Analyse syntaxique en environnement parallèle

François Baccelli, T. Fleury

▶ **To cite this version:**

François Baccelli, T. Fleury. Analyse syntaxique en environnement parallèle. [Rapport de recherche] RR-0049, INRIA. 1981. inria-00076512

HAL Id: inria-00076512
https://inria.hal.science/inria-00076512

Submitted on 24 May 2006

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

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

INRIA

Institut National
de Recherche
en Informatique
et en Automatique

Domaine de Voluceau
Rocquencourt
B.P 105 78150 Le Chesnay
France
Tél. 954 90 20

Rapports de Recherche

N° 49

ANALYSE SYNTAXIQUE EN ENVIRONNEMENT PARALLÈLE

François BACCELLI
Thierry FLEURY

Janvier 1981

RECTIFICATIF

ANALYSE SYNTAXIQUE EN ENVIRONNEMENT PARALLELE

François BACCELLI, Thierry FLEURY

INRIA
Domaine de Voluceau
Rocquencourt, BP 105
78153 Le Chesnay,France

ERRATA

*p.3, 7ème ligne de la procédure décomposition ;
remplacer par :

"$\beta_{j,i+1} \leftarrow g(\beta_{j,i})\ D(Q_{j,i+1})\ d(\beta_{j,i})$; $i \leftarrow i+1$;"

*p.5, équation (11) ; le dernier terme de la somme est :

"... + $\delta_{SQ}\ n_{\mathcal{O}}(w)$"

*p.6, 4ème ligne du paragraphe 4 ; lire :

"la structure générative..."

*p.7, équation (24) ; remplacer par :

"$p^i(\vec{n}) = \sum_{\vec{m} \in D_i} a^i_{\vec{m}}\ C(\vec{n}-\vec{l}_i, \vec{W}^{\vec{m}})$, $i=1,p$"

*p.8, remplacer le paragraphe 5.2 par le paragraphe 5.2 du texte ci-joint (p.II)

p.11, au deuxième "" ; remplacer par :

"si $\vec{\ell}$ est tel que $0 < \ell_i < N_i$..."

*p.11, dernière ligne à gauche ; lire :

"Q : ρ=0,5 ; Q' : ρ=0,9"

*p.10, dernière ligne avant la procédure FILS DROIT; lire :

... la suite $(\theta_1,\ldots,\theta_q)$.

*p.11, ajout de la référence :

[FIS 80] C.N. FISCHER "On parsing and compiling arithmetic expressions on vector computers", ACM Trans. Program. Lang. and Syst. Vol 2 n°2, p.203-24 April 1980.

5.2. Exemple de calcul du parallélisme intrinsèque

Ce paragraphe est consacré au calcul de P dans l'exemple introduit au paragraphe 2.1.

5.2.1. Calcul de $E[F_{SQ}]$ et $E[F_{SG}]$

Les moyennes $E[n_1(w)]$, $E[n_6(w)]$, $E[n_{\mathcal{O}}(w)]$ peuvent se calculer, soit à partir de la méthode récursive (4.4), soit directement à partir de l'équation (20); Pour $E[n_6(w)]$ par exemple, il suffit d'inverser le système linéaire :

$$(26)\quad \begin{cases} \dfrac{d}{dS_6} W_i(1,..,1,S_6,1,..,1)\Big|_1^{\rightarrow} = \delta_{i,6} + \sum_{j=1}^{p} m_{ij} \dfrac{d}{dS_6} W_j(1,..,1,S_6,1,..,1)\Big|_1^{\rightarrow} \\ i = 1,p \end{cases}$$

pour obtenir $E[n_6(w)] = \dfrac{d}{dS_6} W_S(1,..,1,S_6,1,..,1)\Big|_1^{\rightarrow}$.

5.2.2. Calcul de $E[\max_{k=1,3} \max_{j=1,m_k(w)} F_k(w_{j,k}(w))]$

Notons tout d'abord qu'une probabilisation des productions multiples de chacune des sous grammaires G_k, k=1,3, identique à celle des productions correspondantes de G_A, définit sans ambiguité des grammaires stochastiques $\{G_k,Q_k\}$ satisfaisant nécessairement la condition de Frobenius.

Lemme 3 : Soit, pour $\ell \geq 0$, $n \geq 0$:

$$\pi_1(\ell,n) = Q_1[n_1(x) = \ell,\ F_1(x) \leq n] \qquad x \in L(G_1)$$
$$\pi_2(\ell,n) = Q_2[n_6(y) = \ell,\ F_2(y) \leq n] \qquad y \in L(G_2)$$
$$\pi_3(\ell,n) = Q_3[n_6(z) = \ell,\ F_3(z) \leq n] \qquad z \in L(G_3)$$

Soit $f_n(u) = \sum_{\ell=0}^{\infty} u^{\ell}\, \pi_3(\ell,n)$. pour $u \in \mathbb{R}^+$.

L'équation de point fixe $u=f_n(u)$ admet une solution unique $\phi_n \in [0,1]$. De plus,

$$\xi_n = Q[\max_{k=1,3} \max_{j=1,m_k(w)} F_k(w_{j,k}(w)) \leq n]$$

est donné par :

$$\xi_n = \sum_{k=0}^{\infty} \pi_1(k,n) \left[\sum_{\ell=0}^{\infty} \pi_2(\ell,n)(\phi_n)^{\ell}\right]^k \qquad \square$$

Démonstration : Soit $w \in L(G_A)$: $S \overset{*}{\underset{G_A}{\Rightarrow}} w \in \Sigma^*$ (resp. $\hat{w}/M_1 \overset{*}{\underset{G_A}{\Rightarrow}} \hat{w} \in \Sigma^*$, $\tilde{w}/M_6 \overset{*}{\underset{G_A}{\Rightarrow}} \tilde{w} \in \Sigma^*$). Notons que la procédure de décomposition est applicable à $\hat{w}$ et $\tilde{w}$ pour une initialisation appropriée ($\alpha_o = M_1$, $\alpha_o = M_6$). Soit donc $w_{j,k}(w)$, k=1,3, j=0, $m_k(w)$ (resp $w_{j,k}(\hat{w})$, k=2,3, j=0,$m_k(\hat{w})$; $w_{j,k}(\tilde{w})$ k=3, j=0,$m_k(\tilde{w})$) la décomposition de w (resp $\hat{w}$, $\tilde{w}$), ordonnée en fonction du type (k) et de l'ordre dans lequel les mots du type sont découverts (j).

Notons d'autre part que les mots w, $\hat{w}$, $\tilde{w}$ sont des événements de la grammaire stochastique (G_A,Q). Soit $w \in L(G_A)$. $m_1(w) = n_1(w) = 1$; soit $\hat{w}_i(\hat{w})$, le mot de Σ^* généré par le sous arbre de w ayant pour racine de i-ième M_1 de $w_{1,1}(w)$, i=1,$n_1(w_{1,1}(w))$. on a :

$$\xi_n = \sum_{k=0}^{\infty} Q[\{n_1(w_{1,1}(w)) = k\},\ \{F_1(w_{1,1}(w)) \leq n\},$$
$$\bigcap_{i=1}^{k} \{\max_{k=2,3} \max_{j=1,m_k(\hat{w}_i)} F_k(w_{j,k}(\hat{w}_i)) \leq n\}]$$
$$= \sum_{k=0}^{\infty} \pi_1(k,n)(\psi_n)^k$$

où

$$\begin{cases} \psi_n = Q[\max_{k=2,3} \max_{j=1,m_k(\hat{w})} F_k(w_{j,k}(\hat{w})) \leq n] \\ \hat{w} \ /\ M_1 \overset{*}{\underset{G_A}{\Rightarrow}} \hat{w} \quad \text{sous Q.} \end{cases}$$

Soit $\tilde{w}_i$ le mot de Σ^* généré par le sous arbre de $\hat{w}$ ayant pour racine le i-ième M_6 de $w_{1,2}(\hat{w})$, i=1,n_6 i=1,$n_6(w_{1,2}(\hat{w}))$. On a :

$$\psi_n = \sum_{\ell=0}^{\infty} Q[\{n_6(w_{1,2}(\hat{w})) = \ell\},\ \{F_2(w_{1,2}(\hat{w})) \leq n\},$$
$$\bigcap_{i=1}^{\ell} \{\max_{j=1,m_3(\tilde{w}_i)} F_3[w_{j,3}(\tilde{w}_i)] \leq n\}] \ ,$$
$$= \sum_{\ell=0}^{\infty} \pi_2(\ell,n)(\phi_n)^{\ell}$$

où

$$\begin{cases} \phi_n = Q[\max_{j=1,m_3(\tilde{w})} F_3(w_{j,3}(\tilde{w})) \leq n] \\ \tilde{w}_i \ /\ M_6 \overset{*}{\underset{G_A}{\Rightarrow}} \tilde{w} \quad \text{sous Q} \end{cases}$$

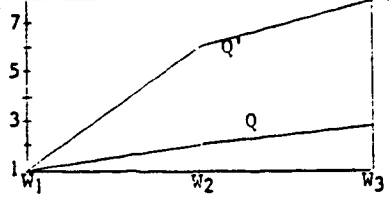
Soit enfin $\acute{w}_i$ le mot de Σ^* généré par le sous arbre de $\hat{w}$ ayant pour racine le i-ième M_6 de $w_{1,3}(\tilde{w})$, i=1,$n_6(w_{1,3}(\tilde{w}))$. On a :

$$\phi_n = \sum_{q=0}^{\infty} Q[\{n_6(w_{1,3}(\tilde{w})) = q\},\ \{F_3(w_{1,3}(\tilde{w})) \leq n\},$$
$$\bigcap_{i=1}^{q'} \{\max_{j=1,m_3(\acute{w}_i)} F_3[w_{j,3}(\acute{w}_i)] \leq n\}]$$
$$= \sum_{q=0}^{\infty} \pi_3(q,n)\ (\phi_n)^q \quad .$$

L'équation de point fixe $u=f_n(u)$ admet une solution unique dans [0,1], puisque $f_n(o) > o$, $f_n(1) \leq 1$, et f_n convexe.

5.2.3. Résultats numériques

Le parallélisme intrinsèque est calculé pour $W_1=\{S\}$, $W_2=\{S,M_1\}$, $W_3=\{S,M_1,M_6\}$ et pour deux probabilisations Q et Q' (voir annexe 5) de G_A :



ANALYSE SYNTAXIQUE EN ENVIRONNEMENT PARALLELE

François BACCELLI, Thierry FLEURY

INRIA
Domaine de Voluceau
Rocquencourt, BP.105
Le Chesnay, FRANCE

Résumé

Un schéma de décomposition de l'analyse syntaxique des mots du langage d'une grammaire context-free en sous analyses syntaxiques exécutables concurremment est proposé. Des exemples de grammaires sont présentés pour lesquelles cette décomposition permet de définir un analyseur syntaxique parallèle adapté à un environnement multiprocesseurs. Les performances de l'analyseur d'une grammaire arithmétique ainsi parallèlisé sont évaluées au moyen de modèles probabilistes et de mesures effectuées sur une réalisation.

Abstract

Some decomposition of the syntactic analysis of the sentences of context-free grammars into sequences of independant sub-tasks is proposed. Examples of grammars are presented, for which this decomposition provides an efficient parser for a multiprocessing environment. The performances of such a parser for an artihmetic infix grammar are evaluated by means of probabilistic models and measurements performed on a real case.

0. Introduction

L'idée d'exécuter, en environnement parallèle, les diverses phases d'un processus de compilation selon un schéma pipe-line a été considérée par de nombreux auteurs : ([BAE 77], [BAN 79]). Mais qu'en est-il du parallélisme intrinsèque à chacune de ces phases ? Peut-on élaborer des algorithmes qui permettent, en environnement parallèle, de réduire de manière significative le coût en temps de chacune de ces phases ? Les résultats théoriques et les réalisations exposés dans cet article tentent de répondre à cette question, en ce qui concerne l'analyse syntaxique.

La première partie de cette étude concerne un schéma de décomposition des mots $w \in L(G)$ d'une grammaire C.F., G conduisant à une décomposition de l'analyse syntaxique de w en une suite de sous analyses syntaxiques exécutables concurremment. Dans la mesure où il existe une procédure efficace qui réalise la segmentation, cette décomposition conduit à la définition d'un analyseur parallèle efficace en environnement multiprocesseurs.

Dans la deuxième partie, des modèles probabilistes fondés sur la notion de grammaire stochastique sont proposés, permettant de prédire les performances d'un analyseur de ce type et de résoudre certains problèmes d'optimisation liés à la finesse de la segmentation. De plus, une réalisation a été entreprise en complément des résultats théoriques, afin d'évaluer directement les performances d'un tel analyseur. Cette réalisation, décrite dans la troisième partie, consiste en l'implémentation de l'analyseur syntaxique parallèle d'une grammaire arithmétique de précédence sur un réseau de trois APPLE II.

SECTION 1 - Une méthode d'analyse syntaxique et environnement parallèle.

1. Généralités sur l'analyse syntaxique [HOP 69]

1.1. Grammaire formelle context-free

1.1.1. Définition

Une grammaire formelle context-free G est un quadruplet $G = \{V, \Sigma, S, P\}$ où

- V est l'ensemble des symboles de l'alphabet de G
- $\Sigma \subset V$ est l'ensemble des symboles de l'alphabet terminal de G
- P est l'ensemble des productions $\alpha \to \beta$ avec $\alpha \in V-\Sigma = V_N$ (alphabet non terminal) $\beta \in V^*$
- S est un élément distingué de V_N n'apparaissant dans aucune partie droite des productions de P.

Une relation $\underset{G}{\Rightarrow}$ est définie entre les éléments de V^* :

$w_1 \underset{G}{\Rightarrow} w_2$ ssi $\exists\, \alpha \in V_N,\ \beta,\gamma,\delta \in V^*$

$w_1 = \gamma\alpha\delta$, $w_2 = \gamma\beta\delta$ et $\alpha \to \beta \in P$.

On nomme langage défini par la grammaire G le sous-ensemble de Σ^* : $L(G) = \{w \in \Sigma^* \mid S \overset{*}{\Rightarrow} w\} \triangleq \{w \in \Sigma^* / \exists\, \alpha_1, \ldots, \alpha_m \in V^* / S \underset{G}{\Rightarrow} \alpha_1 \underset{G}{\Rightarrow} \alpha_2 \ldots \underset{G}{\Rightarrow} \alpha_m \underset{G}{\Rightarrow} w\}$.

Les auteurs remercient PIERRE DERANSART pour l'intérêt qu'il a porté à cette étude et le groupe PEPIN pour sa contribution à la réalisation.

1.1.2. Exemple

La grammaire arithmétique G_A (les symboles terminaux de G_A sont soulignés):

$G_A = \{\Sigma, V, S, P\}$

$\Sigma = \{\underline{+}, \underline{-}, \underline{*}, \underline{/}, \underline{**}, \underline{\surd}, \underline{(}, \underline{)}, \underline{id}, \underline{\neq}\}$

$V_N = \{S, M_i \ (i=0,7)\}$

P : (1) $S \rightarrow M_0 \underline{\neq}$

(2), (2)' $M_0 \rightarrow \underline{\neq} M_1 \mid M_0 \underline{\neq} M_1$

(3(j)) $M_1 \rightarrow M_j \qquad j \geq 2$

(4(j,k,$\bar{+}$)) $M_2 \rightarrow M_j \ \underline{+} \ M_k \mid M_j \ \underline{-} \ M_k \qquad j \geq 2;\ k > 2$

(5(j,k,$\bar{*}$)) $M_3 \rightarrow M_j \ \underline{*} \ M_k \mid M_j \ \underline{/} \ M_k \qquad j \geq 3;\ k > 3$

(6(j,k)) $M_4 \rightarrow M_j \ \underline{**} \ M_k \qquad j > 4;\ k \geq 4$

(7(j)) $M_5 \rightarrow \underline{\surd} M_j \qquad j \geq 5$

(8(j)) $M_6 \rightarrow \underline{(} M_j \underline{)} \qquad j \geq 2$

(9) $M_7 \rightarrow \underline{id}$

1.2. Arbres de dérivation et analyse syntaxique

1.2.1. Arbres de dérivation

Un arbre de dérivation de $w \in L(G)$ est une représentation graphique de l'ensemble des productions de G qui ont été utilisées pour générer w à partir de S. Formellement, un arbre de dérivation de $w \in L(G)$ est un arbre dont chaque noeud interne est un non terminal de G, chaque feuille un terminal et tel que

- les fils (lus de gauche à droite) de tout noeud interne coincident avec une production du non terminal associé à ce noeud
- les feuilles (lues de gauche à droite) coïncident avec w.

1.2.2. Exemple

Un arbre de dérivation de

$\underline{\neq} \ (id - \surd id) / id \ \underline{\neq} \in L(G_A)$

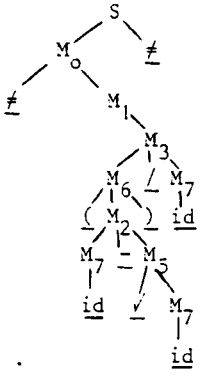


Fig.1 .

1.2.3. Ambiguité

Une grammaire G est dire non ambiguë si chaque $w \in L(G)$ n'est dérivable de S que par un seul arbre, noté A(w). Le caractère de non ambiguité d'une grammaire n'est pas décidable : des conditions suffisantes de non ambiguité peuvent être trouvées dans [GRI71]. La non ambiguité de G_A est démontrée dans [FIS 75].

1.2.4. Algorithme d'analyse syntaxique d'une grammaire non ambigue

Nous considérons comme algorithme d'analyse syntaxique tout algorithme fournissant l'arbre syntaxique unique, générant $w \in L(G)$, à partir de la donnée de w. Un algrithme d'analyse syntaxique de G_A au moyen de fonctions de précédences est présenté en annexe (annexe 1).

2. Décomposition de l'analyse syntaxique d'une grammaire CF

2.1. Sous grammaire

Soit $G = \{\Sigma, V, S, P\}$ une grammaire CF non ambiguë. Soit $B \in V_N$; On notera $PR(B) = (p_1, p_2, \ldots, p_\ell)$, l'ensemble des productions de P admettant B pour partie gauche. Pour $q \in P$, on notera G(q) le non terminal partie gauche de q et D(q) l'élément de V^*, partie droite de q. Ainsi $q = G(q) \rightarrow D(q)$. Soit $W \subset V_N$ fixé. Pour $B \in V_N$, nous nommons sous grammaire engendrée par B la grammaire CF : $G(B) = \{\Sigma_B, V_B, S_B, P_B\}$ construite comme suit :

(1)
- S_B est un nouveau symbole $\notin V$
- $P_B = \bigcup_{q \in PR(B)} \{S_B \rightarrow D(q)\} \cup \{PR(X)/X \in V_N - W \text{ et } B \overset{*}{\underset{G}{\Rightarrow}} \alpha X \beta, \ \alpha, \beta \in V^*\}$
- $V_{N_B} = \{G(q)/q \in P_B\}$
- $\Sigma_B = \{x \in \Sigma \cup W / B \overset{*}{\underset{G}{\Rightarrow}} \alpha x \beta, \quad \alpha, \beta \in V^*\}$

2.2. Exemples de sous grammaires

Considérons l'exemple de la grammaire G_A pour $W = \{S, M_1, M_6\}$; les grammaires $G_3 = G(M_6)$, $G_2 = G(M_1)$, $G_1 = G(S)$ sont caractérisées par :

$\Sigma_3 = \{\underline{M_6}, \underline{+}, \underline{-}, \underline{*}, \underline{/}, \underline{**}, \underline{\surd}, \underline{(}, \underline{)}, \underline{id}\}$

$V_{N_3} = \{S_3, M_j, \ j=2,7, \ j \neq 6\}$

$P_3 = \{S_3 \rightarrow \underline{(} \ M_j \ \underline{)} \ j=2,5 \text{ et } j=7 \ ; \ S_3 \rightarrow \underline{(} \ \underline{M_6} \ \underline{)} \ ;$
$M_2 \rightarrow M_j \ \underline{\pm} \ M_k, \ j \geq 2, \ k > 2, \ j \neq 6, \ k \neq 6 \ ;$
$M_2 \rightarrow \underline{M_6} \ \underline{\pm} \ M_k, \ k > 2 ; \ M_2 \rightarrow M_j \ \underline{\pm} \ \underline{M_6}, \ j \geq 2 \ ;$
$M_3 \rightarrow M_j \ \underline{/} \ M_k, \ j \geq 3, \ k > 3, \ j \neq 6, \ k \neq 6 \ ;$
$M_3 \rightarrow \underline{M_6} \ \underline{/} \ M_k, \ k > 3 \ ; \ M_3 \rightarrow M_j \ \underline{/} \ \underline{M_6}, \ j \geq 3 \ ;$
$M_4 \rightarrow M_j \ \underline{**} \ M_k, \ j > 4, \ k \geq 4, \ j \neq 6 \ ;$
$M_4 \rightarrow \underline{M_6} \ \underline{**} \ M_k, \ k \geq 4 \ ; \ M_4 \rightarrow M_j \ \underline{**} \ \underline{M_6}, \ j > 4 \ ;$
$M_5 \rightarrow \underline{\surd} M_j, \ j=5,7 \ ; \ M_5 \rightarrow \underline{\surd} \underline{M_6} \ ; \ M_7 \rightarrow \underline{id}.$

$\Sigma_2 = \{\underline{M_6}, \underline{+}, \underline{-}, \underline{*}, \underline{/}, \underline{**}, \underline{\surd}, \underline{id}\}$

$V_{N_2} = \{S_2, M_j, \ j=2,7, \ j \neq 6\}$

$P_2 = \{S_2 \rightarrow M_j, \ j=2,7, \ j \neq 6 \ ; \ S_2 \rightarrow \underline{M_6}\} \cup (P_3 - PR(S_3))$

$\Sigma_1 = \{\underline{\neq}, \underline{M_1}\}$

$V_{N_1} = \{S_1, M_0\}$

$P_1 = \{S_1 \rightarrow M_0 \ \underline{\neq} \ ; \ M_0 \rightarrow \underline{\neq} \ \underline{M_1} ; \ M_0 \rightarrow M_0 \ \underline{\neq} \ \underline{M_1}\}$

2.3. Schéma de décomposition de l'analyse syntaxique d'une grammaire CF selon une famille de sous grammaires

Soit $G = \{\Sigma, V, S, P\}$ une grammaire CF non ambiguë. Soit $W = (V_1, \ldots, V_n) \subset V_N$, $V_1 = S$ et $\bar{W} = V_N - W$, deux sous ensembles fixés de V_N. Soit enfin $G_i = \{\Sigma_i, V_i, S_i, P_i\}$ la sous grammaire engendrée par V_i pour W, $i=1,n$. La procédure ci-dessous (2.3.2) permet de décomposer w dand L(G) en une suite de mots appartenant aux langages des sous grammaires $G_1, \ldots, G_n$:

2.3.1. Quelques notations

Soit $\alpha \in V^*$; $Y \subset V$; nous noterons $\phi_Y(\alpha)$ celui des symboles de α appartenant à Y située le plus à droite dans α (0 s'il n'y en a pas). Si $\phi_Y(\alpha) \neq 0$, on peut écrire α sous la forme : $g(\alpha)\phi_Y(\alpha)d(\alpha)$ de manière unique.

Soit w fixé, $w \in L(G)$ et $S \xrightarrow{Q_1} \alpha_1 \xrightarrow{Q_2} \ldots \xrightarrow{Q_i} \alpha_i \overset{*}{\underset{G}{\Rightarrow}} w$ $Q_j \in P$. Supposons que $\alpha_i = \beta c \gamma$, où $c \in V_N$ et $\alpha, \beta \in V^*$ La non ambiguité de G implique l'existence d'une unique règle de production $\{c \to \mu\} \in P$ telle que $\beta\mu\gamma \overset{*}{\Rightarrow} w$. Nous noterons $f(\beta,c,\gamma) \in P$ cette production.

2.3.2. Schéma de décomposition

Ce schéma est indiqué par la procédure ci-dessous qui consiste en une génération formelle de w à partir de S. La spécificité de cette génération réside dans les règles de priorités suivantes, appliquées à toute étape de dérivation :

- dériver en priorité le non terminal de $\bar{W}$ le plus à droite.
- s'il n'y a plus de non terminal de $\bar{W}$ dériver le non terminal de W le plus à droite.

Cette procédure admet en entrée $w \in L(G)$ et en sortie, k mots $w_1 \in L(G_{e(1)}) \ldots, w_k \in L(G_{e(k)})$.

PROC DECOMPOSITION;

$\alpha_0 \leftarrow S$; $j \leftarrow 0$;

tant que $\phi_W(\alpha_j) \neq 0$; Soit $\ell / \phi_W(\alpha_j) = V_\ell$; $e(j+1) \leftarrow \ell$;

$Q_{j,1} \leftarrow f(g(\alpha_j), \phi_W(\alpha_j)), d(\alpha_j))$; $\beta_{j,1} \leftarrow g(\alpha_j) D(Q_{j,1})$ $d(\alpha_j)$ $i \leftarrow 1$;

tant que $\phi_{\bar{W}}(\beta_{j,i}) \neq 0$ [+]

$Q_{j,i+1} \leftarrow f(g(\beta_{j,i}), \phi_{\bar{W}}(\beta_{j,i}), d(\beta_{j,i}))$;

$\beta_{j,i+1} \leftarrow f(g(\beta_{j,i}), D(Q_{j,i+1}), d(\beta_{j,i}))$; $i \leftarrow i+1$;

f tant que

soit $\alpha / \beta_{j,i} = g(\alpha_j)\alpha\, d(\alpha_j)$; $j \leftarrow j+1$; $w_j \leftarrow \alpha$; $\alpha_j \leftarrow \beta_{j-1,i}$;

f tant que

$k \leftarrow j$;

f proc

2.3.3. Quelques propriétés de la décomposition

Par construction, les w_j $j=1,k$ possèdent les propriétés suivantes :

(2) $w_j \in L(G_{e(j)})$ $\quad j=1,k$

[+] on a nécessairement $\phi_{\bar{W}}(g(\alpha_j))=0$ et $\phi_{\bar{W}}(d(\alpha_j))=0$.

(3) $S = \alpha_0 \overset{Q_{0,1}}{\to} \beta_{0,1} \overset{Q_{0,2}}{\to} \beta_{0,2} \cdots \overset{Q_{0,i_0}}{\to} \beta_{0,i_0} = \alpha_1 \overset{Q_{1,1}}{\to} \beta_{1,1} \cdots \overset{Q_{1,i_1}}{\to} \beta_{1,i_1} = \alpha_2 \cdots \beta_{k-1,i_{k-1}} = \alpha_k = w$

si bien que :

(4) $S \overset{*}{\underset{G}{\Rightarrow}} \alpha_1 \overset{*}{\underset{G}{\Rightarrow}} \alpha_2 \cdots \overset{*}{\underset{G}{\Rightarrow}} \alpha_k = w$

les propriétés (5) et (6) sont déduites de (3) :

(5) $S \xrightarrow{\pi_0} \alpha_1 \xrightarrow{\pi_1} \alpha_2 \cdots \xrightarrow{\pi_{k-1}} \alpha_k = w$

où $\alpha_j \xrightarrow{\pi_j}$ consiste à remplacer $\phi_W(\alpha_j)$ par w_{j+1} dans α_j.

(6) $S \xrightarrow{\psi_0} A(\alpha_1) \xrightarrow{\psi_1} A(\alpha_2) \cdots \xrightarrow{\psi_{k-1}} A(\alpha_k) = A(w)$

où $A(\alpha_j) \xrightarrow{\psi_j}$ consiste à remplacer celle des feuilles de $A(\alpha_j)$ appartenant à W et située le plus à droite par le sous arbre $\{\Omega_{j+1}$ /l'arbre de dérivation de w_{j+1} dans $G_{e(j+1)}$ s'écrit $A(w_{j+1}) = {}^{"}S_{e(j+1)}{}^{"}\}$

Ω_{j+1}

2.3.4. Commentaires

La procédure du paragraphe 2.3.4 reste formelle dans la mesure où aucun moyen de calcul de la fonction f (2.3.1) n'est spécifié.

2.4. Exemple de décomposition selon une famille de sous grammaires

Considérons la grammaire arithmétique G_A et $W = \{S, M_1, M_6\}$ soit

(7) $w = \neq(id+id)/(id-id) \neq \surd(id * (id+id)) \neq$

La procédure de décomposition détermine :

	α(j)	$\phi_W(\alpha_j)$	e(j+1)	w_{j+1}
0	S	S	1	$\neq M_1 \neq M_1 \neq$
1	$\neq M_1 \neq M_1 \neq$	M_1	2	$\surd M_6$
2	$\neq M_1 \neq \surd M_6 \neq$	M_6	3	$(id * M_6)$
3	$\neq M_1 \neq \surd(id * M_6) \neq$	M_6	3	(id+id)
4	$\neq M_1 \neq \surd(id * (id+id)) \neq$	M_1	2	M_6/M_6
5	$\neq M_6 / M_6 \neq \surd(id * (id+id)) \neq$	M_6	3	(id-id)
6	$\neq M_6 /(id-id) \neq \ldots$	M_6	3	(id+id)
7	$\neq(id+id)/(id-id) \neq \ldots$	0		

La décomposition (6) de A(w) est donnée ci-dessous

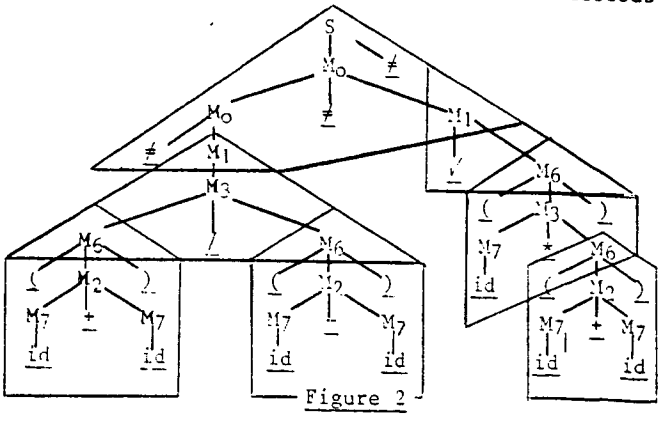


Figure 2

3. Procédure de segmentation selon une famille de sous grammaires

3.1. Introduction

Nous entendons par procédure de segmentation tout algorithme permettant de calculer la décomposition des mots $w \in L(G)$ selon la famille de sous grammaires engendrée par W sans toutefois nécessiter une analyse syntaxique complète de w (les notations sont celles du paragraphe 2). Plus précisément, soit $w \in L(G)$ et $R_{j,i}$ $j=0,k-1$, $i=1,n_j$, la réduction associée à la production $Q_{j,i}$ du schéma de décomposition de w. Considérons la propriété suivante, déduite de (3) :

$$(8)\quad w = \alpha_k \xrightarrow{R_{k-1,i_{k-1}}} \beta_{k-1,i_{k-1}-1} \cdots \cdots \xrightarrow{R_{k-1,1}}$$
$$\alpha_{k-1} \xrightarrow{R_{k-2,i_{k-2}}} \cdots$$
$$\vdots$$
$$\alpha_j \xrightarrow{R_{j-1,i_{j-1}}} \beta_{j-1,i_{j-1}-1} \cdots \cdots \xrightarrow{R_{j-1,1}}$$
$$\vdots$$
$$\alpha_1 \xrightarrow{R_{0,i_0}} \beta_{0,i_0-1} \cdots \cdots \xrightarrow{R_{0,1}} \alpha_0 = S$$

On dira qu'il existe un algorithme de segmentation ascendante si, à toute étape de réduction j, pour j variant de k à 1, il est possible de reconnaître à partir de la forme sentenciale α_j :

- $w_j \in V^*$ dans α_j
- $e(j) \in (1,..n)$

et ce sans avoir à déterminer successivement $R_{j-1,n_{j-1}}, \cdots, R_{j-1,1}$.

3.2. Exemple de procédure de segmentation utilisant des délimiteurs

Considérons la grammaire arithmétique G_A et $W = \{S, M_1, M_6\}$ soit $D = \{(,),\neq\}$; $D \subset \Sigma$ est une famille des délimiteurs permettant une segmentation de l'analyse syntaxique des mots de $L(G_A)$ selon les sous grammaires engendrées par W. Pour $w \in L(G_A)$, on peut écrire w de manière unique sous la forme :

$$(9)\quad w = d_1\gamma_1 d_2\gamma_2 \cdots \gamma_{j-1} d_j \gamma_j \cdots \gamma_m d_{m+1}$$

où $d_i \in D$ et $\gamma_i \in (\Sigma - D)^*$. Soit Γ_j $j=1,m$, le couple $(p(j,1), p(j,2))$ des positions respectives du premier et du dernier symbole de γ_j dans w (ces positions sont repérées par un nombre entier compris entre 1 et $|w|$) si $\gamma_j \neq \varepsilon$, (0,0) sinon. Nous entendons par squelette délimiteur de w, $\mathfrak{D}(w)$:

$$(10)\quad \mathfrak{D}(w) \triangleq d_1\Gamma_1 d_2\Gamma_2 \cdots \Gamma_{j-1} d_j \Gamma_j \cdots \Gamma_m d_{m+1}$$

Le transducteur à pile, ci dessous V admet en entrée le squelette délimiteur de w et en sortie k couples $(\hat{w}_i, \hat{e}(i))$ où $\hat{w}_i \in (\Gamma_j | D | W)^*$ et $\hat{e}(i) \in (1,2,..n)$ $i=1,\hat{k}$.

Le pointeur est initialisé sur Γ_1 :

```
PROC SEGMENTATION
empiler ≠ ; i ← 1 ;
    tant que pointeur ≤ |𝔇(w)| :
        si pointeur = "≠" : dépiler jusqu'au premier
        "≠" non inclus; ceci forme ŵi ; ê(i) ← 2;
        empiler "V_ê(i)"; empiler "≠"; i ← i+1 ;
        sinsi pointeur = ")" : empiler ")" ;
        dépiler jusqu'à la première "(" incluse;
        ceci forme ŵi; ê(i) ← 3; empiler "V_ê(i)";
        i ← i+1
        sinsi pointeur = "(" : empiler "(";
        sinon : empiler "Γj"
        fsi
    f tant que;
    tout dépiler; ceci forme ŵi; ê(i) ← 1; k̂ ← i
f proc
```

Lemme 1 : soit $w \in L(G)$ et $\mathfrak{D}(w)$ son squelette délimiteur. k et $(w_i, e(i))$ $i=1,k$ la décomposition calculée par la procédure formelle DECOMPOSITION pour w et $\hat{k}$, $(\hat{w}_i, \hat{e}(i))$ $i=1,\hat{k}$, calculés par la procédure de segmentation pour $\mathfrak{D}(w)$ on a :

$$\hat{k} = k$$
$$\hat{e}(i) = e(k-i+1) \qquad i=1,k$$
$$\hat{w}_i = w_{k-i+1} \qquad i=1,k$$

La dernière égalité s'entend après remplacement de chacun des $\Gamma_i = (p(j,1), p(j,2))$ par la suite des symboles de w situés entre p(j,1) et p(j,2), bornes comprises. (Pour la démonstration de ce lemme, voir l'annexe 2).

Par exemple, pour

$w = \neq(id+id)/(id-id)\neq\Gamma(id*(id+id))\neq$,

$\mathfrak{D}(w) = \neq\gamma_2(\gamma_3)\gamma_4(\gamma_5)\gamma_6\neq\gamma_7(\gamma_8(\gamma_9)\gamma_{10})\gamma_{11}\neq$

$\Gamma_2=(0,0)$, $\Gamma_3=(3,5)$, $\Gamma_4=(7,7)$, $\Gamma_5=(9,11)$, $\Gamma_6=(0,0)$

$\Gamma_7=(14,14)$, $\Gamma_8=(16,17)$, $\Gamma_9=(19,21)$, $\Gamma_{10}=(0,0)$,

$\Gamma_{11}=(0,0)$.

$\hat{w}_1 = (\Gamma_3)$	$\hat{e}(1) = 3$
$\hat{w}_2 = (\Gamma_5)$	$\hat{e}(2) = 3$
$\hat{w}_3 = \Gamma_2 M_6 \Gamma_4 M_6 \Gamma_6$	$\hat{e}(3) = 2$
$\hat{w}_4 = (\Gamma_9)$	$\hat{e}(4) = 3$
$\hat{w}_5 = (\Gamma_8 M_6 \Gamma_{10})$	$\hat{e}(5) = 3$
$\hat{w}_6 = \Gamma_7 M_6$	$\hat{e}(6) = 2$
$\hat{w}_7 = \neq M_1 \neq M_1 \neq$	$\hat{e}(7) = 1$

3.3. Autre exemple

Il existe une procédure de segmentation des mots de $L(G_A)$ selon la famille $w = \{S, M_1\}$ au moyen du délimiteur "≠".

4. Analyse syntaxique en environnement parallèle

4.1. Schéma de parallélisation d'une analyse syntaxique

On suppose disposer d'un réseau de n processeurs

identiques communiquant, dont un est dit central. On se limite à des algorithmes mettant en oeuvre un contrôle centralisé pour des raisons inhérentes à la compilation (le programme à analyser doit être localisé sur un processeur ainsi que l'arbre calculé). Soit G une grammaire CF. Supposons qu'il existe un algorithme efficace de segmentation des mots w de L(G), calculant la décomposition $(w_j(w), e(j,w))$ pour $j=1,k(w)$ selon les sous grammaires générées par W. L'analyse syntaxique de w peut alors être parallélisée comme suit : (L'analyse lexicale et) la procédure de segmentation de w sont exécutées sur le processeur central; dès que le couple w_j, e(j) est déterminé par ce dernier, l'analyse syntaxique de $w_j \in L(G_{e(j)})$ (i.e. la détermination des réductions $R_{j-1,n_{j-1}}, \cdots R_{j-1,1}$ de $G_{e(j)}$ définies par (8), si cette sous analyse est ascendante) peut être allouée à l'un des processeurs périphériques; Dès que celui-ci a calculé l'arbre $A(w_j)$, il le retourne au processeur central. L'arbre A(w) peut être considéré comme calculé dès que le processeur central possède chacun de $A(w_j)$ pour $j=1,k(w)$ (voir la propriété (6)).

4.2. Exemple

Considérons la grammaire G_A et $w \in L(G_A)$ donné par (7). Les flèches verticales sur le schéma ci-dessous indiquent les étapes d'exécution de la procédure de segmentation auxquelles l'information $(w_j, e(j))$ est calculée et peut être communiquée aux processeurs périphériques. Les flèches non verticales indiquent les communications en retour $(A(w_j))$, du périphérique concerné vers le central; Leur point de convergence indique la fin de l'analyse syntaxique parallèle de w.

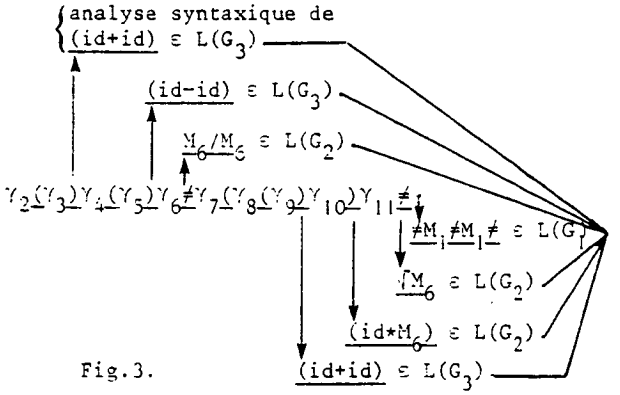


Fig.3.

Remarquons qu'il n'est pas nécessaire d'attendre la fin de l'exécution de la procédure de segmentation pour commencer les analyses indépendantes : l'implémentation d'un pipe-line entre la procédure de segmentation et l'ensemble de ces analyses permet en effet d'initialiser chacune des sous analyses syntaxiques ($w_i \in L(G_{e(i)})$ dès l'étape où $(w_j, e(j))$ est calculé, dans la mesure où il y a un processeur périphérique disponible. On peut évidemment attendre de meilleures performances pour la version avec pipeline de cet algorithme que pour sa version sans pipeline.

SECTION 2 - Evaluation de performances

Introduction : Soit G une grammaire CF segmentable. Une mesure du parallélisme intrinsèque à l'analyse syntaxique de $w \in L(G)$ est introduite (parallélisme ponctuel). On montre cependant que cette mesure est fortement dépendante du point w considéré. C'est pourquoi un modèle de grammaire stochastique est proposé, permettant la définition d'un parallélisme intrinsèque à l'analyse syntaxique de la grammaire G dans son ensemble (sous la forme d'une moyenne de la mesure du parallélisme ponctuel).

1. Algorithmes à fonction de coût

1.1. Définition

La définition ci-dessous est introduite dans un but de simplification des évaluations. Elle spécifie une classe d'algorithmes, admettant en entrée les mots du langage L(G) d'une grammaire C.F. non ambiguë, dont le coût en temps est fonction d'une information partielle sur l'arbre syntaxique (et non de la structure arborescente complète). Soit $(T_1, T_2, \ldots, T_p)$ où $p=|V|$, la suite des symboles du vocabulaire de $G = \{\Sigma, V, S, P\}$, une grammaire CF. L'ordre définissant cette suite est quelconque. Soit, pour $w \in L(G)$, $n_i(w)$ = le nombre d'occurrences du symbole T_i dans l'arbre syntaxique dérivant w de S. Nous dirons qu'un algorithme $\mathcal{A}$ admettant en entrée $w \in L(G)$ est à fonction de coût si le nombre d'instructions élémentaires de $\mathcal{A}(w)$ est une fonction des nombres $n_i(w)$ uniquement.

1.2. Exemples

Considérons la grammaire arithmétique G_A. Nous supposerons que $T_1 = M_1$, $T_6 = M_6$ et nous noterons $n_{\mathcal{A}}(w) = \sum_{i/T_i \in \mathcal{A}} n_i(w)$. Les résultats suivants sont démontrés dans l'annexe 3 :

- L'algorithme d'analyse syntaxique séquentielle proposé par Hopcroft pour les mots de $L(G_A)$ admet une fonction de coût :

$$(11) \quad F_{SQ}(w) = \alpha_{SQ} + \gamma_{SQ} n_6(w) + \delta_{SQ} n(w)$$

- La procédure de segmentation des mots de $L(G_A)$ est exécutée en un nombre d'instructions élémentaires :

$$(12) \quad F_{SG}(w) = \alpha_{SG} + \beta_{SG} n_1(w) + \gamma_{SG} n_6(w)$$

- L'analyse syntaxique des mots de $L(G_i)$ i=2,3, réalisée au moyen d'une version simplifiée de l'algorithme de Hopcroft (Annexe 1) admet une fonction de coût :

$$(13) \quad F_i(w) = \alpha_i + \beta_i n_{\mathcal{A}}(w) \qquad i=2,3 \qquad w \in L(G_i)$$

- L'analyse syntaxique des mots de $L(G_1)$:

$$(14) \quad F_1(w) = \alpha_1 + \beta_1 n_1(w) \qquad w \in L(G_1)$$

2. Parallélisme ponctuel

2.1. Définition

Soit G une grammaire context-free non ambiguë et

un algorithme de segmentation des mots de L(G). Nous définissons le parallélisme de l'analyse syntaxique au point $w \in L(G)$ comme le rapport $p(w)=F_S(w)/F_{//}(w)$ du nombre d'instructions élémentaires , $F_S(w)$ nécessaires à son exécution séquentielle , sur le nombre d'instructions élémentaires nécessaires à son exécution parallélisée en environnement multiprocesseurs idéal (nombre infini de processeurs). La fonction dépend des algorithmes d'analyse syntaxique séquentiel et paralléle .

2.2. Exemple

Soit:

$w = \neq id+\sqrt{id}*(\sqrt{id}+id*(\sqrt{id}+id*(\sqrt{id}+id*id)))\neq \in L(G_A)$

$w' = \neq id+(\sqrt{id}+\sqrt{id}+\sqrt{id}+\sqrt{id})*(id+id)*(id-id)\neq \in L(G_A)$

Calculons le parallélisme relatif à l'algorithme séquentiel de Hopcroft (annexe I) et à l'algorithme parallèle défini au paragraphe 3.2 de la première section, dans sa version sans pipe-line. On obtient (en négligeant le coût des communications dans le réseau) :

$$(15)\quad p(w) = \frac{F_{SQ}(w)}{F_{SG}(w) + \sup_{i=1,k(w)} F_{e(i)}(w_i)}$$

Calculons le rapport $p(w)/p(w')$ pour les valeurs numériques suivantes : $\alpha_{SG}=0$, $\beta_{SG}=6$, $\gamma_{SG}=6$; $\alpha_i=0$, $\delta_i=17$, pour $i=2,3$; $\alpha_1=2$; $\beta_1=4$.

Les coûts d'analyse séquentielle $F_{SQ}(w)$ et $F_{SQ}(w')$ coïncident (les fonctions n_1, n_6, n_7 prennent la même valeur aux points w et w') si bien que :

$$\frac{p(w)}{p(w')} = \frac{F_{SG}(w') + \sup_{i=1,k(w')} F_{e(i)}(w'_i)}{F_{SG}(w) + \sup_{i=1,k(w)} F_{e(i)}(w_i)} = 1.9$$

Malgré des coûts en segmentation identiques ($F_{SG}(w) = F_{SG}(w')$), la délimitation par l'algorithme de segmentation d'un mot de grande taille ($w'_1 = (\sqrt{id}+\sqrt{id}+\sqrt{id}+\sqrt{id})$) dans w', réduit le parallélisme de w' à approximativement la moitié du parallélisme de w. Il apparaît dans cet exemple que les fluctuations dans la taille des mots déterminés par l'algorithme de segmentation, influent de manière déterminante sur le gain en temps d'exécution obtenu grâce à la parallélisation de l'analyse des mots du langage d'une même grammaire.

3. Processus de branchement [HAR 63]

3.1. Processus de branchement multitype

Considérons p distributions de probabilité sur N^p: $p^{(i)}(j_1,j_2,\ldots,j_p)$ $i=1,p$, $j_i \in N$, ainsi que les fonctions génératrices :

$$f^{(i)}(s_1,\ldots,s_p) = \sum_{j_1,j_2,\ldots,j_p \geq 0} p^{(i)}(j_1,\ldots,j_p) s_1^{j_1}\ldots s_p^{j_p}$$

$s_e^{j_e} \in \mathbb{C}$, $|s_e^{j_e}| \leq 1$

Notons $\vec{j} = (j_1,\ldots,j_p) \in N^p$, $\vec{s} = (s_1,\ldots,s_p) \in \mathbb{C}^p$

$\vec{f}(\vec{s}) = (f^{(1)}(\vec{s}),\ldots,f^{(p)}(\vec{s})) \in \mathbb{C}^p$.

3.2. Définitions et propriétés élémentaires

Un processus de branchement p-multiple de fonction génératrice $\vec{f}(\vec{s})$ est une chaîne de Markov $(Z_n, n=0,\ldots)$ sur N^p dont la fonction de transition est donnée par

$$(16)\quad P(\vec{i},\vec{j}) = P[Z_{n+1} = \vec{j} \mid Z_n = \vec{i}\,,\ i,j \in N^p]$$
$$= \text{coefficient de } s_1^{j_1}, s_2^{j_2}\ldots s_p^{j_p} \text{ dans } (f^{(1)}(\vec{s}))^{i_1}.(f^{(2)}(\vec{s}))^{i_2}\ldots(f^{(p)}(\vec{s}))^{i_p}$$

On remarquera que $\vec{0}$ est une état absorbant de la chaîne Z.

On notera $Z_{n.j}^{(\vec{i})}$ $i \in N^p$, $n \in N$, $j \in \{1,..,p\}$ le nombre des éléments de type j dans la n ième "génération" du processus Z lorsque $Z_0 = \vec{i}$.

Soit $M = (m_{ij})$ $i=1,p$ $j=1,p$ la matrice des moments

$$(17)\quad m_{ij} = \frac{\partial f^{(i)}(\vec{s})}{\partial s_j}\Big|_{(1,1,..,1)}$$

On montre la propriété suivante (avec des notions matricielles)

$$(18)\quad E[Z_n | Z_0] = Z_0.M^n;\ \text{soit } (m_{ij}^{(n)}) = M^n \quad i,j \in \{1,..,p\}$$

On dit que le processus Z est positif régulier si :

$$(19)\quad \exists n/\ \forall i,j \in \{1,..,p\},\ m_{ij}^{(n)} > 0$$

On montre de plus la propriété suivante sur les processus réguliers positifs non singuliers (Z est singulier si

$$f^{(i)}(\vec{s}) = \sum_{j=1}^{p} A_{ij}\, s_j \geq A_{ij}\ 0 \quad \forall i=1,..p) :$$

Théorème de Frobénius : si ρ, la plus grande valeur propre de M est inférieure à 1, la probabilité d'absorption est égale à 1 pour toute condition initiale.

3.3. Distribution de l'occurrence de chaque type jusqu'à l'absorption

Soit $W^i(\vec{s})$, la fonction génératrice de la distribution jointe du vecteur

$$\left(\sum_{n=0}^{\infty} Z_{n,j}^{(\vec{1}_i)}\right) \quad j=1,..,p \quad \text{où } \vec{1}_i = (0,..,i,0,..,0)$$

Dans le cas où $\rho < 1$, on démontre [GOO 49] que les fonctions $W^i(\vec{s})$ sont l'unique solution du système de point fixe :

$$(20)\quad W^i(\vec{s}) = s_i.f^i(\overrightarrow{W(\vec{s})}) \quad i=1,..,p$$

4. Grammaire stochastique [GON 78]

Soit G une grammaire C.F non ambiguë. Une loi de probabilité est définie sur les arbres syntaxiques associés aux phrases de L(G). Cette loi, définie sur la structure génératrice est obtenue à partir d'une probabilisation des productions multiples de la grammaire et conduit à la définition d'un processus de branchement multitype (ou grammaire stochastique). On dérive des équations de point fixe sur ce processus la distribution de probabilité du nombre d'occur-

rences de chacun des symboles du vocabulaire de G (terminaux et non terminaux) dans l'arbre syntaxique des mots de L(G), considéré comme événement aléatoire de la grammaire ainsi probabilisée. Ces distributions permettent de déterminer les propriétés analytiques des fluctuations dans la durée des analyses syntaxiques indépendantes délimitées par la segmentation de G, dans le cas d'analyses admettant des fonctions de coût.

4.1. Probabilisation des productions multiples

Soit $G = \{\Sigma, V, S, P\}$, $p=\text{card}(V)$, $n=\text{card}(V_N)$. Soit T_i, $i=1,p$ la suite des symboles du vocabulaire de G et P_i, $i=1,n$ l'ensemble des productions de P dont la partie gauche est T_i. Soit $t(i) = \text{card}(P_i)$ et $r_{i,e,k}$, $i=1,n$, $e=1,t(i)$, $k=1,p$ le nombre d'occurrences du symbole T_k dans la partie droite de la e ième production de P_i. Donnons nous enfin des nombres réels positifs $(q_{i,e}) = Q$, $i=1,n$, $e=1,t(i)$ satisfaisant $\sum_{e=1,t(i)} q_{i,e} = 1$.

La donnée du couple (G,Q) définit complètement un processus de branchement p-multitype (à chaque symbole de V est associé un type) de fonction génératrice :

$$(21)\begin{cases} f^i(S_1,\ldots,S_p) = \sum_{e=1}^{t(i)} q_{i,e} \prod_{k=1}^{p} S_k^{r_{i,e,k}} \\ \text{pour } i=1,n \\ f^i(S_1,\ldots,S_p) = 1 \\ \text{pour } i=n+1,p \end{cases}$$

Il est intéressant de noter que les fonctions génératrices $f^i(\vec{S})$ obtenues, s'expriment sous la forme de polynômes (en S_i $i=1,p$) de degré fini.

4.2. Exemple : probabilisation de la grammaire G_A

On se donne les nombres réels positifs $Q=(q_{i,e})$ suivants:

$$\begin{cases} (q_{1,e}) = (1) \\ (q_{2,e}) = (1-\rho,\rho) \\ (q_{3,e}) = (p^3_j)\ ,j=2,7\ ,\ \sum_j p^3_j = 1 \\ (q_{4,e}) = (p^4_{j,k,r})\ ,j=2,7\ ,\ k=3,7\ ,\ r=+,-\ ,\ \sum_{j,k,r} p^4_{j,k,r} = 1 \\ (q_{5,e}) = (p^5_{j,k,r})\ ,j=3,7\ ,\ k=4,7\ ,\ r=*,/\ ,\ \sum_{j,k,r} p^5_{j,k,r} = 1 \\ (q_{6,e}) = (p^6_{j,k})\ ,j=5,7\ ,\ k=4,7\ ,\ \sum_{j,k} p^6_{j,k} = 1 \\ (q_{7,e}) = (p^7_j)\ ,j=5,7\ ,\ \sum_j p^7_j = 1 \\ (q_{8,e}) = (p^8_j)\ ,j=2,7\ ,\ \sum_j p^8_j = 1 \\ (q_{9,e}) = (1) \end{cases}$$

On en deduit les fonctions génératrices suivantes (avec des notations évidentes) :

$$f^1(\vec{S}) = S_{M_0}.S_{\underline{\neq}}\ ,\ f^2(\vec{S}) = S_{\underline{\neq}}.S_{M_1}.(1-\rho+\rho.S_{M_0}) \ \ldots.$$

jusqu'à:

$$f^8(\vec{S}) = S_{\underline{(}}.S_{\underline{)}}.\left(\sum_{j=2}^{7} p^8_j.S_{M_j}\right)\ ,\ f^9(\vec{S}) = S_{\underline{id}}\ .$$

4.3. Consistance d'une grammaire stochastique

La probabilisation des productions multiples n'induit pas nécessairement une distribution de probabilité non défective sur les arbres de L(G). Pour qu'il en soit ainsi (on dit alors que la grammaire est consistante) il est nécessaire et suffisant que la condition de Frobénius(3.2) soit satisfaite par le processus de branchement associé à la grammaire stochastique. Ainsi,pour une grammaire consistante, le système de point fixe (20) ,vérifié par la fonction génératrice du vecteur $(n_1(w),n_2(w),..,n_p(w))$ admet une solution non défective et unique.

4.4 Calcul de la loi de $(n_1,n_2,\ldots,n_p)$

Les séries génératrices $W^i(\vec{S})$ sont solution du système (20) qui s'écrit sous la forme:

$$(22)\begin{cases} W^i(\vec{S}) = \sum_{\vec{n}\in N^p} p^i(\vec{n}).\vec{S}^{\vec{n}} \\ f^i(\vec{S}) = \sum_{\vec{m}\in D_i} a^i_{\vec{m}}.\vec{S}^{\vec{m}} \end{cases}$$

Le système s'écrit alors:

$$(23)\ \sum_{n\in N^p} p^i(\vec{n}).\vec{S}^{\vec{n}} = S_i.\left(\sum_{\vec{m}\in D_i} a^i_{\vec{m}}.\vec{W}^{\vec{m}}\right)\ ,i=1,p.$$

En égalant les coéfficients,on obtient:

$$(24)\ p^i(\vec{n}) = \sum_{\vec{m}\in D_i} C(\vec{n}-\vec{1}_i,\vec{W}^{\vec{m}})\ ,i=1,p$$

Où $C(\vec{q},\vec{W}^{\vec{m}})$ est le coéfficient de $\vec{S}^{\vec{q}}$ dans $\vec{W}^{\vec{m}}$.

Lemme 2: Le système de point fixe (20) se résoud par récurrence sur les $p^i(\vec{n})$.(Voir l'annexe 4)

5.Evaluation des performances d'algorithmes d'anlyse syntaxique paralléles

5.1.Parallélisme intrinséque à l'analyse syntaxique d'une grammaire ségmentable

Etant donnés la grammaire stochastique (G,Q),un algorithme de ségmentation de fonction de coût F_{SG}, une collection d'algorithmes d'analyse syntaxique pour les grammaires G_i générées par la famille W (section 1) de fonctions de coût F_i,$i=1,n$ et un algorithme d'analyse syntaxique séquentiel des mots w de L(G),de fonction de coût F_{SQ}, on appelle parallélisme intrinséque à l'analyse de ^{SQ}G le rapport:

$$(25)\ P = \frac{E(F_{SQ}(w))}{E(F_{SG}(w)+\max_{k=1,n}\ \max_{j=1,m_k} F_k(w_{j,k}))}$$

où :

$w_{j,k}\in L(G_k)$ est le j iéme mot de type k défini par la ségmentation de w.

m_k est le nombre de mots de type k définis par la ségmentation de w.

5.2. Exemple de calcul du parallélisme intrinséque

Ce paragraphe est consacré au calcul de P dans l'exemple introduit au paragraphe 2.1.

5.2.1. Calcul de $E(F_{SQ})$ et $E(F_{SG})$

Connaissant la loi jointe de (n_1, n_6, n_c) (paragraphe 4.4) ,on obtient $E(F_{SQ})$ à partir de l'équation (11) et $E(F_{SG})$ à partir de l'équation (12).

5.2.2. Calcul de $E(\max F_k(w_{j,k}))$

Les variables aléatoires $w_{j,k}$,k=1,3 ,j=1,m_k sont liées par les contraintes suivantes:

$$(26)\quad \begin{cases} \sum_{k=2}^{3} \sum_{j=1}^{m_k(w)} n_6(w_{j,k}) = m_3(w) \\ \sum_{j=1}^{m_1(w)} n_1(w_{j,1}) = m_2(w) \quad , m_1(w) = 1 \ . \end{cases}$$

La décomposition ci dessous permet de calculer la loi du maximum des variables liées $F_k(w_{j,k})$:

$$P[\max_{k=1,3} \max_{j=1,m_3} F_k(w_{j,k}) \le N] = \sum_{l_2 \ge 0} \sum_{l_3 \ge 0}$$

$$P(m_2(w)=l_2 \ , \ m_3(w)=l_3) \times P[\max_{k=1,3} \max_{j=1,l_k} F_k(w_{j,k}) \le N]$$

$$P[\max_{k=1,3} \max_{j=1,l_k} F_k(w_{j,k}) \le N] = P[F_1(w_{1,1}) \le N] \times$$

$$P[\max_{k=2,3} \max_{j=1,l_k} F_k(w_{j,k}) \le N].$$

$$P[\max_{k=2,3} \max_{j=1,l_k} F_k(w_{j,k}) \le N] = \sum_{q=0}^{l_3} P[\sum_{j=1}^{l_2} n_6(x_j)=q] \times$$

$$P[\sum_{j=1}^{l_2} n_6(y_j)=l_3-q] \times P[(\max F_2(x_j) \le N) \mid \sum_{j=1}^{l_2} n_6(x_j)=q]$$

$$\times P[\max_{j=1,l_3} F_3(y_j) \le N \mid \sum_{j=1}^{l_3} n_6(y_j)=l_3-q].$$

Où les x_j ,j=1,l_2 (resp. les y_j ,j=1,l_2) sont des réalisations indépendantes de mots de $L(G_2)$ (resp. $L(G_3)$).On peut calculer par récurrence chacun des facteurs du produit dans la dernière expréssion (voir l'annexe 4).D'où le principe de la procédure numérique permettant le calcul de $E(\max F_k(w_{j,k}))$.

5.2.3. Résultats numériques

Le parallélisme intrinséque est calculé (fig4) pour des familles croissantes de générateurs de sous grammaires: $W_1=\{S\}$, $W_2=\{S,M_1\}$, $W_3=\{S,M_1,M_6\}$,et pour deux probabilisations de G_A :Q_1 et Q_2.(Voir l'annexe 5 pour les valeurs numériques)

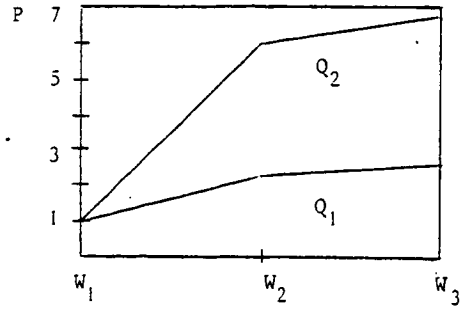


Fig.4.

SECTION III - La réalisation

1. Introduction

Les travaux concernant la réalisation ont porté sur l'implémentation d'algorithmes de contrôle à structure pipe-line, difficiles à modéliser directement, mais qui sont succeptibles d'apporter un gain notable en temps de calcul.

1.1. Le réseau utilisé

Nous disposons de trois APPLE II, interconnectés par un réseau en boucle. Ce réseau fait appel à un algorithme du jeton pour synchroniser les émissions et réceptions de messages entre les processeurs. Le débit utile est de 50 Kbits/s, par paquets d'une taille maximale de 1024 bits. De plus les transmissions sont transparentes vis à vis de chaque APPLE, du fait de l'utilisation de communicateurs spécialisés.

1.2. L'algorithme d'analyse syntaxique parallèle implémenté

La version pipe-line du schéma d'analyse parallèle des mots de $L(G_A)$ proposé dans la première section au paragraphe 4.2 a été implémenté sur le réseau. L'algorithme de segmentation est celui du paragraphe 3.2. L'algorithme d'analyse syntaxique de w_j dans $L(G_{e(j)})$, j=1,k(w), est une version simplifiée de l'algorithme de Hopcroft (la fonction NEST n'est pas nécessaire). L'algorithme séquentiel de référence est celui de Hopcroft.

2. Les différents types de contrôle

2.1. Algorithme d'allocation statique

Si le processeur central peut, à chaque instant évaluer la quantité de travail reçue, et non encore exécutée, par chaque processeur périphérique, il peut allouer de façon optimale (dans le sens où le processeur central choisit le périphérique le moins occupé) les tâches qu'il détecte au cours de la segmentation.

Cette méthode nécessite la connaissance du coût en temps de chaque tâche, de façon à pouvoir estimer la quantité de travail sur chaque site.

2.2. Algorithme d'allocation dynamique

L'allocation se fait de la manière suivante : une file d'attente de sous analyses syntaxiques est implémentée au niveau du processeur central. Dès qu'un processeur périphérique est libre, il sert la première des tâches en attente.

3. Les mesures

3.1. Résultats numériques (deux processeurs périphériques)

Le tableau suivant résume les résultats obtenus :

configuration	Gain en % du temps de calcul en séquentiel
algorithme statique découpage $\neq$	10 %
algorithme dynamique découpage $\neq$	33 %
algorithme dynamique découpage $\neq$, (,)	20 %

3.2. Commentaires

Le gain en temps pour l'allocation statique est faible : ceci est imputable au fait que l'on doit gérer sur chaque périphérique une file d'attente de travaux en entrée et en sortie, ce qui ralentit sensiblement le temps de réponse.

Cette méthode ne doit donc être utilisée que lorsque les coûts de communication sont très importants par rapport au temps de calcul (ce qui n'est pas le cas dans notre application puisque l'on peut estimer le coût de transmission sur le canal à moins de 1% du temps de réponse total).

L'avantage comparé de l'algorithme dynamique réside dans la suppression des tâches de gestion de files d'attente de travaux, sur les périphériques (ce qui se traduit par un gain en place mémoire et une simplification du logiciel d'échange). Cependant, les temps de communication ne sont plus transparents vis à vis du temps d'analyse syntaxique.

Cette seconde méthode est donc intéressante dans le cas où le processeur central est relativement peu occupé et peut répondre de façon rapide aux demandes des processeurs périphériques, lorsque les temps de transmission sont négligeables devant les temps de calcul.

En ce qui concerne la finesse optimale de segmentation, on remarquera que, pour l'algorithme dynamique, la segmentation la plus fine n'est pas la plus performante lorsqu'on dispose seulement de deux propesseurs périphériques (les résultats théoriques de la section II laissent néanmoins penser que l'accroissement du nombre des périphériques rendra plus intéressante la segmentation fine). De manière générale, aucune optimisation n'a encore été effectuée sur les procédures implémentées, ce qui laisse penser que des améliorations sensibles sont envisageables, même sur un petit nombre de processeurs.

ANNEXE 1 -

Exemple : analyse syntaxique par fonction de précédence de la grammaire arithmétique G_A

Nous rappelons les principes d'une analyse syntaxique de G_A au moyen d'une fonction de précédence ainsi qu'un algorithme de Hopcroft fondé sur ces principes. Les résultats de ce paragraphe sont détaillées dans FIS 80.

Soit $\mathcal{O}$ l'ensemble des opérateurs de G :

$$= (\underline{\neq}, \underline{+}, \underline{-}, \underline{*}, \underline{/}, \underline{**}, \underline{\surd})$$

Deux fonctions constantes de $\mathcal{O}$ sur $N^{|\mathcal{O}|}$ sont définies, caractérisant respectivement l'associativité (gauche ou droite) et la priorité de chaun des opérateurs

$$\begin{aligned} ASSOC(\mathcal{O}) &= (-1,-1,-1,-1,-1,+1,+1) \\ &= (assoc(\neq), \ldots, assoc(\surd)) \\ PRIO(\mathcal{O}) &= (0,1,1,2,2,3,4) \\ &= (prio(\neq), \ldots prio(\surd)) \end{aligned}$$

Lorsque $w \in L(G_A)$ s'écrit $w = x_1\theta_1 x_2\theta_2 x_3 \ldots x_q\theta_q x_{n+1}$ où $\theta_i \in \mathcal{O}$ et $x_i \in (\Sigma-\mathcal{O})^*$, on définit deux fonctions de $w \to N^q$ caractérisant respectivement la position et le niveau de parenthèsage de chacun des opérateurs dans w :

$$POS(w) = (pos(\theta_1), \ldots pos(\theta_q))$$

$$NEST(w) = (nest(\theta_1), \ldots nest(\theta_q))$$

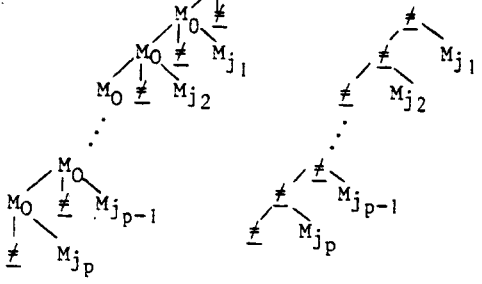
avec $pos(\theta_i) = i$ et $nest(\theta_i)$ = nombre de parenthèses ouvrantes − nombre de parenthèses fermantes dans $x_1, x_2, \ldots, x_i$, $i=1,q$.

Considérons l'arbre de dérivation réduit (dont les noeuds sont soit des opérateurs soit des identificateurs), construit à partir de l'arbre de dérivation de w dans $L(G_A)$ comme suit:

-remplacer tout sous arbre " M_6 / (M_j) " par " M_j ".
pour j=2,7.

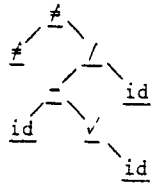
-remplacer tout sous arbre " M_1 / M_j " par " M_j ".

-remplacer le sous arbre " S " par



-remplacer tout M_j j=2,5 par l'opérateur en lequel il se dérive et tout M_7 par id.

L'arbre de dérivation réduit de "$\underline{\neq(id-\surd id)/id\neq}$" est :



Soit PREC $\triangleq$ $\{prec(\theta_1), \ldots, prec(\theta_q)\}$ la fonction de w dans N^q définie par :

$$\begin{aligned} prec(\theta_i) &= pos(\theta_i) * assoc(\theta_i) \\ &\quad + 2 * (q+1) * prio(\theta_i) \\ &\quad + 2 * (q+1) * |\mathcal{O}| * nest(\theta_i) \end{aligned}$$

La fonction ainsi définie permet de calculer l'arbre réduit de w comme le montre le théorème suivant :

Théorème pour $1 \le i \le q$,

Soit $SR(i) \triangleq \{j / i < j \le q / \nexists k / i < k \le j$ et $prec(\theta_k) < prec(\theta_i)\}$,

le plus grand ensemble des positions qui suivent i dont les opérateurs ont une précédence supérieure à celle de θ_i. Si $SR(i) \neq \emptyset$, soit $R(i) = \{j \in SR(i) / \forall k \in SR(i) / prec(\theta j) \le prec(\theta_k)\}$, la position de SR(i) dont l'opérateur est de précédence minimale. Alors, R(i) est la position de l'opérateur racine du sous arbre droit de θ_i dans l'arbre réduit de w. Si $SR(i) = \emptyset$, θ_i n'a pas de sous arbre droit, R(i)=0.

(un théorème analogue permet de déterminer $L(i)$, la position de la racine du sous arbre gauche de θ_i).

Nous présentons un algorithme de Hopcroft, permettant un calcul efficace des fonctions PREC, $L \triangleq (L(1),\ldots,L(q))$, $R \triangleq (R(1),\ldots,R(q))$.

Algorithmes de Hopcroft :

Soit $w \in L(G_A)$, $w = w(1)w(2)\ldots w(|w|)$ où $w(i)$. La procédure suivante permet de calculer la fonction de précédence de w :

```
PROC PRECEDENCE ;
    J ← 0; MP ← 0;
    pour I de 1 a |w| :
        si w(I) = (: MP ← MP + 1
        sinsi w(I) = ) : MP ← MP - 1
        sinsi w(I) ∈ Θ : J ← J + 1 ;
                oper(J) ← w(I) ;
                nest(J) ← MP
        sinon : rien
        fsi
    fpour
    pour I de 1 a J : Prec(I) ← I * assoc(OPER(I))
                + 2 * (J+1) * prio(OPER(I) + 2 * (J+1)
                * (|Θ|+1) * nest(I)
    fpour
fproc
```

L'algorithme pour le calcul du vecteur des fils droits R utilise un automate à pile admettant en entrée la suite $(\Theta_1,\ldots,\Theta_q)$:

```
PROC FILS DROIT ;
prec(OPER(a)) = -∞ ; empile a ; POINTEUR ← 1;
    tant que POINTEUR ≤ q :
    si : prec(OPER(SOMMET PILE)) ≤ prec(OPER(POIN-
                TEUR)) : empile(POINTEUR); POIN-
                TEUR ←POINTEUR + 1 ;
        sinon : R(PILE) ← PREV;
                PREV ← PILE;
                dépile;
        fsi
    ftantque
    fproc
```

ANNEXE II

Démonstration du lemme I

Soit w fixé $\in L(G)$. Considérons, dans la procédure de décomposition de w, la suite d'éléments de w : $U_1=\phi_W(\alpha_{k-1})$, $U_2=\phi_W(\alpha_{k-2})$ ..., $U_k=\phi_W(\alpha_o)=S$.

Lorsque les noeuds de l'arbre A(w) sont parcourus dans l'ordre Postfixé (Fils gauche, Fils droit, Racine), $(U_1,U_2,\ldots,U_k)$ est la suite des noeuds de A(w) dont l'étiquette appartient à W. De plus, on peut associer à chaque symbole U_i, i=1, k-1, un terminal unique de A(w), $s(U_i) \in \{\neq,)\}$, que nous appellerons délimiteur droit de U_i :

Si $U_i=M_6$, $s(u_i)$ est le symbole ")", fils droit de U_i dans A(w).

Si $U_i=M_1$, $s(U_i)$ est le symbole "$\neq$", frère droit de la racine de M_1.

On montre aisément que le délimiteur droit $s(U_{i+1})$ est à droite du délimiteur droit $s(U_i)$ dans w, i=1 i=1,k-2. Considérons maintenant la procédure de segmentation de w. Soit $\hat{\alpha}_i$ le contenu de la pile, concaténé avec la suite des symboles à droite du pointeur (pointeur inclus), juste après la réduction de $\hat{w}_i$ en $V_{\hat{e}(i)}$. Nous démontrons ci-dessous par récurrence que $\hat{\alpha}_j=\alpha_{k-j}$; $\hat{e}(j)=e(k-j+1)$; $\hat{w}_j=w_{k-j+1}$, pour j=1,k. (ce qui implique $\hat{k}=k$) :

j=1 : supposons w tel que e(k)=3. Donc $\phi_W(\alpha_{k-1})=M_6$ et $\alpha_k=w=g(\alpha_{k-1})w_k d(\alpha_{k-1})$, où w_k est de la forme (γ) où $\gamma \in (\Sigma-D)^*$ et $)=s(U_1)$, le premier délimiteur droit rencontré dans w, lu de gauche à droite, mis à part le premier $\neq$. Ainsi, la segmentation de w détermine bien $\hat{e}(1)=e(k)$, $\hat{w}_1=w_k$ et donc $\hat{\alpha}_1=\alpha_{k-1}$. La preuve est similaire pour e(k)=1.

Supposons l'hypothèse de récurrence vérifiée jusqu'à j-1 < k-1. Supposons que w est tel que e(k-j+1)=3. Donc $\phi_W(\alpha_{k-j})=M_6$ et $\alpha_{k-j+1}=\beta_{j-1}=g(\alpha_{k-j})w_{k-j+1}d(\alpha_{k-j})$, où w_{k-j+1} s'écrit sous la forme (γ), où $s(U_j)=)$ est lej-ième symbole {≠ ou)} de w et $\gamma \in (V-D)^*$; on en déduit que $\hat{e}(j)=e(k-j+1)$, $\hat{w}(j)=w(k-j+1)$ et donc, $\beta_j=\alpha_{k-j}$. Preuve similaire pour e(k-j+1)=1. CQFD

ANNEXE III -

Exemples d'algorithmes à fonction de coût

1. La procédure "PRECEDENCE" admet une fonction de coût

La première phase de l'algorithme (calculant OPER et NEST) admet une fonction de coût puisqu'elle consiste en une lecture séquentielle des terminaux de w et que le traitement effectué ne dépend que du type de symbole. Sa fonction de coût, f_1, est de la forme

$$f_1(w) = a + bn_6(w) + cn_\Theta(w)$$

De même pour la deuxième phase de l'algorithme, qui demande le même traitement en chacun de ses pas et s'exécute en un nombre de pas égal à $n_\Theta(w)$ le nombre des opérateurs de w. Soit

$$f_2(w) = e.n_\Theta(w)$$

2. La procédure "FILS DROIT" admet une fonction de coût

Tout opérateur est empilé puis dépilé, donc on effectue pour chacun une et une seule fois la somme des traitements des deux hypothèses du test, c'est-à-dire un nombre constant d'instructions. La fonction de coût de cet algorithme, f_3, est donc :

$$f_3(w) = d.n_\Theta(w) .$$

Ainsi, l'algorithme séquentiel global d'analyse syntaxique des mots de $L(G_A)$ admet une fonction de coût linéaire en $(n_6(w),n_\Theta(w))$.

3. Procédure de segmentation

L'argument d'entrée de la procédure est une suite alternée de m(w) "Γ" et m(w) "d". La procédure crée, au cours de son exécution $n_6(w)$ "M_6" et $n_1(w)$ "M_1". Outre ces créations, la procédure empile et dépile une seule fois chaque symbole Γ,d,M_6,M_1. D'où le résultat...

ANNEXE IV -

Démonstration du lemme 2

$p^i(o,o,\ldots,o)=0$. $\forall$ i=1,...,p. Supposons calculé

tous les $p^i(\vec{n})$ $i=1,\ldots,p$ pour $\vec{n} \leq \vec{N}-\vec{1}_j=(N_1,..,N_j-1,\ldots,N_p)$

- si $i=j$ on voit que :

$$p^i(\vec{n}) = \sum_{\vec{m} \in D_i} a^i_{\vec{m}} . C(\vec{n}-\vec{1}_i, \vec{W}^{\vec{m}})$$

$$= \sum_{\vec{m} \in D_i} a^i_{\vec{m}} . C(\vec{n}-\vec{1}_j, \vec{W}^{\vec{m}})$$

qui se calcule donc directement $\forall\ \vec{n} \leq \vec{N}$, $n_j=N_j$.

- si $i \neq j$ on peut écrire :

$$p^i(\vec{n}) = \sum_{\vec{m} \in D_i} a^i_{\vec{m}} C(\vec{n}-\vec{1}_i, \vec{W}^{\vec{m}})$$

$$= \sum_{\vec{m} \in D_i} a^i_{\vec{m}} C(\vec{n}-\vec{1}_i, W^{k\vec{m}-\vec{1}_k})$$

$$= \sum_{\vec{m} \in D_i} a^i_{\vec{m}} \sum_{\vec{\ell}=0}^{\vec{n}-\vec{1}_i} p^k(\vec{\ell}) . C(\vec{n}-\vec{1}_i-\vec{\ell}, \vec{W}^{\vec{m}-\vec{1}_k})$$

et D_i est borné donc la décomposition de $C(\vec{n}-\vec{1}_i, \vec{W}^{\vec{m}})$ est finie.

On remarque que :

* si $\vec{\ell}$ est tel que $\ell_j=N_j$ alors $(\vec{n}-\vec{1}_i-\vec{\ell})_j=0$ et donc $C(\vec{n}-\vec{1}_i-\vec{\ell}, \vec{W}^{\vec{m}-\vec{1}_k})$ peut se calculer directement

* si $\vec{\ell}$ est tel que $0 < \ell_j < M_j$ alors de même $\vec{0} < (\vec{n}-\vec{1}_i-\vec{\ell})_j < N_j$ et $C(\vec{n}-\vec{1}_i-\vec{\ell}, \vec{W}^{\vec{m}-\vec{1}_k})$ se calcule directement

* enfin si $\ell_j=0$ alors on multiplie $C(\vec{n}-\vec{1}_i-\vec{\ell}, \vec{W}^{\vec{m}-\vec{1}_k})$ par $p^k(\vec{\ell})$ qui est connu.

donc $p^i(\vec{n})$ s'exprime sous une forme linéaire en fonction des $p^k(\vec{\ell})$ tels que $\ell_j=N_j$.

Si on appelle X le vecteur des $(p^i(\vec{n}))_{i=1,p;n_j=N_j}$, le système s'écrit sous la forme matricielle $AX=B$, que l'on peut résoudre dans le cas général par la méthode de gauss. Des méthodes similaires sont utilisées pour les calculs numériques du paragraphe 5.2.2, section II.

ANNEXE V -

Les fonctions de coût sont les suivantes :

$F_{SQ}(w) = 5 + 10n_6 + 40\ n_{\mathcal{O}}$

$F_{SG}(w) = 10 + 7n_6 + 5\ n_1$

$F_2(w) = 5 + 40\ n_{\mathcal{O}} + 5\ n_6 = F_3(w)$.

Les probabilisations sont les suivantes :

$f^0(s) = (1-\rho) + \rho S_1$, $f^1(s) = 0,4\ S_2 + 0,6\ S_3$,

$f^2(s) = 0,7 + 0,15\ S_3 + 0,15\ S_3.S_4$,

$f^3(s) = 0,4 + 0,3\ S_3 + 0,1\ S_4 + 0,1\ S_5 + 0,1\ S_6^2$,

$f^4(s) = 1$, $f^5(s) = S_6$, $f^6(s) = S_2$. avec

Q_1 : $\rho=0,5$, Q_2 : $\rho=0,9$.

REFERENCES

[BAE 77] J.L. BAER, C.S. ELLIS : "Model, design, and evaluation of a compiler for a parallel processing environment", IEEE soft. Eng. Vol. SE.3, N°6, November 1977.

[BAN 79] J.P. BANATRE, J.P. ROUTEAU, L. TRILLING : " An event driven compiling technique", Communications of the ACM, January 1979, Vol. 22, N° 1.

[FIS 75] C.M. FISCHER : "On parsing and compiling Arithmetic expressions in parallel computational environment", Technical Report ## 243, Wisconsin Madison University 1975.

[GON 78] R.C. GONZALES, M.G. THOMASON : "Syntactic pattern recognition", Addison Wesley 1978.

[GOO 49] I.J. GOOD : "The number of individuals in the cascade process", Proc. Camb., Phil. Soc. 45, 1949, 360-363.

[GRI 71] D. GRIES : "Compiler construction for digital computers", 1971, John Wiley & sons, New York.

[HAR 63] T.E. HARRIS : "The theory of branching processes", 1963, Springer, Berlin.

[HOP 69] J.E. HOPCROFT et J.D. ULLMAN : "Formal languages and their relation to automata", ADDISON-WESLEY 1969.