

ALGORITHMES D'ANALYSE SYNTAXIQUE POUR LANGAGES "CONTEXT-FREE" (1)

M. BRASSEUR et J. COHEN (2)

Résumé. — La première partie de cet article (parue dans le numéro précédent) comportait des généralités : aspects théoriques du problème, différents types d'analyse, aperçu sur les travaux existant en analyse syntaxique. L'algorithme d'analyse descendante était ensuite présenté. Cette seconde partie étudie les algorithmes d'analyse ascendante et d'analyse multiple ; elle donne, enfin, les conclusions.

DEUXIÈME PARTIE

III. — ANALYSE ASCENDANTE

Pour cet algorithme, nous exprimerons la grammaire d'une manière un peu différente. Les sujets des règles seront placés en partie droite, le symbole \Rightarrow remplaçant le symbole \rightarrow ; la grammaire est dite sous forme de reconnaissance. D'une manière analogue à celle utilisée au chapitre précédent, nous grouperons les productions en un ensemble d'arbres, ainsi que le montre l'exemple suivant.

Soit la grammaire G_7 (3) :

| | | |
|---------------------|---------------|------------------------------------|
| $aS \Rightarrow S$ | | $aS \Rightarrow S$ |
| $aAb \Rightarrow S$ | Elle s'écrira | \downarrow $Ab \Rightarrow S$ |
| $b \Rightarrow A$ | en 2 arbres : | \downarrow $a \Rightarrow A$ |
| $aAa \Rightarrow A$ | | $b \Rightarrow A$ |

1^{er} arbre à 2 branches, la deuxième ayant 2 sous-branches
 2^e arbre

(1) Manuscrit reçu le 27 janvier 1965.

(2) Institut de Mathématiques Appliquées de Grenoble, groupe Algoal.

(3) Le langage décrit par cette grammaire est composé des chaînes $a^{m+n}ba^n$ avec $m > 1$ et $n > 0$ (a^n représente la chaîne formée de a concaténé $n - 1$ fois à lui-même : a^3 représente aaa).

Dans le calculateur, les données correspondantes sont stockées en trois tableaux à une dimension, *ARBRE*, *PRÉCÉDANT*, *ALTERNANT* d'indice *J*, dont les significations sont analogues à celles utilisées au chapitre II.

| J | PRÉCÉDANT | ARBRE | ALTERNANT |
|----|-----------|-------|-----------|
| 1 | * * | a | * |
| 2 | 1 | S | 5 |
| 3 | 2 | S | * |
| 4 | * | * | * |
| 5 | 1 | A | * |
| 6 | 5 | b | 9 |
| 7 | 6 | S | * |
| 8 | * | * | * |
| 9 | 5 | a | * |
| 10 | 9 | A | * |
| 11 | * | * | * |
| 12 | * * | b | * |
| 13 | 12 | A | * |
| 14 | * | * | * |

Dans *ARBRE*, la marque "*" suit le sujet des règles. Le premier symbole de chacune d'elles constitue la racine de l'arbre.

Pour les besoins de l'algorithme, les données précédentes relatives à la grammaire doivent être complétées par les tableaux *TERMINAL*, *RACINE* qui sont construits comme au chapitre II; nous donnons, ci-dessous, ceux qui correspondent à la grammaire G_7 (1).

| SYMBOLE | TERMINAL | RACINE |
|---------|----------|--------|
| a | vrai | 1 |
| b | vrai | 12 |
| A | faux | * |
| S | faux | * |

Nous introduisons également le tableau *SUCCESSEUR* qui indique si l'application d'une règle commençant par le symbole actuellement considéré peut conduire au but cherché.

L'algorithme d'analyse descendante gagnerait en efficacité par sa présence, mais l'exposé perdrait en clarté. Son absence augmenterait

(1) Un élément du vocabulaire non terminal peut aussi être racine d'un arbre.

considérablement le nombre d'essais infructueux dans l'analyse ascendante, qui serait alors pratiquement irréalisable.

Le tableau *SUCCESSEUR* est une matrice booléenne carrée, de dimension égale au nombre de symboles du vocabulaire. *SUCCESSEUR* [P, Q] a la valeur **vrai** si l'on peut trouver une suite $\alpha_1, \alpha_2, \dots, \alpha_i, \dots, \alpha_n$ telle que :

- 1) $\alpha_1 = P; \alpha_n = Q; \alpha_i \in V_N \cup V_T$; pour tout $1 < i \leq n, \alpha_i \in V_N$
- 2) pour tout $1 \leq i < n$, il existe une règle $\alpha_i \omega \Rightarrow \alpha_{i+1}$, ω étant une chaîne sur $V_T \cup V_N \cup \Lambda$

De cette définition, il résulte la propriété transitive suivante :
si *SUCCESSEUR* [P_1, P_2] et *SUCCESSEUR* [P_2, P_3] sont tous deux **vrai**, alors *SUCCESSEUR* [P_1, P_3] est **vrai**.

Nous donnons ci-dessous la matrice *SUCCESSEUR* correspondant à la grammaire G_7 .

| | <i>a</i> | <i>b</i> | <i>A</i> | <i>S</i> |
|----------|----------|----------|----------|----------|
| <i>a</i> | faux | faux | vrai | vrai |
| <i>b</i> | faux | faux | vrai | faux |
| <i>A</i> | faux | faux | faux | faux |
| <i>S</i> | faux | faux | faux | faux |

Les tableaux mentionnés ici, à l'exception du dernier, peuvent s'obtenir facilement à partir de la description de la grammaire donnée sous forme normale de Backus. Nous avons programmé l'algorithme correspondant que nous ne jugeons pas nécessaire de présenter ici.

La détermination de la matrice *SUCCESSEUR* est plus complexe. Elle se fait à partir d'une matrice M contenant les suites α_1, α_2 , obtenues directement à partir des règles de la grammaire.

$$SUCCESSEUR = M_n = \bigcup_{K=1}^n M^K$$

avec

$$M^1 = M \quad \text{et} \quad M^{K+1} = M^K \cap M \quad (1)$$

(1) Par définition : si $A \cap B = C$, C est tel que $c_{ij} = \bigcup_i a_{ii} \cap b_{ij}$

si $A \cup B = D$, D est tel que $d_{ij} = a_{ij} \cup b_{ij}$

$$\bigcup_{K=1}^n M^K = M^1 \cup M^2 \cup \dots \cup M^K \cup \dots \cup M^n.$$

M^K donne les suites $\alpha_1, \dots, \alpha_K$.

Warshall a montré qu'il existait une manière plus simple et équivalente de former *SUCCESSEUR*. Dans la définition ci-dessus, la valeur de n est à déterminer.

Considérons la suite

$$M_1, M_2 = M^1 \cup M^2, \dots, M_K = M^1 \cup M^2 \cup \dots \cup M^K, M_{K+1}, \dots, M_n.$$

Pour une certaine valeur de K , on a : $M_K \equiv M_{K+1}$. On prendra $M_n = M_K$

Warshall (cf. référence 41) a défini l'algorithme suivant qui permet de déterminer M_n à partir de M_1 déjà appelé *SUCCESSEUR*. N est la dimension de la matrice.

```

pour J := 1 pas 1 jusqu'à N faire
  pour I := 1 pas 1 jusqu'à N faire
    début
      si SUCCESSEUR [I, J] alors
        début
          pour L := 1 pas 1 jusqu'à N faire
            SUCCESSEUR [I, L] := SUCCESSEUR [I, L] V
            SUCCESSEUR [J, L]
          fin
        fin
      fin
    fin ;

```

La matrice *SUCCESSEUR*, stockée comme ci-dessus, exige un grand nombre de mémoires. Comme elle est creuse, en général, il est intéressant de la ranger sous forme de liste. Ceci est indispensable quand l'encombrement de *SUCCESSEUR* dépasse les disponibilités du calculateur en mémoires rapides. Une autre solution serait d'utiliser chaque position binaire d'un mot machine pour indiquer le caractère *vrai* ou *faux* d'un élément de *SUCCESSEUR*. C'est ainsi que fit Irons qui utilisa cette matrice dans son compilateur *Psyco*.

La chaîne que nous voulons analyser est stockée en machine sous forme d'un tableau à une dimension *CHAINE* de même signification qu'au chapitre II.

L'algorithme que nous allons décrire maintenant a pour base celui d'Irons dans *Psyco*. Il a été modifié selon les suggestions d'Unger afin de revenir en arrière dans les cas notés au chapitre I (cf. aperçu sur les travaux existant en analyse syntaxique).

Les piles *BUT* et *TROUVE* ainsi que les procédures *EMPILERBUT* et *EMPILERTRUVE* sont utilisées comme au chapitre II.

Voyons sur un exemple quel principe guide l'algorithme.

SEUR formés à partir de G_6 . La figure 13 présente la chaîne d'entrée identique à celle analysée au chapitre II. Elle donne également la structure de cette chaîne construite à partir du contenu de la pile *TROUVE* présenté sur la partie gauche de la figure.

Les différences constatées entre cette structure et celle obtenue par l'algorithme d'analyse descendante proviennent de ce que les règles récursives à gauche sont traitées différemment dans l'une et l'autre analyse.

Algorithme d'analyse ascendante

```

INITIALISATION :  $K := L := I := J := 1$  ;
                  EMPILERBUT (AXIOME, I, MARQUE) ;
                  si  $\neg$  SUCCESSEUR [CHAINE [I], AXIOME] alors
                    allera ERREUR ;

RENTREEARBRE :  J := RACINE [CHAINE [I]] ;

VERIFSITERM :   J := J + 1 ; I := I + 1 ;
                  si TERMINAL [ARBRE [J]] alors
                    allera si CHAINE [I] = ARBRE [J] alors
                      VERIFSITERM sinon RECULER ;
                  si ARBRE [J + 1] = ETOILE alors
                    début
                      I := I - 1 ;
                      si ARBRE [J] = BUT [K - 3] alors
                        début
                          EMPILERTROUVE (BUT [K - 3], BUT
                            [K - 2], BUT [K - 1], I, J) ;
                          si BUT [K - 3] = AXIOME alors allera
                            TERMINE ;
                        ESSAYEREXPANS : si SUCCESSEUR [BUT [K - 3],
                            BUT [K - 3]] alors
                          début
                            J := RACINE [BUT
                              [K - 3]] ;
                            allera VERIFSITERM
                          fin ;
                    ENLEVERBUT : J := BUT [K - 1] ;
                               K := K - 3 ;
                               allera VERIFSITERM
                    fin ;
                  si  $\neg$  SUCCESSEUR [ARBRE [J], BUT [K - 3]]

```

```

    alors allera RECULER2 ;
    EMPILETROUVE (ARBRE [J], BUT [K - 2],
    0, I, J) ;
    J := RACINE [ARBRE [J]] ;
    allera VERIFSITERM
  fin ;
  si ¬ SUCESSEUR [CHAINE [I], ARBRE [J]] alors
  allera RECULER ;
  EMPILEBUT (ARBRE [J], I, J) ;
  allera RENTREEARBRE ;
RECULER :
  I := I - 1 ;
RECULER2 :
  si ALTERNANT [J] ≠ ETOILE alors
  début
    J := ALTERNANT [J] - 1 ;
    allera VERIFSITERM
  fin ;
  J := PRECEDANT [J] ;
  si ¬ TERMINAL [ARBRE [J]] ∧ PRECEDANT [J]
  ≠ DEUXETOILES alors
  début
    EMPILEBUT (TROUVE [L - 5], TROUVE
    [L - 4], TROUVE [L - 3]) ;
    ENLEVERTROUVE : J := TROUVE [L - 1] ;
    L := L - 5 ;
    allera RECULER2
  fin ;
  si ¬ TERMINAL [ARBRE [J]] ∧ PRECEDANT [J]
  = DEUXETOILES alors
    allera si BUT [K - 3] = TROUVE [L - 5]
    alors ENLEVERBUT
    sinon ENLEVERTROUVE ;
  si PRECEDANT [J] ≠ DEUXETOILES alors allera
  RECULER ;
  J := BUT [K - 1] ;
  K := K - 3 ;
  allera si J = MARQUE alors ERREUR sinon
  RECULER ;

```

REMARQUE. — Les étiquettes *ERREUR* et *TERMINE* sont analogues à celles utilisées dans l'algorithme d'analyse descendante. *MARQUE* est une variable entière qui permet la détection d'erreur.

K et *L* sont les indices respectifs des piles *BUT* et *TROUVE*.

| J | PRÉCÉDANT | ARBRE | ALTERNANT |
|----|-----------|-------|-----------|
| 1 | ** | # | * |
| 2 | 1 | P | * |
| 3 | 2 | # | * |
| 4 | 3 | S | * |
| 5 | * | * | * |
| 6 | ** | P | * |
| 7 | 6 | s | * |
| 8 | 7 | A | * |
| 9 | 8 | P | * |
| 10 | * | * | * |
| 11 | ** | A | * |
| 12 | 11 | P | * |
| 13 | * | * | * |
| 14 | ** | v | * |
| 15 | 14 | e | 34 |
| 16 | 15 | E | * |
| 17 | 16 | A | * |
| 18 | * | * | * |
| 19 | ** | E | * |
| 20 | 19 | p | * |
| 21 | 20 | T | * |
| 22 | 21 | E | * |
| 23 | * | * | * |
| 24 | ** | T | * |
| 25 | 24 | E | 27 |
| 26 | * | * | * |
| 27 | 24 | m | * |
| 28 | 27 | F | * |
| 29 | 28 | T | * |
| 30 | * | * | * |
| 31 | ** | F | * |
| 32 | 31 | T | * |
| 33 | * | * | * |
| 34 | 14 | F | * |
| 35 | * | * | * |
| 36 | ** | n | * |
| 37 | 36 | F | * |
| 38 | * | * | * |
| 39 | ** | g | * |
| 40 | 39 | E | * |
| 41 | 40 | d | * |
| 42 | 41 | F | * |
| 43 | * | * | * |

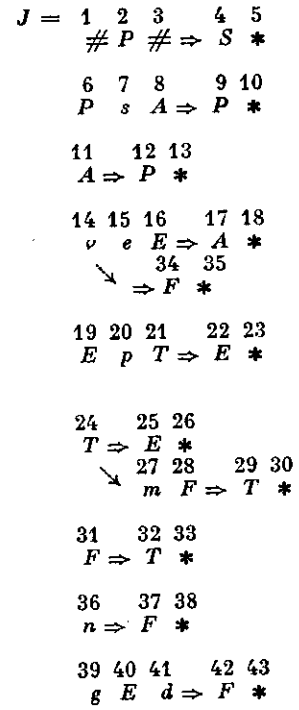


Figure 11

| SYMBOLE | TERMINAL | RACINE |
|---------|----------|--------|
| # | 1 | 1 |
| P | 0 | 6 |
| S | 0 | * |
| A | 0 | 11 |
| s | 1 | * |
| v | 1 | 14 |
| e | 1 | * |
| E | 0 | 19 |
| T | 0 | 24 |
| p | 1 | * |
| F | 0 | 31 |
| m | 1 | * |
| n | 1 | 36 |
| g | 1 | 39 |
| d | 1 | * |

1 signifie vrai 0 signifie faux

Figure 12 a

| # | P | S | A | s | v | e | E | T | p | F | m | n | g | d |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| # | | | | | | | | | | | | | | |
| P | | | | | | | | | | | | | | |
| S | | | | | | | | | | | | | | |
| A | | | | | | | | | | | | | | |
| s | | | | | | | | | | | | | | |
| v | | | | | | | | | | | | | | |
| e | | | | | | | | | | | | | | |
| E | | | | | | | | | | | | | | |
| T | | | | | | | | | | | | | | |
| p | | | | | | | | | | | | | | |
| F | | | | | | | | | | | | | | |
| m | | | | | | | | | | | | | | |
| n | | | | | | | | | | | | | | |
| g | | | | | | | | | | | | | | |
| d | | | | | | | | | | | | | | |

| signifie vrai blanc signifie faux

Figure 12 b

| L | TROUVE [L] (sujet) | TROUVE [L + 1] (I initial) | TROUVE [L + 3] (I final) | TROUVE [L + 4] (J) |
|----|-----------------------|-------------------------------|-----------------------------|-----------------------|
| 1 | F | 4 | 4 | 34 |
| 6 | T | 4 | 4 | 32 |
| 11 | F | 7 | 7 | 34 |
| 16 | T | 7 | 7 | 32 |
| 21 | E | 7 | 7 | 25 |
| 26 | F | 9 | 9 | 37 |
| 31 | T | 9 | 9 | 32 |
| 36 | E | 7 | 9 | 22 |
| 41 | F | 6 | 10 | 42 |
| 46 | T | 4 | 10 | 29 |
| 51 | E | 4 | 10 | 25 |
| 56 | A | 2 | 10 | 17 |
| 61 | P | 2 | 10 | 12 |
| 66 | F | 14 | 14 | 37 |
| 71 | T | 14 | 14 | 32 |
| 76 | E | 14 | 14 | 25 |
| 81 | A | 12 | 14 | 17 |
| 86 | P | 2 | 14 | 9 |
| 91 | S | 1 | 15 | 4 |

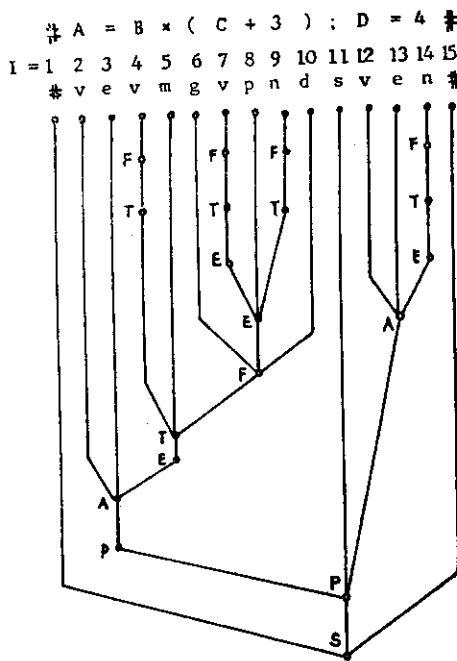


Figure 13

IV. — ANALYSE MULTIPLE

Rappelons que cet algorithme d'analyse ascendante permet d'obtenir toutes les structures associées à une même chaîne d'entrée. Il est utilisé au Centre de Traduction Automatique (C.E.T.A.) de Grenoble pour l'analyse des langages naturels (cf. référence 40).

Nous avons vu (chapitre I, aspects théoriques du problème) que la grammaire devait être préalablement transformée de façon à ce que toutes les règles soient de l'une des formes suivantes :

$$\Sigma_i \rightarrow \Sigma_j \Sigma_k \quad \text{règles de construction}$$

ou

$$\Sigma_e \rightarrow \rho_j \quad \text{règles lexicographiques}$$

Le processus de transformation indiqué au chapitre I pourrait être programmé afin d'être effectué automatiquement sur un ordinateur.

La grammaire obtenue est réécrite sous forme de grammaire de reconnaissance G'' et seules les règles de longueur deux sont stockées en machine dans un tableau M . La grammaire G'' pouvant contenir plusieurs règles ayant des sujets différents pour une même partie gauche, les éléments du tableau sont, en fait, des groupes de symboles ; pour permettre de repérer un des éléments du groupe, M aura une troisième dimension KM .

$M[U, V, KM]$ sera égal au sujet de la KM^e règle ayant comme partie droite l'élément U suivi de l'élément V .

Supposons que A, B et C aient les codes respectifs 1, 2 et 3 et que la grammaire G'' comprenne les règles : $BC \Rightarrow B$ et $BC \Rightarrow A$, alors

$$M[2, 3, 1] = 2 \quad (\text{c'est-à-dire } B)$$

$$M[2, 3, 2] = 1 \quad (\text{c'est-à-dire } A).$$

Les règles de la grammaire stockées dans le tableau M étant de longueur 2, la structure de la chaîne sera représentée par un arbre binaire, c'est-à-dire un arbre tel qu'à chaque nœud aboutissent deux branches.

Les règles de longueur 1 servent seulement à transformer la chaîne d'entrée. À chacun de ses éléments ρ , on fait correspondre un ou plusieurs éléments non terminaux selon qu'il existe, dans G'' , une ou plusieurs règles ayant ρ comme partie gauche. Par exemple si G'' contient les règles $\rho \Rightarrow \Sigma_i$ et $\rho \Rightarrow \Sigma_j$, à l'élément ρ de la chaîne d'entrée correspondront deux symboles Σ_i et Σ_j .

Comme zone de travail, l'algorithme utilise un tableau à trois dimensions $T[I, J, K]$ dont chaque élément représente un nœud de l'arbre binaire. Un terme tel que $T[I, J, K]$ indique le nœud auquel aboutit la K^e structure associée à une sous-chaîne commençant au 1^{er} groupe de symboles et ayant la longueur J .

Nous appellerons "niveau" la deuxième dimension J de ce tableau car tous les groupes de symboles construits à partir des sous-chaînes de même longueur se trouvent sur une même ligne de la matrice T .

Nous pouvons en effet considérer le tableau T comme une matrice dont I repère la colonne et J la ligne. Chaque élément de cette matrice est un groupe de symboles, l'indice K repérant le K^e d'entre eux.

L'algorithme utilise comme données le tableau M et la première ligne de la matrice T qui correspond à la chaîne d'entrée transformée, soit $T[I, 1, K]$, I variant de 1 à L .

Par niveaux successifs, l'algorithme calcule chacun des ensembles que constituent les éléments de la matrice T de la manière suivante : les divers $T[I, J, K]$ sont les sujets (s'ils existent) des règles ayant comme partie gauche :

$$\begin{array}{ll} \text{soit } T[I, 1, K1] & \text{suivi par } T[I + 1, J - 1, K2], \\ \text{soit } T[I, 2, K1] & \text{suivi par } T[I + 2, J - 2, K2], \\ \vdots & \vdots \\ \text{soit } T[I, L1, K1] & \text{suivi par } T[I + L1, J - L1, K2], \\ \vdots & \vdots \\ \text{soit } T[I, J - 1, K1] & \text{suivi par } T[I + J - 1, 1, K2], \end{array}$$

en donnant à $K1$ et $K2$ toutes les valeurs possibles.

Notons que la sous-chaîne de longueur J est bien composée de deux sous-chaînes dont la somme des longueurs est égale à J et que l'indice K se nomme $K1$ ou $K2$ quand il appartient à un élément situé dans la partie gauche d'une règle.

Pratiquement, dans le programme, l'épuisement des possibilités offertes par un ensemble $T [I, J, K]$ donné est repéré par l'existence d'une marque notée *ETOILE* à l'endroit correspondant. Ceci s'applique également à la matrice M . La figure 14 donne une représentation graphique du tableau T et de la façon dont on peut obtenir le groupe de symboles situés sur la ligne 5 et la colonne 2.

Le but de l'algorithme est la recherche des $T [1, L, K]$ égaux à S , c'est-à-dire des diverses structures associées à la chaîne de longueur L dont le premier élément est $I = 1$. Si $K = 0$, la chaîne d'entrée n'appartient pas au langage. Si $K = 1$, la chaîne possède une seule structure. Si $K > 1$, elle est ambiguë. Cette ambiguïté peut provenir de l'application des règles de construction, dans l'algorithme, ou simplement des règles lexicographiques.

Nous présentons, en fin de chapitre, le programme correspondant écrit en ALGOL tel qu'il a été testé sur le calculateur. Nous avons déjà donné la signification des tableaux (entiers) M et T et des entiers $KM, I, J, K1, K2, L1, L, ETOILE$. Une mémoire temporaire *ALPHA* permet de stocker des résultats intermédiaires.

Nous n'avons pas cherché en écrivant ce programme, à optimiser le nombre de mémoires utilisées ni le temps. La matrice M , qui est très creuse, pourrait être stockée sous forme de liste de même que la matrice T . L'ordre dans lequel sont calculés les ensembles $T [I, J, K]$ peut être modifié (cf. référence 40) de manière à éviter certaines recherches et à gagner du temps. Dans le cas où l'on a la répétition d'un même élément en deux ou plusieurs points du tableau T où I et J ont la même valeur, on pourrait continuer l'analyse avec un seul d'entre eux en gardant en mémoire les coordonnées des divers points qui représentent cet élément. Cela éviterait de propager les ambiguïtés, ce qui ralentit l'exécution de l'algorithme.

Le lecteur trouvera, figure 15, la transformation de la grammaire G_6 successivement en G_6^1, G_6^2 et G_6' .

Partant de G' , nous passons en G'' ; nous construisons la matrice M (cf. fig. 16) et transformons la chaîne d'entrée $\# A = B \times (C + 3) \#$ en la suivante :

| | | | | | | | | | | |
|-----|-------|-----|-------|-----|-----|-------|-----|-----|-----|-----|
| # | ν | e | ν | m | g | ν | p | n | d | # |
| B | C | D | C | I | K | C | H | E | J | B |
| | E | | E | | | E | | F | | |
| | F | | F | | | F | | T | | |
| | T | | T | | | T | | | | |

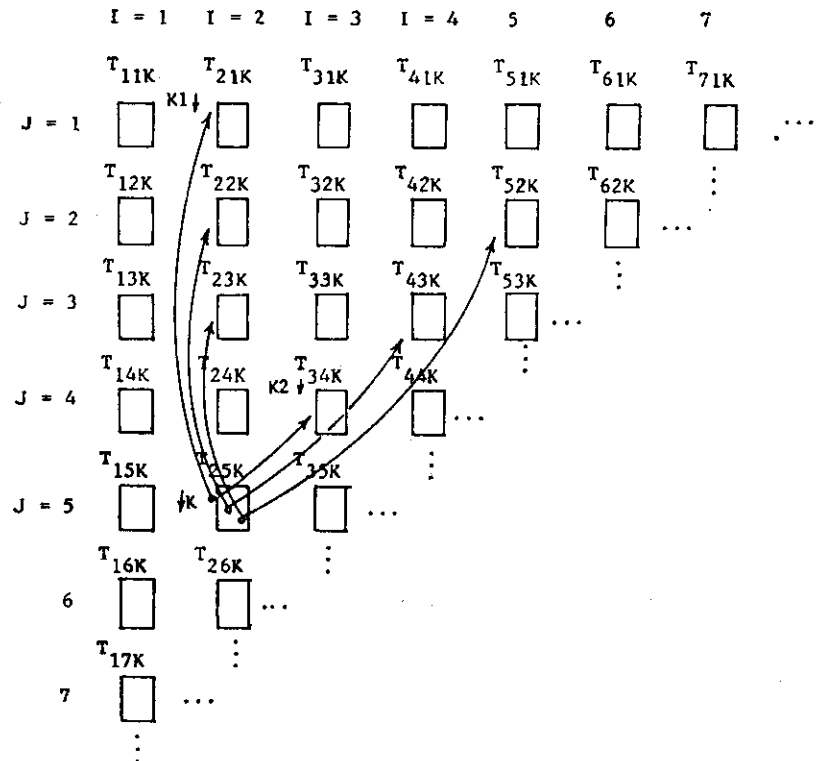


Figure 14

T_{25K} : ensemble des nœuds qui achèvent les structures associées aux sous-chaînes commençant par le deuxième élément ($I = 2$) et ayant la longueur 5 ($J = 5$). Cet ensemble est constitué par les sujets des règles ayant comme partie gauche :

- 1) T_{21K1} suivi par T_{34K2}
- 2) T_{22K1} suivi par T_{43K2}
- 3) T_{23K1} suivi par T_{52K2}
- 4) T_{24K1} suivi par T_{61K2}

La chaîne d'entrée est une partie de la chaîne analysée par les deux premiers algorithmes.

Le résultat final de l'analyse, qui donne une seule structure, est présenté figure 17. Nous n'indiquons pas les intermédiaires n'aboutissant pas à une structure pour la chaîne complète.

Si l'on veut reconstituer l'arbre binaire directement par l'algorithme, il faut stocker en machine les éléments permettant de retrouver les deux composantes de la règle dont $T[I, J, K]$ est sujet. On peut, par exemple,

utiliser un tableau TB à quatre dimensions dans lequel sont notées $L1, K1, K2$ correspondant à I, J, K donnés :

$$TB [I, J, K, 1] : = L1$$

$$TB [I, J, K, 2] : = K1$$

$$TB [I, J, K, 3] : = K2$$

Ces données suffisent à retrouver $T [I, L1, K1]$ et $T [I + L1, J - L1, K2]$, les deux nœuds qui aboutissent au nœud $T [I, J, K]$.

| G_6 | G_6^1 | G_6^2 | G'_6 |
|-------------------------|-------------------------|--|--|
| $S \rightarrow \# P \#$ | $S \rightarrow \# P \#$ | $\left\{ \begin{array}{l} S \rightarrow S_1 B \\ S_1 \rightarrow \# P \\ B \rightarrow \# \end{array} \right.$ | $\left\{ \begin{array}{l} S \rightarrow S_1 B \\ S_1 \rightarrow B P \\ P \rightarrow S_3 A \\ S_3 \rightarrow P G \end{array} \right. \quad B \rightarrow \#$ |
| $P \rightarrow P s A$ | $P \rightarrow P s A$ | $\left\{ \begin{array}{l} P \rightarrow S_3 A \\ S_3 \rightarrow P s \end{array} \right.$ | $\left\{ \begin{array}{l} P \rightarrow S_2 E \\ S_2 \rightarrow C D \end{array} \right. \quad G \rightarrow s$ |
| $P \rightarrow A$ | $P \rightarrow v e E$ | $\left\{ \begin{array}{l} P \rightarrow S_2 E \\ S_2 \rightarrow v e \end{array} \right.$ | $\left\{ \begin{array}{l} A \rightarrow S_2 E \\ E \rightarrow S_4 T \\ S_4 \rightarrow E H \end{array} \right. \quad C \rightarrow v$ |
| $A \rightarrow v e E$ | $A \rightarrow v e E$ | $\left\{ \begin{array}{l} E \rightarrow S_4 T \\ S_4 \rightarrow E p \end{array} \right.$ | $\left\{ \begin{array}{l} E \rightarrow S_5 F \\ S_5 \rightarrow T I \end{array} \right. \quad D \rightarrow e$ |
| $E \rightarrow E p T$ | $E \rightarrow E p T$ | $\left\{ \begin{array}{l} E \rightarrow S_5 F \\ S_5 \rightarrow T m \end{array} \right.$ | $\left\{ \begin{array}{l} E \leftarrow S_6 J \\ S_6 \rightarrow K E \end{array} \right. \quad I \rightarrow m$ |
| $E \rightarrow T$ | $E \rightarrow T m F$ | $\left\{ \begin{array}{l} E \rightarrow v \\ E \rightarrow n \end{array} \right.$ | $\left\{ \begin{array}{l} T \rightarrow S_5 F \\ T \rightarrow S_6 J \end{array} \right. \quad E \rightarrow v$ |
| $T \rightarrow T m F$ | $E \rightarrow n$ | $\left\{ \begin{array}{l} T \rightarrow v \\ T \rightarrow n \end{array} \right.$ | $\left\{ \begin{array}{l} E \rightarrow n \\ E \rightarrow n \end{array} \right.$ |
| $T \rightarrow F$ | $E \rightarrow g E d$ | $\left\{ \begin{array}{l} E \rightarrow S_6 J \\ S_6 \rightarrow g E \\ J \rightarrow d \end{array} \right.$ | $\left\{ \begin{array}{l} K \rightarrow g \\ J \rightarrow d \end{array} \right.$ |
| $F \rightarrow v$ | $T \rightarrow T m F$ | $\left\{ \begin{array}{l} T \rightarrow S_5 F \\ T \rightarrow v \end{array} \right.$ | $\left\{ \begin{array}{l} T \rightarrow v \\ T \rightarrow n \end{array} \right.$ |
| $F \rightarrow n$ | $T \rightarrow v$ | $\left\{ \begin{array}{l} T \rightarrow n \\ T \rightarrow S_6 J \end{array} \right.$ | $\left\{ \begin{array}{l} F \rightarrow v \\ F \rightarrow n \end{array} \right.$ |
| $F \rightarrow g E d$ | $T \rightarrow n$ | $\left\{ \begin{array}{l} F \rightarrow v \\ F \rightarrow n \end{array} \right.$ | $\left\{ \begin{array}{l} F \rightarrow v \\ F \rightarrow n \end{array} \right.$ |
| | $T \rightarrow g E d$ | $\left\{ \begin{array}{l} F \rightarrow n \\ F \rightarrow S_6 J \end{array} \right.$ | $\left\{ \begin{array}{l} F \rightarrow S_6 J \end{array} \right.$ |

Figure 15

Algorithme d'analyse multiple

```

pour J := 2 pas 1 jusqu'à L faire
  pour I := 1 pas 1 jusqu'à L - J + 1 faire
    début K := 1 ;
      pour L1 := 1 pas 1 jusqu'à J - 1 faire
        début K1 := 0 ;
          pour K1 := K1 + 1 tant que T [I, L1, K1] ≠ ETOILE faire
            début K2 := 0 ;
              pour K2 := K2 + 1 tant que T [I + L1, J - L1,
                K2] ≠ ETOILE faire
                début KM := 1 ;
                  CODA : ALPHA := M [T [I, L1, K1],
                    T [I + L1, J - L1, K2], KM] ;
                  si ALPHA = ETOILE alors allera
                    EPUISEE ;
                  T [I, J, K] := ALPHA ;
                  K := K + 1 ; KM := KM + 1 ;
                  allera CODA ;
            EPUISEE : fin
          fin
        fin
      fin
    fin
  fin
fin ;

```

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 |
|-------------------|---|----|---|----|---|---|---|---|----|----|----|----|----|----|----------------|----------------|----------------|----------------|----------------|----------------|
| | S | P | A | E | T | F | B | C | D | G | H | I | J | K | S ₁ | S ₂ | S ₃ | S ₄ | S ₅ | S ₆ |
| 1 S | | | | | | | | | | | | | | | | | | | | |
| 2 P | | | | | | | | | | 17 | | | | | | | | | | |
| 3 A | | | | | | | | | | | | | | | | | | | | |
| 4 E | | | | | | | | | | | 18 | | | | | | | | | |
| 5 T | | | | | | | | | | | | 19 | | | | | | | | |
| 6 F | | | | | | | | | | | | | | | | | | | | |
| 7 B | | 15 | | | | | | | | | | | | | | | | | | |
| 8 C | | | | | | | | | 16 | | | | | | | | | | | |
| 9 D | | | | | | | | | | | | | | | | | | | | |
| 10 G | | | | | | | | | | | | | | | | | | | | |
| 11 H | | | | | | | | | | | | | | | | | | | | |
| 12 I | | | | | | | | | | | | | | | | | | | | |
| 13 J | | | | | | | | | | | | | | | | | | | | |
| 14 K | | | | 20 | | | | | | | | | | | | | | | | |
| 15 S ₁ | | | | | | | 1 | | | | | | | | | | | | | |
| 16 S ₂ | | | | 2 | 3 | | | | | | | | | | | | | | | |
| 17 S ₃ | | | 2 | | | | | | | | | | | | | | | | | |
| 18 S ₄ | | | | | 4 | | | | | | | | | | | | | | | |
| 19 S ₅ | | | | | | 4 | 5 | | | | | | | | | | | | | |
| 20 S ₆ | | | | | | | | | | | | | | | 4 | 5 | 6 | | | |

Figure 16

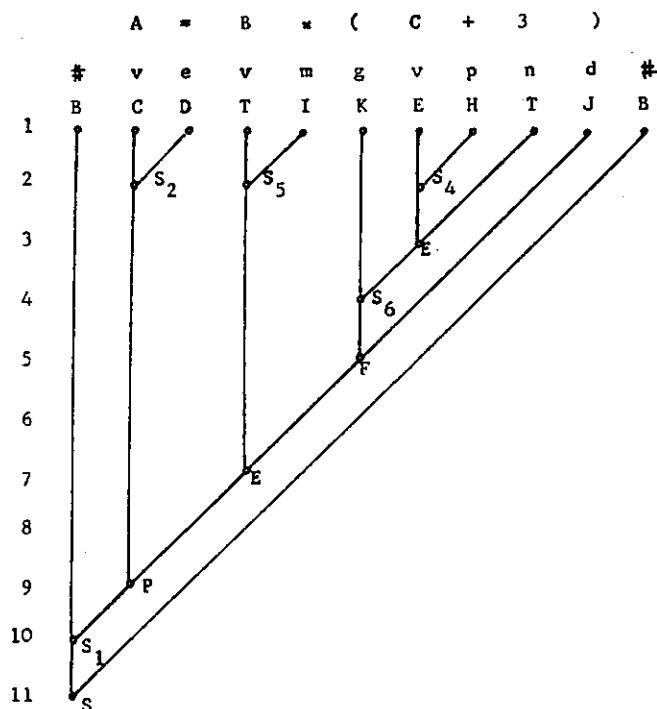


Figure 17

V. — CONCLUSION

Nous ne chercherons pas dans ces conclusions à établir une hiérarchie de valeurs entre les trois algorithmes présentés. Bien qu'ayant le moyen de connaître l'encombrement des mémoires provoqué par chacun d'eux et de mesurer le temps de déroulement d'un programme sur le calculateur utilisé, nous n'avons pas établi de comparaisons, le nombre de chaînes analysées et celui des grammaires auxquelles elles se réfèrent étant relativement réduit.

Les algorithmes présentés ne sont pas vraiment comparables, le troisième donnant plusieurs analyses pour une chaîne, les deux premiers donnant une seule réponse. Même si nous limitons notre comparaison à ceux-ci nous avons dit que leur degré de généralité quant aux langages auxquels ils s'appliquent était différent. D'autre part, pour le deuxième algorithme, le temps d'analyse est diminué grâce à la matrice *SUCCESSEUR* qui pourrait être utilisée avec le même effet dans le premier algorithme.

Récemment Griffiths et Petrick (cf. référence 18) ont établi une comparaison entre les efficacités relatives d'algorithmes d'analyse pour langages « context-free ». Ils ont simulé sur un calculateur les automates à piles construits grâce à chacun des algorithmes et ont pu ainsi comparer

les temps nécessaires pour analyser plusieurs chaînes de langages différents (1). Mais le simple fait de décrire les algorithmes par des automates fausse la réponse en ne tenant pas compte des caractéristiques de la machine utilisée : nombre de registres d'index, utilisation de l'adressage indirect, capacité de mémoires, etc...

Nous présenterons simplement des remarques suggérées par l'étude faite ici dans le but de trouver un algorithme pour réaliser la première phase du travail d'un compilateur : l'analyse. Le choix dépend évidemment du calculateur employé et de la famille de langages que l'on désire traiter. L'ordre dans lequel sont rangées les règles de la grammaire joue un rôle quant à la rapidité d'analyse. Un choix doit être fait pour cet ordre en fonction de l'algorithme employé. Il est évident que ce choix conditionne également la structure obtenue pour une chaîne ambiguë dans le cas de l'analyse unique.

La présence en mémoire de la matrice *SUCCESSEUR* apporte un gain de temps, mais exige des mémoires disponibles. La capacité de la machine décidera de son emploi ou non. Un traitement de liste pour cette matrice diminue d'ailleurs les exigences en quantité de mémoires, au cas où la matrice est tellement creuse que son rangement par bit n'est pas justifié. *SUCCESSEUR* rend l'algorithme sélectif, en ce sens que ce dernier choisit un but intermédiaire quand il sait pouvoir atteindre le but final par cette voie.

Il ressort de l'étude faite par Griffiths et Petrick que *l'algorithme sélectif d'analyse ascendante est plus efficace que celui d'analyse descendante*. Dans leurs méthodes, la partie de recul est implicitement contenue dans le fait que l'automate est non déterministe. Tel que nous présentons le recul, il relève uniquement, pour une grammaire donnée, des techniques de programmation. Le retour en arrière est réduit si l'on transforme la grammaire sous forme standard et les algorithmes sont alors plus rapides s'ils sont également sélectifs. Mais la transformation opérée peut modifier les unités syntaxiques qui, parfois, perdent leur valeur sémantique. Dans le cas où la génération suit l'analyse, comme dans un compilateur, il faudra donc reconstituer les unités sémantiques avant de passer à la phase de génération. Le problème est de savoir si l'on perd ainsi tous les avantages obtenus précédemment.

Si l'on désire seulement faire une recherche d'erreurs, nous avons vu qu'il valait mieux obtenir toutes les structures possibles pour une chaîne donnée. Cela permet de mieux détecter les erreurs et facilite leur correction. Dans ce cas, la grammaire transformée en forme normale ou standard permet d'utiliser des algorithmes plus rapides.

Nous avons dit plus haut que le choix de l'algorithme dépendait du type de langages que l'on doit traiter. Dans le cas où les grammaires ont des règles récursives à gauche, de la forme $A \rightarrow A\alpha$ (α : chaîne sur $V_T \cup V_N$), le traitement présenté au chapitre II suffit. Il est inutile de

(1) Quelques-unes de ces chaînes étaient en ALGOL et en LISP.

rendre l'algorithme plus puissant que le langage auquel il s'applique. Il est donc intéressant de connaître les propriétés de la grammaire correspondante ; les études faites dans ce but peuvent aider à concevoir des algorithmes efficaces et adaptés à une classe de langages bien définie.

Pour la bibliographie se référer à la première partie de l'article, parue dans le n° 2, 1965.