Passage d'une expression rationnelle à un automate fini non-déterministe

Djelloul Ziadi Jean-Luc Ponty Jean-Marc Champarnaud

Résumé

Le but de cet article est de présenter un nouvel algorithme séquentiel en temps $\Omega(n^2)$ pour la conversion d'une expression rationnelle simple, ayant n occurrences de symboles, en son automate de Glushkov (automate d'états finis, non nécessairement déterministe, sans ε-transitions, ayant $n+1$ états, représenté par sa table de transitions).

Les algorithmes produits par A. Brüggemann-Klein et par C.H. Chang et R. Paige évaluent les opérations de fermeture (étoile de Kleene) en unions (d'ensembles de transitions) disjointes, selon les deux variantes suivantes :

$$\delta \uplus \mathtt{A} = (\delta \backslash (\delta \cap \mathtt{A})) \cup \mathtt{A} = \delta \cup (\mathtt{A} \backslash (\delta \cap \mathtt{A}))$$

A. Brüggemann-Klein introduit la notion de "*Star Normal Form*" (forme normale de fermeture) et calcule "*au plus tard*" les transitions induites par un opérateur de fermeture. Chang et Paige évaluent de façon paresseuse la fonction de transitions avec une structure de données appelée CNNFA (Compressed Normal NFA) à base de forêts d'états.

À complexité temporelle égale, notre algorithme est intéressant pour les raisons suivantes. Tout d'abord il repose sur une représentation originale de l'automate de Glushkov, basée sur des forêts d'états différentes de celles de Chang et Paige. Le passage de l'expression à cette représentation de l'automate est en temps linéaire par rapport au nombre d'occurrences de lettres n; la traduction en table de transitions est en $\Omega(n^2)$. L'évaluation des opérations de fermeture sur cette représentation est réalisée en unions disjointes par un algorithme récursif original en temps linéaire par rapport à n. Par ailleurs, cet algorithme séquentiel se prête bien à la parallélisation dans le modèle PRAM.

Received by the editors May 95.
Communicated by M. Boffa.
1991 *Mathematics Subject Classification :* 68-02, 68-06, 68P05, 68Q68.
Key words and phrases : Automates, expressions rationnelles, automates de Glushkov.

1 Introduction

Dans le cadre du développement du logiciel AUTOMATE [7][8] nous cherchons à obtenir des algorithmes performants, notamment pour la conversion d'une expression rationnelle en un automate fini. B. W. Watson [11] a établi une taxinomie des algorithmes de construction d'un automate fini à partir d'une expression rationnelle. L'automate obtenu, selon les algorithmes, est ou n'est pas déterministe et contient ou non des ε-transitions. Dans cet article nous nous intéresserons aux algorithmes de construction de l'automate de Glushkov qui est une représentation naturelle d'une expression rationnelle [4].

Soit E une expression rationnelle, on note s la taille de E (en notation postfixée) et n le nombre d'occurrences de symboles dans E. On peut sans perte de généralité (et sans modifier la complexité des algorithmes étudiés) considérer que deux opérateurs de fermeture ne peuvent apparaître dans la relation père-fils dans l'arbre syntaxique de E. Il est aisé de voir que, sous cette hypothèse, s et n sont du même ordre de grandeur. Par conséquent, nous exprimerons les complexités des algorithmes en fonction de n.

La construction directe de l'automate de Glushkov d'une expression rationnelle E est en $O(n^3)$ [10]. Récemment A. Brüggemann-Klein [6] a montré que cette complexité peut être ramenée à $O(n^2)$. Son algorithme consiste à transformer l'expression E en une expression $E^\bullet$ appelée forme normale de fermeture de E à partir de laquelle l'algorithme de Glushkov fournit l'automate de E en $O(n^2)$. La structure de données utilisée par A. Brüggemann-Klein pour représenter un ensemble est une liste chaînée. Plus récemment Chang et Paige [9] ont adopté une structure de données appelée CNNFA (compressed normal NFA) qu'ils utilisent pour simuler la fonction de transition δ. Ainsi une représentation de l'automate de Glushkov peut être calculée en $O(n)$ en utilisant un espace mémoire en $O(n)$.

En premier lieu nous présenterons la terminologie et les conventions d'écriture utilisées dans ce rapport, ainsi que certaines définitions utiles. Par la suite, nous verrons l'algorithme tel qu'il a été décrit par Glushkov [10] [6]. En troisième lieu il s'agit de présenter les améliorations apportées par A. Brüggemann-Klein puis par Chang et Paige, d'en fournir une interprétation, pour ensuite les comparer. Enfin nous présenterons un nouvel algorithme qui utilise la structure de l'arbre syntaxique de l'expression E pour construire une représentation originale de l'automate de Glushkov, basée sur des forêts d'états. L'intérêt de cette structure de données est qu'elle est calculée en temps $O(n)$ grâce à un algorithme récursif adapté; la table de transitions s'en déduit en $\Omega(n^2)$.

2 Définitions et conventions d'écriture

Dans cette section, nous introduisons la terminologie et les notations utilisées dans cet article. A quelques exceptions près ces notations peuvent être trouvées dans [6].

Soit Σ un ensemble fini non vide de symboles, appelé alphabet. Σ^* représente l'ensemble des mots sur Σ. Le mot vide est noté ε. Les symboles de l'alphabet sont représentés par les premières lettres minuscules de l'alphabet telles que a, b, c... L'union ($\cup$), le produit ($\cdot$) et la fermeture (*) sont les opérations rationnelles classiques sur les parties de Σ^*.

Définition 2.1 [5] Les langages rationnels de Σ^* sont les éléments de la plus petite famille, fermée pour les trois opérations rationnelles, qui contient les singletons et le langage vide.

Définition 2.2 [2] Une expression rationnelle sur un alphabet Σ est définie récursivement comme suit :

- $\emptyset$ est une expression rationnelle qui représente l'ensemble vide.
- ε est une expression rationnelle qui représente le langage $\{\varepsilon\}$.
- $\forall a \in \Sigma, a$ est une expression rationnelle qui représente le langage $\{a\}$.
- Si F et G sont deux expressions rationnelles qui représentent respectivement les langages $L(F)$ et $L(G)$ alors :
  - l'expression rationnelle $(F) + (G)$ représente le langage $L(F) \cup L(G)$,
  - l'expression rationnelle $(F)\cdot(G)$ représente le langage $L(F)\cdot L(G)$,
  - l'expression rationnelle $(F)^*$ représente le langage $\bigcup_{i=0}^{\infty} L^i(F)$. [1]

Afin de préciser la position des symboles dans l'expression, les symboles sont indicés dans l'ordre de lecture. Par exemple à partir de l'expression $(a + \varepsilon).b.a$ on obtient l'expression indicée $(a_1 + \varepsilon).b_2.a_3$; les indices sont appelés *positions* et sont représentés par les dernières lettres minuscules de l'alphabet telles que x, y, z. L'ensemble des positions pour une expression E est noté $Pos(E)$. Si F est une sous-expression de E, on note $\mathrm{Pos}_E(F)$ le sous-ensemble des positions de E qui sont positions de F [2]. χ est l'application qui fait correspondre à chaque position de $Pos(E)$ le symbole de Σ qui apparaît à cette position dans E. A toute expression E on associe bijectivement une expression indicée $\overline{E}$. On notera $\sigma = \{\alpha_1, ..., \alpha_n\}$, où $n = |Pos(E)|$, l'alphabet de $\overline{E}$.

L'automate de Glushkov

Afin de construire un automate d'états finis non déterministe (abbr. AFN) reconnaissant $L(E)$, Glushkov a établi trois fonctions que l'on peut définir de la façon suivante sur l'expression indicée $\overline{E}$:

Définition 2.3 $First(\overline{E}) = \{x \in Pos(E) \mid \exists u \in \sigma^* : \alpha_x u \in L(\overline{E})\}$

$First(\overline{E})$ représente l'ensemble des positions initiales des mots du langage $L(\overline{E})$.

Définition 2.4 $Last(\overline{E}) = \{x \in Pos(E) \mid \exists u \in \sigma^* : u\alpha_x \in L(\overline{E})\}$

$Last(\overline{E})$ représente l'ensemble des positions finales des mots du langage $L(\overline{E})$.

---

[1] *avec* $L^0(F) = \{\varepsilon\}$

[2] *pour* $E = F + G$ *et* $E = F\cdot G$, $Pos_E(F) \cap Pos_E(G) = \emptyset$; *pour* E^*, $Pos(E^*) = Pos(E)$

Définition 2.5 $Follow(\overline{E}, x) = \{y \in Pos(E) \mid \exists v \in \sigma^*,\ \exists w \in \sigma^* : v\alpha_x\alpha_y w \in L(\overline{E})\}$

$Follow(\overline{E}, x)$[3] représente l'ensemble des positions qui suivent immédiatement la position x dans l'expression E.

Par abus de langage, on notera $First(E)$ pour $First(\overline{E})$, $Last(E)$ pour $Last(\overline{E})$ et $Follow(E, x)$ pour $Follow(\overline{E}, x)$.

Définition 2.6 On définit par récurrence la fonction $Null_E$ qui vaut $\{\varepsilon\}$ si ε appartient au langage $L(E)$ et $\emptyset$ sinon.

$$
\begin{array}{lcll}
Null_\emptyset & = & \emptyset & (2.61)\\
Null_\varepsilon & = & \{\varepsilon\} & (2.62)\\
Null_a & = & \emptyset & (2.63)\\
Null_{F+G} & = & Null_F \cup Null_G & (2.64)\\
Null_{F\cdot G} & = & Null_F \cap Null_G & (2.65)\\
Null_{F^*} & = & \{\varepsilon\} & (2.66)
\end{array}
$$

Notation 2.7 Pour tout ensemble X, on note $\mathcal{I}_X$ la fonction de X dans $\{\{\varepsilon\}, \emptyset\}$ telle que :

$$\mathcal{I}_X(x) = \begin{cases} \emptyset & si\ x\ \notin X \\ \{\varepsilon\} & si\ x\ \in X \end{cases}$$

Proposition 2.8 *First peut être calculée récursivement comme suit :*

$$
\begin{array}{lcll}
First(\emptyset) & = & \emptyset & (2.81)\\
First(\varepsilon) & = & \emptyset & (2.82)\\
First(\alpha_x) & = & \{x\} & (2.83)\\
First(F+G) & = & First(F) \cup First(G) & (2.84)\\
First(F\cdot G) & = & First(F) \cup Null_F\cdot First(G) & (2.85)\\
First(F^*) & = & First(F) & (2.86)
\end{array}
$$

Proposition 2.9 *Last peut être calculée récursivement comme suit :*

$$
\begin{array}{lcll}
Last(\emptyset) & = & \emptyset & (2.91)\\
Last(\varepsilon) & = & \emptyset & (2.92)\\
Last(\alpha_x) & = & \{x\} & (2.93)\\
Last(F+G) & = & Last(F) \cup Last(G) & (2.94)\\
Last(F\cdot G) & = & Last(G) \cup Null_G\cdot Last(F) & (2.95)\\
Last(F^*) & = & Last(F) & (2.96)
\end{array}
$$

[3] $x \notin Pos(E) \Rightarrow Follow(\overline{E}, x) = \emptyset$

Proposition 2.10 *$Follow(E, x)$ peut être calculée récursivement comme suit :*

$$\begin{array}{lcll}
Follow(\emptyset, x) & = & \emptyset & (2.101)\\
Follow(\varepsilon, x) & = & \emptyset & (2.102)\\
Follow(a, x) & = & \emptyset & (2.103)\\
Follow(F+G, x) & = & \mathcal{I}_{Pos(F)}(x)\cdot Follow(F, x) \cup \mathcal{I}_{Pos(G)}(x)\cdot Follow(G, x) & (2.104)\\
Follow(F\cdot G, x) & = & \mathcal{I}_{Pos(F)}(x)\cdot Follow(F, x) & \\
 & & \cup \mathcal{I}_{Pos(G)}(x)\cdot Follow(G, x) & \\
 & & \cup \mathcal{I}_{Last(F)}(x)\cdot First(G) & (2.105)\\
Follow(F^*, x) & = & Follow(F, x) \cup \mathcal{I}_{Last(F)}(x)\cdot First(F) & (2.106)
\end{array}$$

Ces fonctions permettent de définir l'automate $\overline{M}$ qui reconnaît le langage $L(\overline{E})$. On a :

$$\overline{M} = (Q, \sigma, \overline{\delta}, s_I, F)$$

1. $Q = Pos(E) \cup \{s_I\}$
2. $\forall x \in First(\overline{E}), \overline{\delta}(s_I, \alpha_x) = \{x\}$
3. $\forall x \in Pos(E), \forall y \in Follow(\overline{E}, x), \overline{\delta}(x, \alpha_y) = \{y\}$
4. $F = Last(\overline{E}) \cup Null_E\cdot \{s_I\}$

A partir de $\overline{M}$ on construit l'automate

$$M_E = (Q, \Sigma, \delta, s_I, F)$$

de la façon suivante : les arcs de M_E se déduisent de ceux de $\overline{M}$ en remplaçant leur étiquette α_x par $\chi(x)$. On a donc :

$$\delta = \{(q, a, q') \mid q \in Q,\ q' \in Q,\ a \in \Sigma,\ \exists\ \alpha_x \mid (q, \alpha_x, q') \in \overline{\delta} \text{ et } \chi(x) = a\}$$

Proposition 2.11 *Dans l'automate M_E, tous les arcs arrivant en un même état ont la même étiquette :*

$$\left.\begin{array}{l}(p, a, q) \in \delta\\(p', b, q) \in \delta\end{array}\right\} \Rightarrow a = b$$

Preuve. On remarque que par construction les arcs de $\overline{M}$ sont de la forme (x, α_y, y). Il s'en suit que les arcs de M sont de la forme $(x, \chi(y), y)$. ■

Proposition 2.12 *Si l'automate $\overline{M}$ reconnaît le langage $L(\overline{E})$ alors l'automate M_E reconnaît le langage $L(E)$.*

Preuve. Soit $L(\overline{M})$ le langage reconnu par l'automate $\overline{M}$ et $L(M_E)$ le langage reconnu par l'automate M_E.
Par construction, $L(\overline{M}) = L(\overline{E})$. Montrons que $L(M_E) = L(E)$.

1. $L(M_E) \subset L(E)$:
   Soit $u = u_1u_2 \dots u_p \in L(M_E)$, avec $u_j \in \Sigma$.
   Il existe un chemin réussi dans $M_E, (s_I, i_1, i_2, \dots, i_p)$, d'étiquette $u_1u_2 \dots u_p$. Le même chemin est également réussi dans $\overline{M}$, où son étiquette est $\alpha_{i_1}\alpha_{i_2} \dots \alpha_{i_p}$, avec $\chi(\alpha_{i_j}) = u_j, \forall j \in [1,p]$. Comme $L(\overline{M}) = L(\overline{E})$, on a : $\alpha_{i_1} \dots \alpha_{i_p} \in L(\overline{E})$. On en déduit $\chi(\alpha_{i_1}) \dots \chi(\alpha_{i_p}) \in L(E)$, soit $u \in L(E)$.

2. $L(E) \subset L(M_E)$:
   Soit $u = u_1u_2 \dots u_p \in L(E)$, avec $u_j \in \Sigma$.
   Il existe une suite de positions $(i_1, i_2, \dots, i_p)$ telles que $\chi(\alpha_{i_j}) = u_j, \forall j \in [1,p]$ et $\alpha_{i_1}\alpha_{i_2} \dots \alpha_{i_p} \in L(\overline{E})$. Comme $L(\overline{E}) = L(\overline{M})$, on a $\alpha_{i_1}\alpha_{i_2} \dots \alpha_{i_p} \in L(\overline{M})$. Le chemin $(s_I, i_1, i_2, \dots, i_p)$ est donc réussi dans $\overline{M}$. Il l'est également dans M_E, par conséquent $\chi(\alpha_{i_1}) \dots \chi(\alpha_{i_p}) \in L(M_E)$, soit $u \in L(M_E)$. ■

L'automate M_E est appelé "*automate de Glushkov*" de l'expression E. Il peut être défini directement comme suit :

$$M_E = (Q, \Sigma, \delta, s_I, F)$$

avec,

1. $Q = Pos(E) \cup \{s_I\}$
2. $\forall a \in \Sigma, \delta(s_I, a) = \{x \in First(E) / \chi(x) = a\}$
3. $\forall x \in Q, \forall a \in \Sigma, \delta(x, a) = \{y / y \in Follow(E, x) \text{ et } \chi(y) = a\}$
4. $F_E = Last(E) \cup Null_E \cdot \{s_I\}$

Remarquons que Berstel et Pin [3] présentent le calcul récursif des fonctions First, Last et Follow dans le cadre plus général des langages locaux.

3 Algorithme de Glushkov

Une implémentation de base de l'algorithme de Glushkov, rappelée dans [6] est la suivante :

On construit l'arbre syntaxique $T(E)$ correspondant à l'expression E. Chaque nœud ν de cet arbre représente une sous-expression E_ν de E et à chaque nœud ν on associe les variables suivantes :

- $First(\nu)$: liste qui représente l'ensemble $First(E_\nu)$
- $Last(\nu)$: liste qui représente l'ensemble $Last(E_\nu)$
- $Follow(\nu, x)$: liste qui représente l'ensemble $Follow(E_\nu, x)$
- $Null_\nu$: vaut $\{\varepsilon\}$ si $L(E_\nu)$ reconnaît le mot vide, $\emptyset$ sinon.

Ces variables sont calculées lors d'un parcours postfixé de l'arbre $T(E)$. ν_g et ν_d désignent respectivement le fils gauche et le fils droit d'un nœud ν étiqueté '+' ou '·' et ν_f désigne le fils d'un nœud ν étiqueté '*'. Le code incrémental suivant permet de calculer ces variables en chaque nœud ν.

Selon que ν **vaut**
$\emptyset$: $First(\nu) = \emptyset$; $Last(\nu) = \emptyset$; $Null_\nu = \emptyset$;
ε : $First(\nu) = \emptyset$; $Last(\nu) = \emptyset$; $Null_\nu = \{\varepsilon\}$;
x : $First(\nu) = \{x\}$; $Last(\nu) = \{x\}$; $Null_\nu = \emptyset$;
$Follow(\nu, x) = \emptyset$;
$+$: $First(\nu) = First(\nu_g) \cup First(\nu_d)$;
$Last(\nu) = Last(\nu_g) \cup Last(\nu_d)$;
$Null_\nu = Null_{\nu_g} \cup Null_{\nu_d}$;
$\cdot$: $First(\nu) = First(\nu_g) \cup Null_{\nu_g}.First(\nu_d)$;
$Last(\nu) = Last(\nu_d) \cup Null_{\nu_d}.Last(\nu_g)$;
$Null_\nu = Null_{\nu_g} \cap Null_{\nu_d}$;
pour tout $x \in Last(\nu_g)$ **faire**
$Follow(\nu, x) = Follow(\nu_g, x) \cup First(\nu_d)$;
finpour
$*$: $First(\nu) = First(\nu_f)$;
$Last(\nu) = Last(\nu_f)$;
$Null_\nu = \{\varepsilon\}$;
pour tout $x \in Last(\nu_f)$ **faire**
$Follow(\nu, x) = Follow(\nu_f, x) \cup First(\nu_f)$; [*]
finpour
finselonque

Toutes les unions d'ensembles, mise à part l'union [*], sont disjointes et peuvent être implémentées en $O(1)$, ceci en concaténant les listes représentant les ensembles en question. Il est possible de réaliser l'union [*] en temps constant en calculant *Follow* (ν, x) pour tout x de $Last_{\nu_f}$, par l'une des deux formules suivantes :

- $Follow(\nu, x) = [Follow(\nu_f, x) \setminus First_{\nu_f}] \bigcup First_{\nu_f}$ (1)
- $Follow(\nu, x) = Follow(\nu_f, x) \bigcup [First_{\nu_f} \setminus Follow(\nu_f, x)]$ (2)

Ces deux approches ont donné lieu à deux algorithmes :

- l'algorithme de Brüggemann-Klein [6] basé sur la formule (1);
- l'algorithme de Chang et Paige [9] basé sur la formule (2).

4 Algorithme de A. Brüggemann-Klein

On dit qu'une expression E est sous *forme normale de fermeture* (FNF) si elle vérifie, pour toute expression H telle que H^* soit une sous-expression de E, la condition suivante :

$$\forall x \in \mathit{Last}(H), \mathit{Follow}(H,x) \cap \mathit{First}(H) = \emptyset$$

L'intérêt de travailler sur une expression sous forme normale de fermeture est que l'opération *fermeture* y est réalisée à l'aide d'unions disjointes.

L'algorithme de Brüggemann-Klein consiste à construire, à partir d'une expression E, une expression $E^\bullet$ telle que :

1. $E^\bullet$ est sous forme normale de fermeture
2. $M_{E^\bullet} = M_E$
3. $E^\bullet$ peut être calculée à partir de E en temps linéaire.

Afin de calculer $E^\bullet$, toute expression H telle que H^* est une sous-expression de E est remplacée par une expression $H^{\circ *}$ telle que :

- H° vérifie la condition FNF
- $\mathit{First}(H^\circ) = \mathit{First}(H)$
- $\mathit{Last}(H^\circ) = \mathit{Last}(H)$
- $\mathit{Follow}(H^{\circ *}, x) = \mathit{Follow}(H^*, x) \quad \forall x \in \mathit{Pos}(H)$

Définition 4.1 [6] H° est définie récursivement de la façon suivante :

$$\begin{array}{rl}
[H = \varepsilon \text{ ou } \emptyset] & H^\circ = \emptyset \\
[H = a] & H^\circ = a \\
[H = F + G] & H^\circ = F^\circ + G^\circ \\
[H = FG] & H^\circ = \left\{ \begin{array}{ll} FG & \text{si } \mathit{Null}_F = \emptyset \text{ et } \mathit{Null}_G = \emptyset \\ F^\circ G & \text{si } \mathit{Null}_F = \emptyset \text{ et } \mathit{Null}_G = \{\varepsilon\} \\ FG^\circ & \text{si } \mathit{Null}_F = \{\varepsilon\} \text{ et } \mathit{Null}_G = \emptyset \\ F^\circ + G^\circ & \text{si } \mathit{Null}_F = \{\varepsilon\} \text{ et } \mathit{Null}_G = \{\varepsilon\} \end{array} \right. \\
[H = F^*] & H^\circ = F^\circ
\end{array}$$

Remarque 4.2 Si $\mathit{Null}_F = \emptyset$ et $\mathit{Null}_G = \{\varepsilon\}$ alors $F^\circ = F$ et $(FG)^\circ = FG$. En effet, si $\mathit{Null}_F = \emptyset$ alors il n'existe pas de transition allant d'un état de $\mathit{Last}(F)$ à un état de $\mathit{First}(F)$. De même, si $\mathit{Null}_F = \{\varepsilon\}$ et $\mathit{Null}_G = \emptyset$ alors $G^\circ = G$ et $(FG)^\circ = FG$.

Proposition 4.3 *Ainsi définie, H° vérifie la propriété suivante :*

$$\begin{array}{rl}
\mathit{Follow}(H^\circ, x) = & \mathcal{I}_{\mathit{Last}(H)}(x) \cdot [\mathit{Follow}(H,x) \backslash \mathit{First}(H)] \\
& \cup \mathcal{I}_{\mathit{Pos}(H) \backslash \mathit{Last}(H)}(x) \cdot \mathit{Follow}(H,x)
\end{array}$$

Preuve. La preuve se fait par récurrence sur la taille de l'expression H. La proposition 4.3 est évidente pour les expressions de taille 0 et 1. Supposons qu'elle soit vraie pour les expressions de taille inférieure à n. Soit H une expression de taille n, montrons que la proposition est vraie pour H :

Nous distinguons trois cas :

1. $[H = F + G], avec\ 1 \leq |F| \leq n-1$ et $1 \leq |G| \leq n-1$.
   Pour tout x de $Pos(H)$, on a :

$$Follow(H^\circ, x) = \mathcal{I}_{Pos(F)}(x) \cdot Follow(F^\circ, x) \cup \mathcal{I}_{Pos(G)}(x) \cdot Follow(G^\circ, x),$$

   par hypothèse de récurrence on a :

$$\begin{aligned}
Follow(F^\circ, x) &= \mathcal{I}_{Last(F)}(x) \cdot [Follow(F, x) \backslash First(F)] \\
&\quad \cup \mathcal{I}_{Pos(F)\backslash Last(F)}(x) \cdot Follow(F, x) \\
\text{et } Follow(G^\circ, x) &= \mathcal{I}_{Last(G)}(x) \cdot [Follow(G, x) \backslash First(G)] \\
&\quad \cup \mathcal{I}_{Pos(G)\backslash Last(G)}(x) \cdot Follow(G, x) \\
\text{d'où } Follow(H^\circ, x) &= \mathcal{I}_{Last(F)}(x) \cdot [Follow(F, x) \backslash First(F)] \\
&\quad \cup \mathcal{I}_{Last(G)}(x) \cdot [Follow(G, x) \backslash First(G)] \\
&\quad \cup \mathcal{I}_{Pos(F)\backslash Last(F)}(x) \cdot Follow(F, x) \\
&\quad \cup \mathcal{I}_{Pos(G)\backslash Last(G)}(x) \cdot Follow(G, x) \\
\text{donc } Follow(H^\circ, x) &= \mathcal{I}_{Last(H)}(x) \cdot [Follow(H, x) \backslash First(H)] \\
&\quad \cup \mathcal{I}_{Pos(H)\backslash Last(H)}(x) \cdot Follow(H, x)
\end{aligned}$$

2. $[H = F.G], avec\ 1 \leq |F| \leq n-1$ et $1 \leq |G| \leq n-1$.

   Sur le modèle précédent, on effectue une démonstration par récurrence. Quatre cas sont à envisager :

   - $Null_F = \emptyset$ et $Null_G = \emptyset$ entraîne $Follow(H^\circ, x) = Follow(F.G, x)$
   - $Null_F = \emptyset$ et $Null_G = \{\varepsilon\}$ entraîne $Follow(H^\circ, x) = Follow(F^\circ.G, x)$
   - $Null_F = \{\varepsilon\}$ et $Null_G = \emptyset$ entraîne $Follow(H^\circ, x) = Follow(F.G^\circ, x)$
   - $Null_F = \varepsilon$ et $Null_G = \varepsilon$ entraîne $Follow(H^\circ, x) = Follow(F^\circ + G^\circ, x)$, traité en 1.

3. $[H = F^*], avec |F| = |H| - 1$.
   Comme précédemment, la démonstration se fait par récurrence à partir de l'égalité suivante : $Follow(H^\circ, x) = Follow(F^\circ, x)$. ■

Définition 4.4 On définit $E^\bullet$ récursivement de la façon suivante :

$$\begin{array}{rl}
[E = \emptyset \mid \varepsilon \mid a] & E^\bullet = E \\
[E = F + G] & E^\bullet = F^\bullet + G^\bullet \\
[E = FG] & E^\bullet = F^\bullet G^\bullet \\
[E = F^*] & E^\bullet = F^{\bullet\circ *}
\end{array}$$

Exemple 4.5 Calcul de $E^\bullet$ pour $E = (a^*b^*)^*ab$ (voir Figure 1).

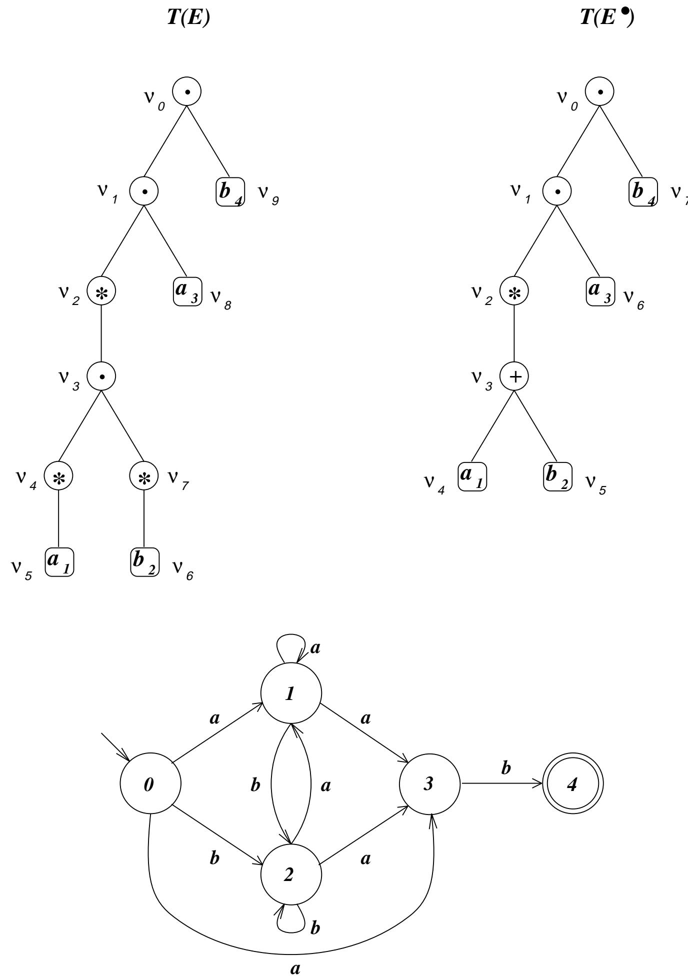


Figure 1: $E = (a^*b^*)^*ab$; $E^\bullet = (a+b)^*ab$; $M_E = M_{E^\bullet}$

$$\begin{aligned}
E^\bullet &= ((a^*b^*)^*ab)^\bullet = ((a^*b^*)^*a)^\bullet b^\bullet \quad = ((a^*b^*)^*)^\bullet a^\bullet b \\
&= (a^*b^*)^{\bullet\circ *}ab \quad = ((a^*)^\bullet (b^*)^\bullet)^{\circ *}ab = (a^{\bullet\circ *}b^{\bullet\circ *})^{\circ *}ab \\
&= (a^*b^*)^{\circ *}ab \quad = (a^{*\circ} + b^{*\circ})^*ab \quad = (a^\circ + b^\circ)^*ab \\
E^\bullet &= (a+b)^*ab
\end{aligned}$$

A. Brüggemann-Klein démontre dans [6] que $E^\bullet$ ainsi définie vérifie les trois propriétés :

1. $E^\bullet$ est sous forme normale de fermeture

2. $M_{E^\bullet} = M_E$

3. $E^\bullet$ peut être calculée à partir de E en temps linéaire.

ce qui permet d'énoncer le théorème suivant :

Théorème 4.6 *[6] L'automate de Glushkov M d'une expression rationnelle E peut être calculé en temps $\Omega(n^2)$, en passant par la forme normale de fermeture de E.*

5 Algorithme de Chang et Paige

L'algorithme de Glushkov permet de construire la fonction de transition δ selon la définition suivante :

Définition 5.1

$$\begin{array}{lll}
\delta_\emptyset & = \emptyset & 5.1.1 \\
\delta_\varepsilon & = \emptyset & 5.1.2 \\
\delta_a & = \emptyset & 5.1.3 \\
\delta_{F+G} & = \delta_F \cup \delta_G & 5.1.4 \\
\delta_{F.G} & = \delta_F \cup \delta_G \cup \Big(Last(F) \times First(G)\Big) & 5.1.5 \\
\delta_{F^*} & = \delta_F \cup \Big(Last(F) \times First(F)\Big) & 5.1.6
\end{array}$$

Toutes les unions qui figurent dans la définition 5.1 sont disjointes, exceptée la dernière : $\delta_F \cup Last(F) \times First(F)$ (5.1.6). Pour se ramener à une union disjointe, Chang et Paige [9] définissent l'ensemble suivant :

$$\mathrm{Nred}(F) = \Big[Last(F) \times First(F)\Big] \backslash \delta_F$$

et remplacent la formule (5.1.6) par :

$$\delta_{F^*} = \delta_F \cup \mathrm{Nred}(F)$$

D'autre part, ils proposent de calculer $\mathrm{Nred}(F)$ de façon récursive, selon les formules suivantes :

Définition 5.2

$$\begin{array}{lll}
\mathrm{Nred}(\emptyset) & = & \emptyset \\
\mathrm{Nred}(\varepsilon) & = & \emptyset \\
\mathrm{Nred}(a) & = & Last(a) \times First(a) \\
\mathrm{Nred}(G+H) & = & \mathrm{Nred}(G) \cup \mathrm{Nred}(H) \\
 & & \cup\, Last(H) \times First(G) \\
 & & \cup\, Last(G) \times First(H) \\
\mathrm{Nred}(GH) & = & Null_H.\mathrm{Nred}(G) \cup Null_G.\mathrm{Nred}(H) \\
 & & \cup\, Last(H) \times First(G) \\
\mathrm{Nred}(G^*) & = & \emptyset
\end{array}$$

Afin de pouvoir comparer les approches de Chang et Paige et de A. Brüggemann-Klein , nous désignons par δ la fonction de transition calculée par l'algorithme de Chang et Paige et par d celle qui est calculée par l'algorithme de A. Brüggemann-Klein .

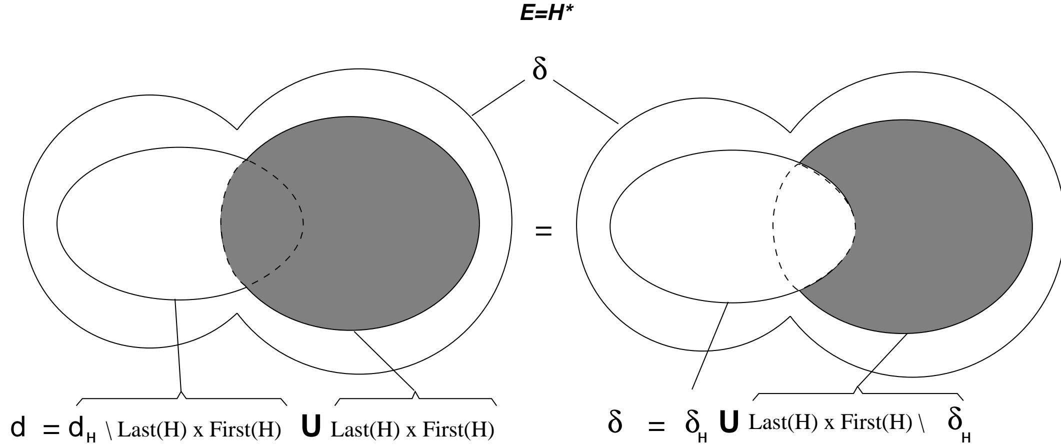


Figure 2: $d = \delta$

Définition 5.3 d est calculée récursivement comme suit :

$$\begin{array}{ll} d_\emptyset & = \emptyset \\ d_\varepsilon & = \emptyset \\ d_a & = \emptyset \\ d_{F+G} & = d_F \cup d_G \\ d_{F.G} & = d_F \cup d_G \cup Last(F) \times First(G) \\ d_{F^*} & = d_{F^\circ} \cup Last(F) \times First(F) \qquad (F^* \text{ est remplacée par } (F^\circ)^*) \\ & \qquad \text{où } d_{F^\circ} = d_F \backslash [Last(F) \times First(F)] \end{array}$$

Proposition 5.4 *Pour toute expression rationnelle E nous avons $d = \delta$.*

Preuve. d et δ correspondent aux deux variantes possibles pour calculer une union en union disjointe (cf. fig.2). La preuve se fait par récurrence sur la taille de E.

1. $|E| = 0$, $|E| = 1$ la proposition 5.4 est évidente.
2. $|E| = n + 1$

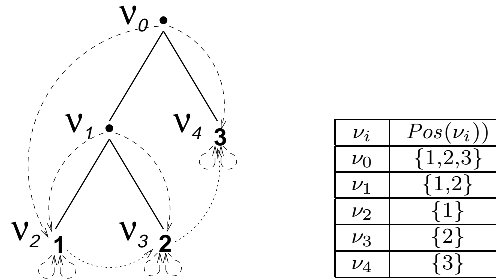
   (a) $[E = F + G]$ sachant que $d_F = \delta_F$ et $d_G = \delta_G$
   $\Leftrightarrow d_F \cup d_G = \delta_F \cup \delta_G$
   $\Leftrightarrow d = \delta$

   (b) $[E = F.G]$ sachant que $d_F = \delta_F$ et $d_G = \delta_G$
   $\Leftrightarrow d_F \cup d_G = \delta_F \cup \delta_G$
   $\Leftrightarrow d = \delta$

   (c) $[E = F^*]$ sachant que $d_{F^*} = d_{F^{\circ\bullet *}} = \delta_{E^\bullet}$
   Posons $R = First(F)$ et $L = Last(F)$,
   $\Leftrightarrow d = d_{F^\circ} \cup L \times R = \Big[\delta_F \backslash (L \times R)\Big] \cup L \times R$ fig. 2 (d)
   $\Leftrightarrow d = \delta_F \cup L \times R = \delta_F \cup \Big[(L \times R) \backslash \delta_F\Big]$ fig. 2 (δ)
   $\Leftrightarrow d = \delta_F \cup \text{Nred}(F) = \delta$

■

Chang et Paige utilisent une structure de données leur permettant de calculer une représentation de δ en temps $O(n)$ et en espace $O(s)$; et de passer à la matrice de transition en $O(n^2)$.



ν_i	$Pos(\nu_i)$)
ν_0	{1,2,3}
ν_1	{1,2}
ν_2	{1}
ν_3	{2}
ν_4	{3}

Figure 3: Ensembles $Pos(\nu_i)$ des feuilles associées aux nœuds ν_i

6 Un nouvel algorithme

Nous présentons dans cette section un algorithme original qui calcule une représentation de l'automate de Glushkov M_E d'une expression rationnelle E en temps et en espace $O(n)$. Dans cet algorithme, le calcul des First est effectué sur une forêt déduite de l'arbre $T(E)$, de même pour le calcul des Last. La fonction de transitions δ de l'automate M_E est représentée par un ensemble de liens entre ces deux forêts, un lien correspondant à un produit cartésien $L \times F$ où L est un ensemble Last et F un ensemble First.

6.1 Calcul des First

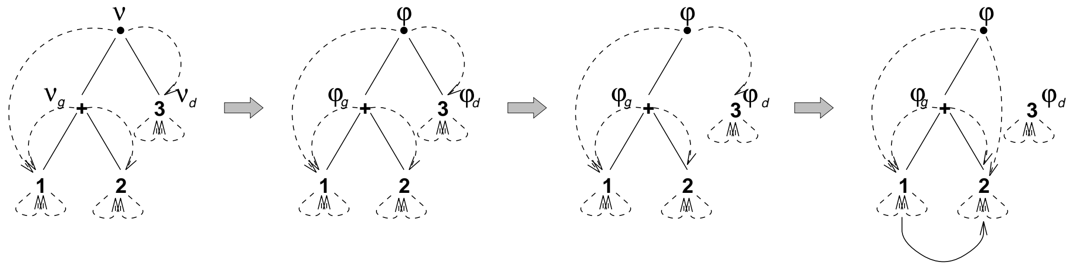
Soit $T(E)$ l'arbre syntaxique associé à l'expression E. A chaque nœud ν est associé le sous-arbre de racine ν et l'ensemble des feuilles de ce sous-arbre, que l'on notera $Pos(\nu)$ par abus d'écriture. Cet ensemble peut être représenté par une liste. Chaque nœud ν, y compris une feuille, dispose d'un pointeur vers la feuille la plus à gauche et d'un pointeur vers la feuille la plus à droite, ce qui permet d'accéder en temps constant à l'ensemble $Pos(\nu)$ associé à un nœud ν (voir figure 3).

A partir de l'arbre $T(E)$, on construit la forêt $TF(E)$ qui représente l'ensemble des First définis par Glushkov, de la manière suivante :

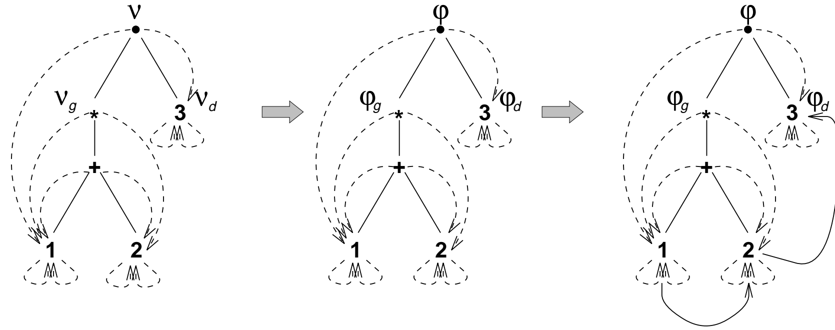
- $TF(E)$ est initialisée par une copie de $T(E)$; chaque nœud ν de $T(E)$ est rebaptisé φ dans $TF(E)$
- pour chaque nœud φ étiqueté "·", on supprime le lien avec son fils droit φ_d si $L(E_{\nu_g})$ ne reconnaît pas le mot vide (voir figure 4). On respecte ainsi la définition :

$$First(F.G) = First(F) \cup Null_F.First(G)$$

- pour chaque nœud φ :
  - on met à jour le pointeur le plus à gauche et le pointeur le plus à droite
  - on relie la feuille la plus à droite de son fils gauche à la feuille la plus à gauche de son fils droit



(a) $\varepsilon \notin L(E_{\nu_g})$, le lien (φ, φ_d) est supprimé



(b) $\varepsilon \in L(E_{\nu_g})$, le lien (φ, φ_d) est conservé

Figure 4: Mise à jour des liens pour le calcul de First

Remarque 6.1

1. Par la proposition 2.8 on a : $First(F+G) = First(F) \cup First(G)$ et $First(F^*) = First(F)$. Par conséquent l'initialisation de $TF(E)$ par une copie de $T(E)$ assure que les liens relatifs aux nœuds étiquetés autrement que par '·' sont corrects.

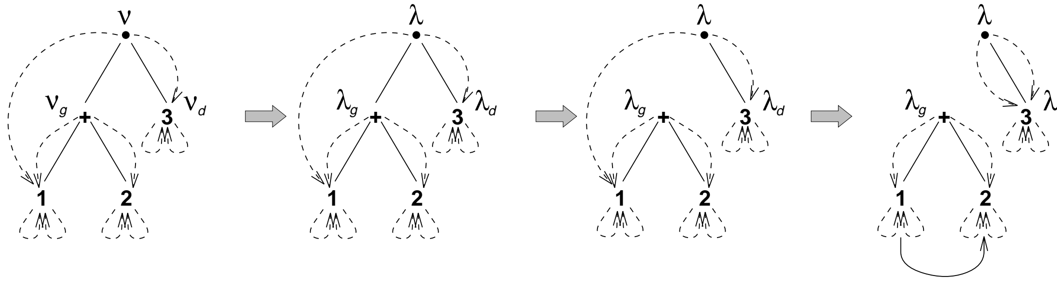
2. L'arbre syntaxique inchangé permet d'accéder aux différents arbres de la forêt $TF(E)$ une fois les liens supprimés.

Après élimination de ces liens et mise à jour des pointeurs, on obtient la forêt $TF(E)$, pour chaque nœud φ de laquelle l'ensemble $Pos(\varphi)$ est exactement l'ensemble First de la sous-expression E_ν. A l'intérieur d'un arbre de $TF(E)$ les feuilles sont chaînées. On accède en temps constant à $First(E_\nu)$. En particulier $First(E) = Pos(\varphi_0)$. Sur l'exemple de la figure 6, pour l'expression $E = (a_1^* b_2^*) a_3 (b_4 + a_5)(b_6 a_7)$, $First(E)$ est l'ensemble associé à la racine de l'arbre $\mathcal{A}_0$ de la forêt $TF(E)$.

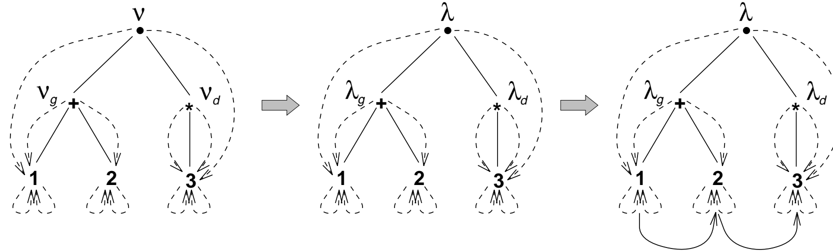
6.2 Calcul des Last

De la même façon qu'on construit la forêt des First, on construit la forêt des Last $TL(E)$ (chaque nœud ν de $T(E)$ est rebaptisé λ dans $TL(E)$), en respectant la définition 2.9. Dans $TL(E)$, on supprime pour chaque nœud λ étiqueté "·" le lien avec son fils gauche λ_g si le langage $L(E_{\nu_d})$ ne reconnaît pas le mot vide (voir figure 5). On respecte ainsi la définition :

$$Last(F.G) = Last(G) \cup Null_G.Last(F)$$



(a) $\varepsilon \notin L(E_{\nu_d})$, le lien (λ, λ_g) est supprimé



(b) $\varepsilon \in L(E_{\nu_d})$ le lien (λ, λ_g) est conservé.

Figure 5: Mise à jour des liens pour le calcul de Last

De même que pour la forêt $TF(E)$, pour chaque nœud λ de $TL(E)$ on met à jour les différents pointeurs. A chaque nœud λ de la forêt $TL(E)$ est associé un ensemble $Pos(\lambda)$ qui est exactement l'ensemble Last de la sous-expression E_ν. A l'intérieur d'un arbre de $TL(E)$ les feuilles sont chaînées. On accède en temps constant à $Last(E_\nu)$. En particulier, $Last(E) = Pos(\lambda_0)$. Sur l'exemple de la figure 6, pour l'expression $E = (a_1^* b_2^*) a_3 (b_4 + a_5)(b_6 a_7)$, $Last(E)$ est l'ensemble associé à la racine de l'arbre $\mathcal{A}_4$ de la forêt $TL(E)$.

6.3 Calcul de $First(E)$ et de $Last(E)$

Rappelons que si ν_0 est la racine de l'arbre $T(E)$, $First(E)$ est l'ensemble $Pos(\varphi_0)$ dans la forêt $TF(E)$ et $Last(E)$ est l'ensemble $Pos(\lambda_0)$ dans la forêt $TL(E)$.

Lemme 6.2 *Pour toute expression rationnelle E de taille s, nous pouvons calculer First(E) en un temps $O(n)$ en utilisant un espace mémoire en $O(n)$.*

Preuve. La construction de l'arbre $T(E)$ à partir d'une expression rationnelle E s'effectue en temps linéaire sur la taille de E. En effet les expressions rationnelles peuvent être décrites par des grammaires $LL(1)$ [1]. D'autre part, le calcul récursif des pointeurs vers la feuille la plus à gauche et la feuille la plus à droite dans l'arbre $T(E)$ s'effectue en $O(n)$. La mise à jour de ces pointeurs dans la forêt $TF(E)$ et le chaînage des feuilles est également en $O(n)$. Par conséquent, la forêt des First $TF(E)$ peut être calculée en $O(n)$ en utilisant un espace mémoire en $O(n)$. ■

De même,

Lemme 6.3 *Pour toute expression rationnelle E dont le nombre d'occurrences de lettres est n, nous pouvons calculer $Last(E)$ en un temps $O(n)$ en utilisant un espace mémoire en $O(s)$.*

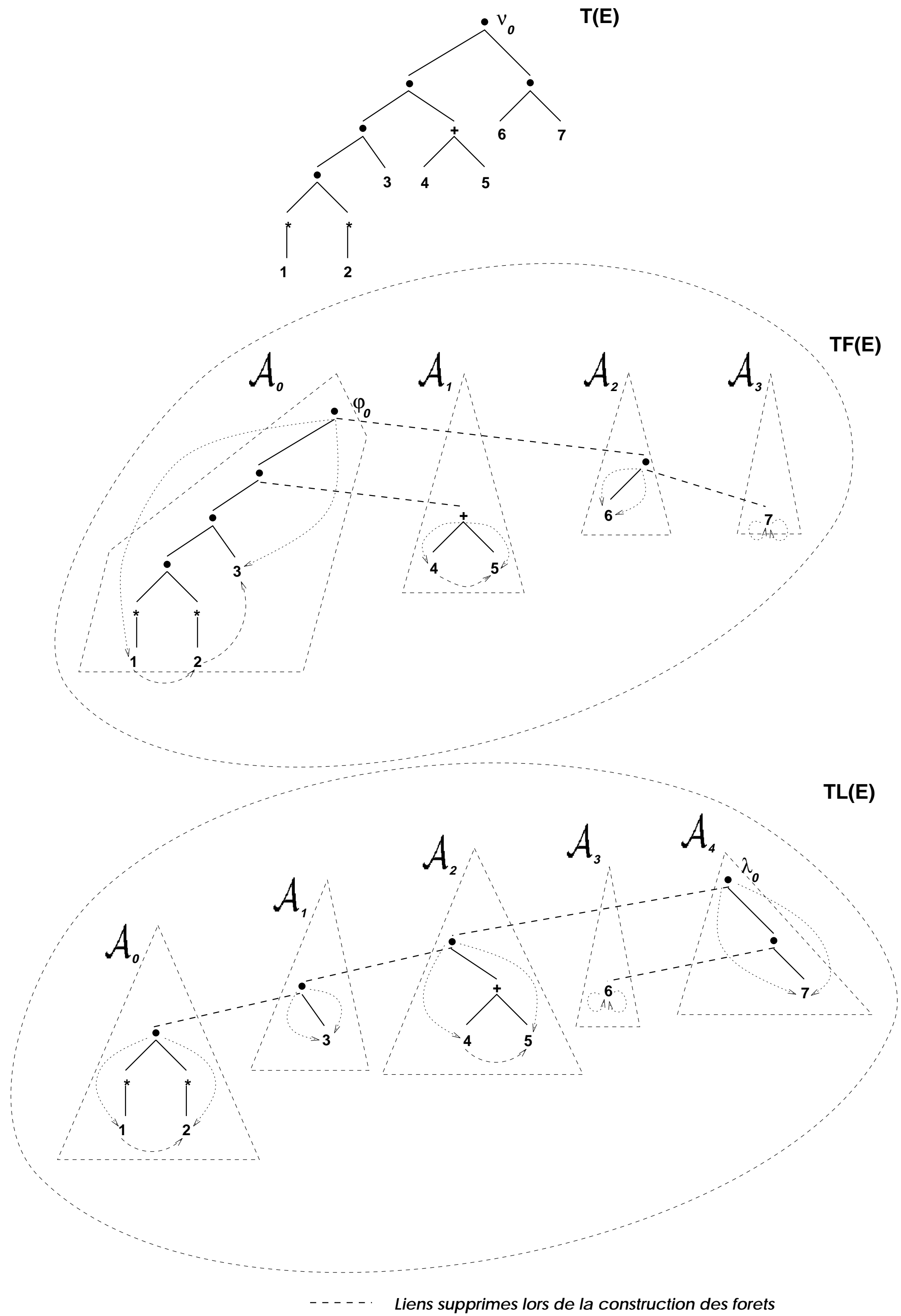


Figure 6: Les forêts $TF(E)$ et $TL(E)$ de l'expression $E = (a_1^* b_2^*) a_3 (b_4 + a_5)(b_6 a_7)$

6.4 Calcul de δ

6.4.1 Représentation de δ

Il s'agit d'une représentation de δ par un ensemble de liens entre les forêts $TL(E)$ et $TF(E)$. δ est l'ensemble des arcs de l'automate de Glushkov de l'expression E. δ peut être calculé récursivement sur l'arbre $T(E)$ à partir des deux ensembles suivants :

- Δ_ν est l'ensemble des arcs calculés lors de l'évaluation du nœud ν.
- D_ν est l'ensemble des arcs calculés avant l'évaluation du nœud ν.

On a les formules suivantes :

$$\Delta_\nu = \begin{cases} Last(E_{\nu_g}) \times First(E_{\nu_d}) & \text{si } \nu \text{ est étiqueté } \cdot \\ Last(E_{\nu_f}) \times First(E_{\nu_f}) & \text{si } \nu \text{ est étiqueté } * \\ \emptyset & \text{sinon} \end{cases}$$

$$D_\nu = \begin{cases} \emptyset & \text{si } \nu \text{ est une feuille} \\ D_{\nu_f} \cup \Delta_\nu & \text{si } \nu \text{ est étiqueté } * \\ D_{\nu_g} \cup D_{\nu_d} \cup \Delta_\nu & \text{sinon} \end{cases}$$

$$\delta = D_{\nu_0} \cup (\{s_I\} \times First(E))$$

Proposition 6.4

$$\delta = \left(\bigcup_{\nu \in T(E)} \Delta_\nu \right) \cup \left(\{s_I\} \times First(E) \right)$$

Preuve. Pour toute expression rationnelle de taille 1 on a : $\Delta_\nu = \emptyset$ et $D_\nu = \emptyset$. Supposons que pour toute expression E, telle que $1 < |E| < n$, on ait : $D_\nu = \bigcup_{\nu' \in T(E)} \Delta_{\nu'}$.

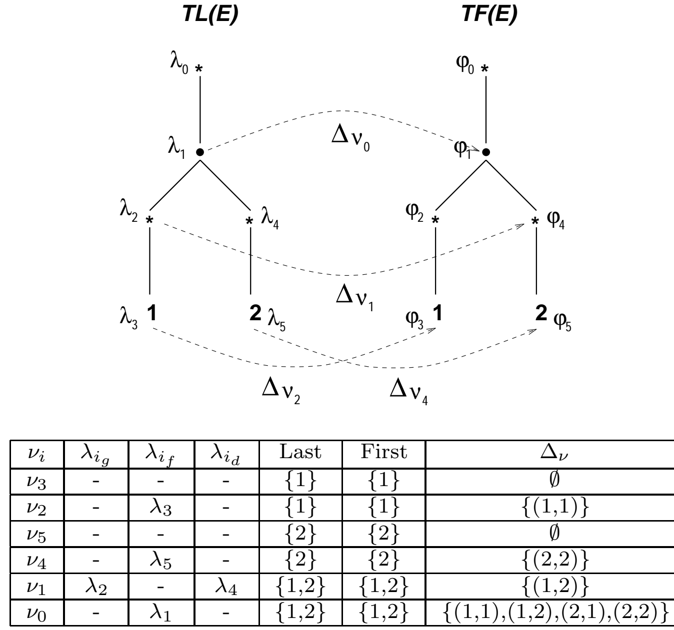
Montrons que la propriété est vraie pour toute expression de taille inférieure ou égale à n : Soit E_ν telle que $|E_\nu| = n$.

On a $D_\nu = \begin{cases} D_{\nu_f} \cup \Delta_\nu & \text{si } \nu \text{ est étiqueté } * \\ D_{\nu_g} \cup D_{\nu_d} \cup \Delta_\nu & \text{sinon} \end{cases}$

Par hypothèse de récurrence, on a :

$$D_{\nu_f} = \bigcup_{\nu' \in T(E_{\nu_f})} \Delta_{\nu'},$$
$$D_{\nu_g} = \bigcup_{\nu' \in T(E_{\nu_g})} \Delta_{\nu'},$$
$$D_{\nu_d} = \bigcup_{\nu' \in T(E_{\nu_d})} \Delta_{\nu'}.$$

Par conséquent on a :



ν_i	λ_{i_g}	λ_{i_f}	λ_{i_d}	Last	First	Δ_ν
ν_3	-	-	-	{1}	{1}	$\emptyset$
ν_2	-	λ_3	-	{1}	{1}	{(1,1)}
ν_5	-	-	-	{2}	{2}	$\emptyset$
ν_4	-	λ_5	-	{2}	{2}	{(2,2)}
ν_1	λ_2	-	λ_4	{1,2}	{1,2}	{(1,2)}
ν_0	-	λ_1	-	{1,2}	{1,2}	{(1,1),(1,2),(2,1),(2,2)}

Figure 7: Calcul des Δ_ν pour l'expression $E = (a_1^* b_2^*)^*$

$$D_\nu = \begin{cases} \left(\bigcup_{\nu' \in T(E_{\nu_f})} \Delta_{\nu'} \right) \cup \Delta_\nu & \text{si } \nu \text{ est étiqueté *} \\ \left(\bigcup_{\nu' \in T(E_{\nu_g})} \Delta_{\nu'} \right) \cup \left(\bigcup_{\nu' \in T(E_{\nu_g})} \Delta_{\nu'} \right) \cup \Delta_\nu & \text{sinon} \end{cases}$$

soit $\quad D_\nu = \bigcup_{\nu' \in T(E_\nu)} \Delta'_\nu$

La propriété est donc vraie pour toute expression, en particulier :

$$D_{\nu_0} = \bigcup_{\nu \in T(E)} \Delta_\nu$$

D'où : $\delta = \left(\bigcup_{\nu \in T(E)} \Delta_\nu \right) \cup \Big(\{s_I\} \times \mathit{First}(E) \Big)$

■

Représentation de δ : chaque Δ_ν non vide est un produit cartésien de la forme $\mathit{Last}(E_{\nu'}) \times \mathit{First}(E_{\nu''})$ représenté par un lien (λ', φ'') entre le nœud λ' de la forêt $TL(E)$ correspondant à ν' et le nœud φ'' de la forêt $TF(E)$ correspondant à ν''; δ est donc représenté par l'ensemble L de tous ces liens.

Complexité du calcul de L : ajouter un lien à L se fait en temps constant; on obtient tous les liens en faisant le parcours récursif de $T(E)$. L est donc calculé en temps $O(s)$.

6.4.2 Elimination des liens redondants

Soit $\Delta_\nu \neq \emptyset$. S'il existe $\Delta_{\nu'}$ tel que $\Delta_\nu \subseteq \Delta_{\nu'}$, on dit que le lien représentant Δ_ν est redondant. Sur l'exemple de la figure 7 on a : $\Delta_{\nu_1} \subset \Delta_{\nu_0}$, $\Delta_{\nu_2} \subset \Delta_{\nu_0}$ et $\Delta_{\nu_4} \subset \Delta_{\nu_0}$. Δ_{ν_1}, Δ_{ν_2} et Δ_{ν_4} sont donc redondants.

Eliminer les liens redondants permettra d'exprimer δ sous forme d'union d'ensembles disjoints. Les deux propositions suivantes permettent de caractériser les

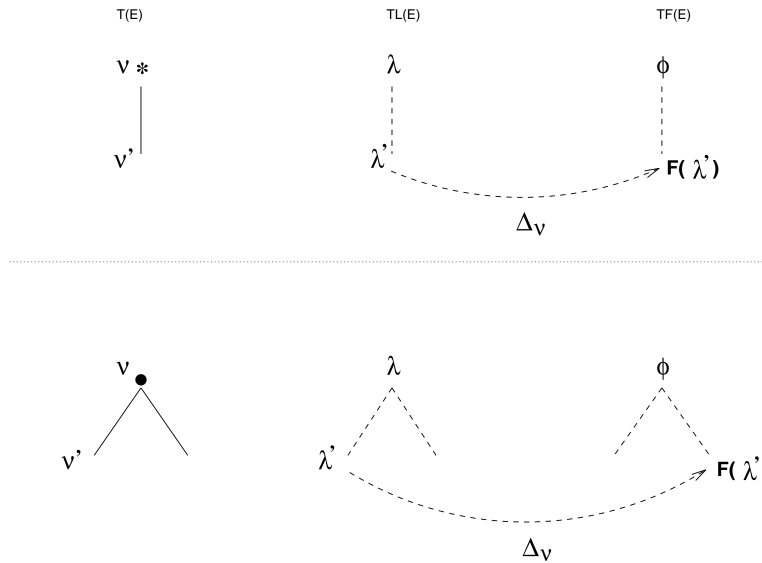


Figure 8: Correspondance entre les nœuds de $T(E)$, $TL(E)$ et $TF(E)$

liens redondants.

Rappel : Soient deux nœuds s et s' d'une forêt. Le nœud s' est descendant du nœud s si et seulement si s' appartient à l'arbre de racine s.

Proposition 6.5 *Soient ν et ν' deux nœuds dans $T(E)$, et φ et φ' les nœuds correspondants dans TF(E). Alors :*

$$\begin{cases} \text{si } \varphi' \text{est descendant de } \varphi : & First(E_{\nu'}) \subseteq First(E_\nu) \\ \text{sinon :} & First(E_{\nu'}) \cap First(E_\nu) = \emptyset \end{cases}$$

Preuve. Rappelons que l'ensemble des feuilles de l'arbre de racine φ, $Pos(\varphi)$, est égal à $First(E_\nu)$.

- Si φ' est descendant de φ, alors l'arbre de racine φ' est un sous-arbre de l'arbre de racine φ. Par conséquent $Pos(\varphi')$ est inclus dans $Pos(\varphi)$. Donc $First(E_{\nu'})$ est inclus dans $First(E_\nu)$.
- Si φ' n'est pas descendant de φ, alors les arbres de racine φ et φ' sont disjoints. Par conséquent $Pos(\varphi)$ et $Pos(\varphi')$ sont disjoints et donc $First(E_\nu)$ et $First(E_{\nu'})$ sont disjoints. ■

Soit ν un nœud de $T(E)$, λ (resp. φ) le nœud correspondant dans $TL(E)$ (resp. $TF(E)$). Supposons $\Delta_\nu \neq \emptyset$. Désignons par λ' le nœud de $TL(E)$ correspondant au fils de ν si ν est étiqueté '*', ou au fils gauche de ν si ν est étiqueté '·'. Désignons par $F(\lambda')$ le nœud de $TF(E)$ correspondant au fils de ν si ν est étiqueté '*', ou au fils droit de ν si ν est étiqueté '·'. Δ_ν est donc l'ensemble $Pos(\lambda') \times Pos(F(\lambda'))$. Sur la structure de données utilisée, Δ_ν est représenté par un lien entre λ' dans $TL(E)$ et $F(\lambda')$ dans $TF(E)$ (voir figure 8).

Proposition 6.6 *Soient ν_1 et ν_2 deux nœuds de $T(E)$. Soient λ_1 et λ_2 les nœuds correspondants dans $TL(E)$. Soient λ'_1 et λ'_2 définis comme précédemment. Soient*

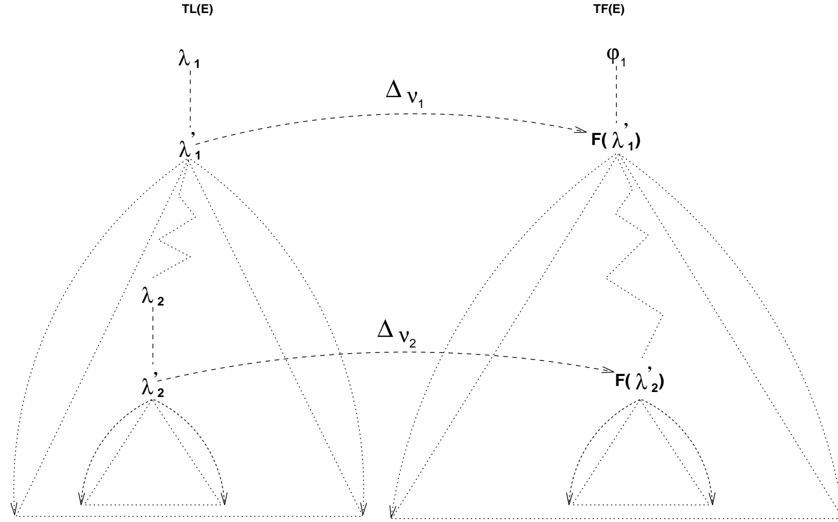


Figure 9: $\Delta_{\nu_2} \subseteq \Delta_{\nu_1}$

$F(\lambda'_1)$ et $F(\lambda'_2)$ définis comme précédemment.

On a : $\Delta_{\nu_2} \subseteq \Delta_{\nu_1} \Leftrightarrow \begin{cases} \lambda'_2 \textit{ est descendant de } \lambda'_1 \textit{ dans } TL(E) \\ F(\lambda'_2) \textit{ est descendant de } F(\lambda'_1) \textit{ dans } TF(E) \end{cases}$

Preuve.

$\Delta_{\nu_1} = Pos(\lambda'_1) \times Pos(F(\lambda'_1))$
$\Delta_{\nu_2} = Pos(\lambda'_2) \times Pos(F(\lambda'_2))$

$$\Delta_{\nu_2} \subseteq \Delta_{\nu_1} \iff \begin{cases} Pos(\lambda'_1) \subseteq Pos(\lambda'_2) \\ Pos(F(\lambda'_1)) \subseteq Pos(F(\lambda'_2)) \end{cases}$$

$$\Delta_{\nu_2} \subseteq \Delta_{\nu_1} \overset{\text{prop.6.5}}{\iff} \begin{cases} \lambda'_2 \text{ est descendant de } \lambda'_1 \text{ dans } TL(E) \\ F(\lambda'_2) \text{ est descendant de } F(\lambda'_1) \text{ dans } TF(E) \end{cases}$$

■

L'algorithme éliminer décrit ci-dessous met en œuvre la proposition 6.6 pour éliminer les liens redondants. Précisons quelques points utiles à la compréhension de l'algorithme :

- Les liens $(\lambda'_1, F(\lambda'_1))$ et $(\lambda'_2, F(\lambda'_2))$ de la propriété 6.6 sont les liens $(a, F(a))$ et $(b, F(b))$ de l'algorithme; a et b sont des nœuds de $TL(E)$ ou bien ne sont pas définis (nil).
- La fonction $D(x, y)$ teste si x est descendant de y dans la forêt $TF(E)$; x et y sont des nœuds de $TF(E)$ ou bien ne sont pas définis. On a : $D(\text{nil}, \text{nil}) = D(\text{nil}, y) = D(x, \text{nil}) = \text{faux}$.
- Les fonctions $g(a), d(a), f(a)$ fournissent le fils gauche, le fils droit, ou le fils (cas de la fermeture) du nœud a de $TL(E)$.
- L'algorithme permet d'éliminer les liens redondants dans un arbre de la forêt $TL(E)$. Le parcours de cet arbre est tel que b est toujours descendant de a.
- L'algorithme est appelé sur chaque arbre de la forêt $TL(E)$, b prenant pour valeur la racine de l'arbre, et a la valeur nil.

Appel	b	a	$F(b)$	$F(a)$	$D(F(b),F(a))$	Action(s)
1	λ_0	nil	nil	nil	faux	Eliminer(λ_1,nil) (2)
2	λ_1	nil	λ_1	nil	faux	Eliminer(λ_2,λ_1) (3) Eliminer(λ_4,λ_1) (7)
3	λ_2	λ_1	λ_4	λ_1	vrai	Supp. le lien (λ_2, φ_4) Eliminer(λ_3,λ_1) (4)
4	λ_3	λ_1	λ_3	λ_1	vrai	Supp. le lien (λ_3, φ_3) Eliminer(nil,λ_1) (5) Eliminer(nil,λ_1) (6)
5	nil	λ_1	-	-	-	-
6	nil	λ_1	-	-	-	-
7	λ_4	λ_1	nil	λ_1	faux	Eliminer(λ_5,λ_1) (8)
8	λ_5	λ_1	λ_5	λ_1	vrai	Supp. le lien (λ_5, φ_5) Eliminer(nil,λ_1) (9) Eliminer(nil,λ_1) (10)
9	nil	λ_1	-	-	-	-
10	nil	λ_1	-	-	-	-

Figure 10: Déroulement de l'algorithme Eliminer sur l'exemple de la figure 7

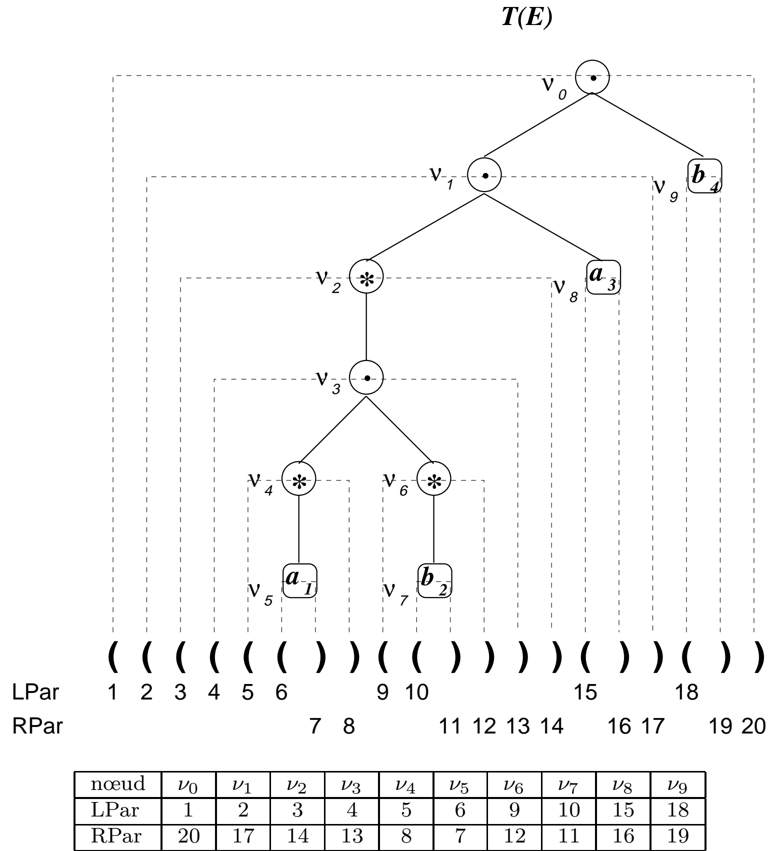
Algorithme Eliminer

```
Eliminer(b,a)
begin
  if b ≠ nil then
     if D(F(b),F(a)) then                                     ⎫
        F(b) ← nil              (supp. du lien (b,F(b)))      ⎪
        switch (b)                                            ⎬ Δ_b ⊆ Δ_a
        case '·','+',x :  Eliminer(g(b),a)                    ⎪
                          Eliminer(d(b),a)                    ⎪
        case '*' :        Eliminer(f(b),a)                    ⎭
     else if F(b) ≠ nil then                                          ⎫
              switch (b)                                   ⎫          ⎪
              case '·','+',x :  Eliminer(g(b),b)           ⎬ Δ_b ≠ ∅  ⎪
                                Eliminer(d(b),b)           ⎪          ⎪
              case '*' :        Eliminer(f(b),b)           ⎭          ⎬ Δ_a ∩ Δ_b = ∅
          else switch (b)                                  ⎫          ⎪
              case '·','+',x :  Eliminer(g(b),a)           ⎬ Δ_b = ∅  ⎪
                                Eliminer(d(b),a)           ⎪          ⎪
              case '*' :        Eliminer(f(b),a)           ⎭          ⎪
          endif                                                       ⎭
     endif
  endif
end
```

Complexité de l'algorithme d'élimination :

L'algorithme est exécuté sur chaque arbre de la forêt $TL(E)$. Au cours de son exécution chaque nœud de $T(E)$ est visité une fois et une seule. La suppression d'un lien ($F(b) \leftarrow$ nil) est en temps constant, sous réserve que :

- Dans $TF(E)$ les arbres sont indexés de manière distincte.
- Un nœud de $TF(E)$ est numéroté par un triplet formé par



nœud	ν_0	ν_1	ν_2	ν_3	ν_4	ν_5	ν_6	ν_7	ν_8	ν_9
LPar	1	2	3	4	5	6	9	10	15	18
RPar	20	17	14	13	8	7	12	11	16	19

Figure 11: Exemple de parenthésage pour $E = (a_1^* b_2^*)^*.a_3.b_4$

1. le numéro d'arbre dans la forêt $TF(E)$
2. le numéro de la parenthèse ouvrante (Lpar(ν_i)) qui lui est associée dans l'arbre $T(E)$ (cf. fig. 11)
3. le numéro de la parenthèse fermante (Rpar(ν_i)) qui lui est associée dans l'arbre $T(E)$ (cf. fig. 11)

Remarque 6.7 La numérotation des parenthèses (fig. 11) est réalisée en temps linéaire lors de la construction de l'arbre $T(E)$. Ainsi l'indexation de l'ensemble des nœuds de la forêt $TF(E)$ par un triplet est réalisée en temps linéaire.

$F(b)$ est indexé par le triplet $(\text{Arb}(b), \text{Lpar}(b), \text{Rpar}(b))$ et $F(a)$ est indexé par le triplet $(\text{Arb}(a), \text{Lpar}(a), \text{Rpar}(a))$; le test de descendance $D(F(b), F(a))$ s'écrit donc :

$$D\Big(F(b), F(a)\Big) = \Big(\underbrace{(\text{Arb}(a) = \text{Arb}(b))}_{\mathbf{A}} \wedge \overbrace{((\text{Lpar}(b) \geq \text{Lpar}(a)) \wedge (\text{Rpar}(b) \leq \text{Rpar}(a)))}^{\mathbf{B}}\Big)$$

- **A** est *vrai* si a et b sont dans le même arbre dans la forêt $TF(E)$
- **B** est *vrai* si b est descendant de a dans la forêt $TF(E)$

Par conséquent, ce test peut être fait en temps constant.

Lemme 6.8 *La représentation de δ par un ensemble de liens non redondants peut être calculée en temps $O(n)$.*

Cette représentation permet de calculer δ comme une union d'ensembles disjoints.

La relation entre la forme normale de fermeture et notre algorithme d'élimination des liens redondants est fournie par la proposition suivante :

Proposition 6.9 *si E est sous forme normale de fermeture, alors :*

$$\delta = \Big(\biguplus_{\nu \in T(E)} \Delta_\nu \Big) \cup \Big(\{s_I\} \times First(E) \Big)$$

Preuve.

- D'une part, la propriété 6.6 nous permet d'affirmer qu'après terminaison de l'algorithme d'élimination, la condition suivante est vérifiée :

  Pour tout λ_1 et tout λ_2 dans $TL(E)$, tels que $F(\lambda_1)$ et $F(\lambda_2)$ existent dans $TL(E)$ on a :

  (λ_2 est descendant de λ_1) $\Rightarrow$ ($F(\lambda_2)$ n'est pas descendant de $F(\lambda_1)$).

- D'autre part, si E est sous forme normale de fermeture, pour toute expression H telle que H^* soit une sous-expression de E, on a :

  $$\forall x \in Last(H), Follow(H, x) \cap First(H) = \emptyset$$

  Soit λ le nœud de $TL(E)$ correspondant à la sous-expression H.

  Comme $x \in Last(H)$, on a : x est descendant de λ.

  Soit λ' un descendant de λ dans $TL(E)$ tel que x soit descendant de λ' et tel que $F(\lambda')$ existe.

  On a : $First(E_{\lambda'}) \subseteq Follow(H, x)$.

  Supposons que $F(\lambda')$ soit descendant de $F(\lambda)$. Alors $First(E_{\lambda'}) \cap First(H) \neq \emptyset$.

  On en déduit que $Follow(H, x) \cap First(H) \neq \emptyset$, ce qui est en contradiction avec la condition *FNF*. Par conséquent, $F(\lambda')$ ne peut être descendant de $F(\lambda)$. ■

6.4.3 De la représentation par forêts à l'automate de Glushkov

Il s'agit de construire la table de transitions de l'automate $\overline{M}$. L'ensemble $First(\nu_0)$ permet de trouver les arcs issus de l'état initial s_I. On parcourt l'arbre $T(E)$; chaque fois que l'on rencontre un lien $Last(\nu) \times First(\nu')$, $M[i, j]$ reçoit j, $\forall i \in Last(\nu), \forall j \in First(\nu')$.

Les liens étant non redondants, on fait cette opération exactement une fois pour chaque arc. D'où une étape réalisée en temps proportionnel au nombre de transitions, c'est à dire en $\Omega(n^2)$.

Lemme 6.10 *Le passage de la représentation de δ par un ensemble de liens non redondants à la table de transitions est en $\Omega(n^2)$.*

6.5 Algorithme

Nous présentons ci-dessous les étapes importantes de l'algorithme général pour le passage d'une expression rationnelle à son automate de Glushkov.

1. Construction de l'arbre $T(E)$ et initialisation de ses nœuds $O(n)$
2. Construction des forêts $TL(E)$ et $TF(E)$ $O(n)$
3. Calcul des ensembles Δ_ν $O(n)$
4. Suppression des liens redondants $O(n)$
5. Passage à la matrice de transitions $O(n^2)$

Théorème 6.11 *Nous pouvons calculer l'automate de Glushkov M_E associé à une expression rationnelle E en temps $\Omega(n^2)$ en utilisant un espace mémoire de l'ordre de $O(n^2)$.*

Preuve. La preuve de ce théorème est une déduction des lemmes 6.2, 6.3, 6.8 et 6.10. ∎

6.6 Exemple de construction

L'exemple suivant (figure 12) donne les principales étapes de construction de l'automate de Glushkov.

L'ensemble des arcs de l'automate (figure 13) est donné par l'union des ensembles Δ_{ν_i} disjoints, à laquelle on ajoute les arcs menant d'un état initial unique aux éléments de l'ensemble $\mathit{First}(\nu_0)$.

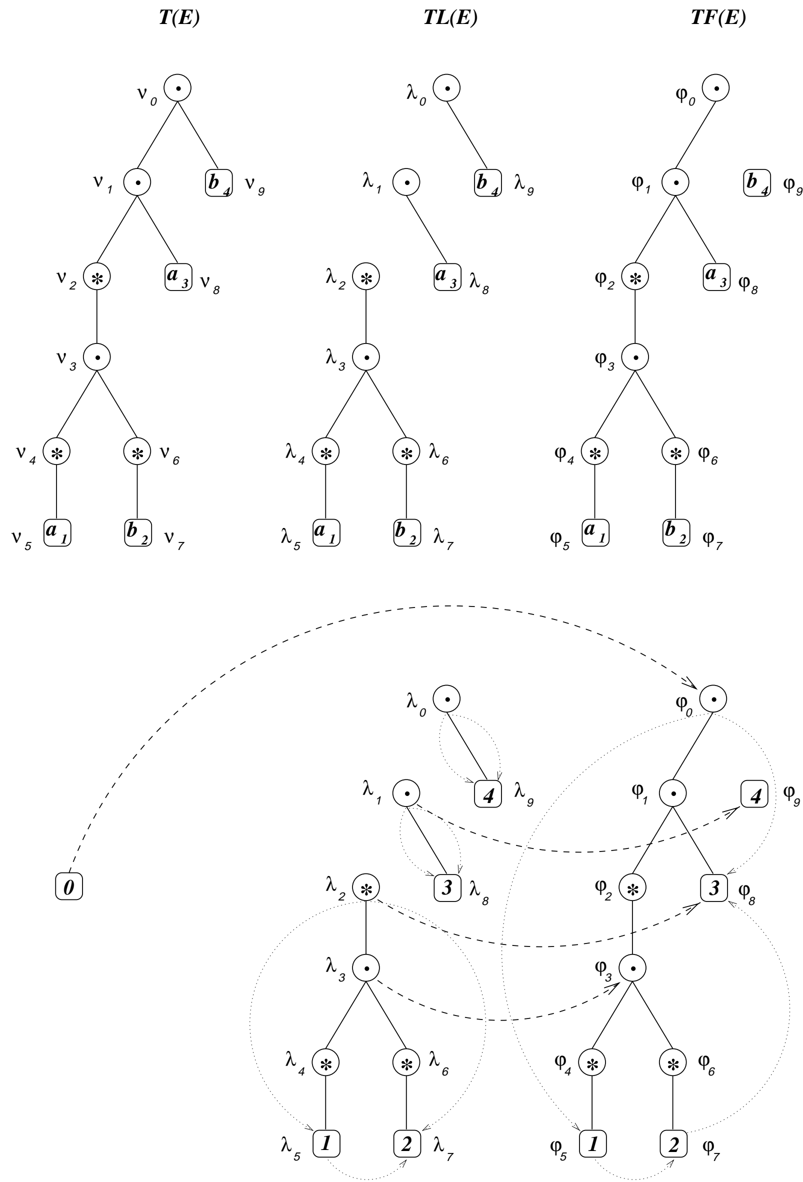
$$\delta = \Delta_{\nu_2} \cup \Delta_{\nu_1} \cup \Delta_{\nu_0} \cup (\{s_I\} \times \mathit{First}(\nu_0))$$
$$\delta = \{(0,1),(0,2),(0,3),(1,1),(1,2),(2,1),(2,2),(1,3),(2,3),(3,4)\}$$

Les états finaux de l'automate sont les éléments de l'ensemble $\mathit{Last}(\nu_0)$, auxquels on ajoute l'état initial si E reconnaît le mot vide.

7 Conclusion

L'algorithme de A. Brüggemann-Klein commence par une phase de prétraitement qui consiste à transformer l'expression en forme normale de fermeture. Le calcul de δ se fait alors en unions disjointes. Notre algorithme calcule une représentation de δ comme un ensemble de produits cartésiens (non évalués) dont seule une union disjointe est finalement calculée. Il fournit une représentation par forêts de l'automate de Glushkov de E en temps linéaire en la taille de E. Un algorithme naïf permet d'en déduire la table de M_E en temps proportionnel au nombre de transitions, donc avec la même complexité que l'algorithme de A. Brüggemann-Klein .

De plus, la nature même de cette structure de données intermédiaire semble intéressante. D. Ziadi en a tiré parti pour construire un algorithme parallèle optimal dans le modèle CREW-PRAM [13] [12].



ν_i	$Last(\nu_i)$	$First(\nu_i)$		Δ_{ν_i}
ν_5	{1}	{1}		$= \emptyset$
ν_4	{1}	{1}	$Last(\nu_5) \times First(\nu_5) = \{1\} \times \{1\}$	$= \{(1,1)\}$
ν_7	{2}	{2}		$= \emptyset$
ν_6	{2}	{2}	$Last(\nu_7) \times First(\nu_7) = \{2\} \times \{2\}$	$= \{(2,2)\}$
ν_3	{1,2}	{1,2}	$Last(\nu_4) \times First(\nu_6) = \{1\} \times \{2\}$	$= \{(1,2)\}$
ν_2	{1,2}	{1,2}	$Last(\nu_3) \times First(\nu_3) = \{1,2\} \times \{1,2\}$	$= \{(1,1),(1,2),(2,1),(2,2)\}$
ν_8	{3}	{3}		$= \emptyset$
ν_1	{3}	{1,2,3}	$Last(\nu_2) \times First(\nu_8) = \{1,2\} \times \{3\}$	$= \{(1,3),(2,3)\}$
ν_9	{3}	{3}		$= \emptyset$
ν_0	{4}	{1,2,3}	$Last(\nu_1) \times First(\nu_9) = \{3\} \times \{4\}$	$= \{(3,4)\}$

Figure 12: exemple de construction sur l'expression $E = (a_1^* b_2^*)^*.a_3.b_4$

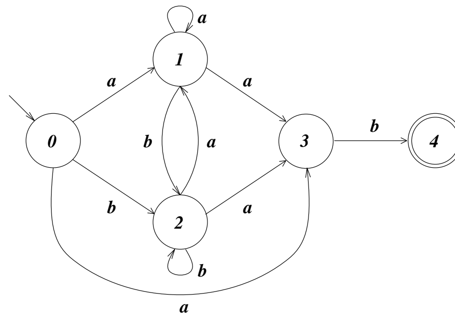


Figure 13: L'automate de Glushkov de l'expression $E = (a_1^* b_2^*)^*.a_3.b_4$

Références

[1] A. Aho, R. Sethi et J-D. Ullman, *Compilateurs : Principes, techniques et outils*, InterEditions, Paris (1991).

[2] A. Aho et J.D. Ullman, *Concepts fondamentaux de l'Informatique*, Dunod, Paris (1992).

[3] J. Berstel et J-E. Pin, Local languages and the Berry-Sethi algorithm, *rapport LITP* **95/13**, IBP (1995).

[4] G. Berry et R. Sethi, From Regular Expression to Deterministic Automata, *Theoret. Comput. Sci.* **48** (1996) 117–126.

[5] D. Beauquier, J. Berstel et Ph. Chrétienne, *Eléments d'algorithmique*, Masson, Paris (1992).

[6] A. Brüggemann-Klein, Regular Expressions into Finite Automata, *Theoret. Comput. Sci.* **120** (1993) 197–213.

[7] J-M. Champarnaud, AUT : un langage pour la manipulation des automates et des semi-groupes finis, Théorie des automates et applications, ed. D. Krob, Deuxièmes journées franco-belges, *Publication de l'Université de Rouen* **176** (1991) 29–43.

[8] J-M. Champarnaud et G. Hansel, AUTOMATE, a computing package for automata and finite semigroups, *Journ. Symbolic Computation* **12** (1991) 197–220.

[9] C.H. Chang et R. Paige, From regular expressions to dfa's using compressed nfa's. *Lecture Notes Comput. Sci.* **644** (1992) 88–108.

[10] V-M. Glushkov, The abstract theory of automata, *Russian Mathematical Surveys* **16** (1961) 1–53.

[11] B.W. Watson, Taxonomies and Toolkits of Regular Language Algorithms, *CIP-DATA Koninklijke Bibliotheek, Den Haag, Ph. D*, Eindhoven University of Technology (1995).

[12] D. Ziadi, Algorithmique parallèle et séquentielle des automates, *Thèse de doctorat*, Université de Rouen, 1996, rapport LIR à paraître.

[13] D. Ziadi et J-M. Champarnaud, Algorithme parallèle efficace de passage d'une expression rationnelle à un automate, *LIR95.10 Informatique Fondamentale*, Université de Rouen (1995), soumis.

Djelloul Ziadi, Jean Luc Ponty et Jean-Marc Champarnaud
Laboratoire d'**I**nformatique de **R**ouen
Université de Rouen
Place E. Blondel
F-76821 Mont-Saint Aignan Cédex (France)
{ziadi,ponty,jmc}@dir.univ-rouen.fr