ /98/

<terme bool> ::= <fact bool> // <terme bool><et><fact bool> /97/

<et> ::= ∧ /96/

<fact bool> ::= <prim bool> // <non><prim bool> /95/

<non> ::= ¬ /94/

<prim bool> ::= <expra> // <prim bool><op><expra> /93/

<opr> ::= > /87/ ≥ /88/ < /89/ ≤ /90/ = /91/ ≠ /92/

<expra> ::= <terme> // <expra><op><terme> /86/ <opa><terme> /84/

<op> ::= <opa> /85/

<opa> ::= + /82/ - /83/

<terme> ::= <secondaire> // <terme><opm><secondaire> /79/

<opm> ::= * /80/ / /81/.

Expression :

<expression> ::= <expression simple> // <expression simple><si>
<expression simple><sinon><expression> /102/

<si> ::= deux points /100/

<sinon> ::= <virgule> /101/

Expression fermée :

<expression fermée> ::=
<souvrir><premier repère><fin decloc de repère><liste de déclarations locales>
<point virgule><fin d'expression fermée><sfermer> /116/
<souvrir><premier rangée><fin decloc de rangée><liste de déclarations locales>
<pointgule><fin d'expression fermée><sfermer> /116/
<souvrir><fin d'expression fermée><sfermer> //

<pointgule> ::= <point virgule> /117/

<fin d'expression fermée> ::= <vide> /115/ <suite> //

< suite > ::= < expression > //
< expression > < fin d'affectation > //
< expression > < sepins > < suite > //
< expression > < fin d'affectation > < sepins > < suite > //

< sepins > ::= < point virgule > /114/

< fin d'affectation > ::= < deux points égal > < expression > /112/

Programme :

< programme > ::= < définition des grammaires > < liste de déclarations >
< expression fermée > /120/

BIBIOTHEQUE SCIENCES NANCY 1
D 0952029676

NOM DE L'ETUDIANT : BELLEGARDE Françoise

Nature de la thèse : Doctorat de Spécialité en Mathématiques Appliquées

Vu, Approuvé

et permis d'imprimer

NANCY, le 17 oct 72

Le Président du Conseil de l'Université de NANCY

J.R. HELLUY