3 Automate de Glushkov

Dans cette section, on s'intéresse à la construction d'un automate à partir d'une expression régulière par la méthode de Glushkov.

3.1 Langages locaux et expressions rationnelles linéaires

Définition 3.1. Soit L un langage. On dit qu'un langage est *local* si il existe deux parties P et S de Σ et une partie N de Σ^2 tel que

$$L \setminus \{\varepsilon\} = (P\Sigma^* \cap A^*S) \setminus \Sigma^* N \Sigma^*.$$

On peut facilement calculer un automate déterministe qui reconnaît un langage local à partir de P, S et N grâce à la proposition suivante :

Proposition 3.1. *Soit P, S et N les paramètres d'un langage local L.*
L'automate A qui reconnaît L est

$$\mathcal{A} = \langle \Sigma \cup \{I\}, \{I\}, S, \{(1, a, a) | a \in \Sigma\} \cup \{(a, b, b) | (a, b) \in \Sigma^2 / ab \notin N\} \rangle$$

Preuve
Soit $u = a_1 \cdots a_n$ un mot dans L On a $a_n \in S$, $a_1 \in P$ et $\forall i \in [\![1, n]\!], a_i a_{i+1} \notin N$. Par définition de S, P et N, le chemin $1 \xrightarrow{a_1} a_1 \xrightarrow{a_2} a_2 \cdots a_{n-1} \xrightarrow{a_n} a_n$ est accepté par A, donc A accepte u.
Si u est un mot reconnu par A, on a un chemin réussi : $1 \xrightarrow{a_1} a_1 \xrightarrow{a_2} a_2 \cdots a_{n-1} \xrightarrow{a_n} a_n$. Donc, par définition de S, P et N, on a $a_1 \in P$, $a_n \in S$ et $\forall i \in [\![1, n]\!], a_i a_{i+1} \notin N$.
En conclusion, le langage reconnu par A est L. ■

Remarque : Le nombre d'états de cet automate est de $|\Sigma| + 1$.
De plus, l'automate A ainsi construit est un automate local.

Définition 3.2. Un automate déterministe $\mathcal{A} = \langle Q, I, F, \delta \rangle$ sur l'alphabet Σ est dit *local* si

$$\forall a \in \Sigma, Card(\{p \cdot a / p \in Q\}) = 1.$$

Exemple 3.1. Soit L, le langage local défini par $\Sigma = \{a, b, c\}$, $S = \{a, c\}$, $P = \{a, b\}$ et $N = \{ab, bc, ca\}$, alors on a l'automate suivant, qui reconnait L et qui est local.

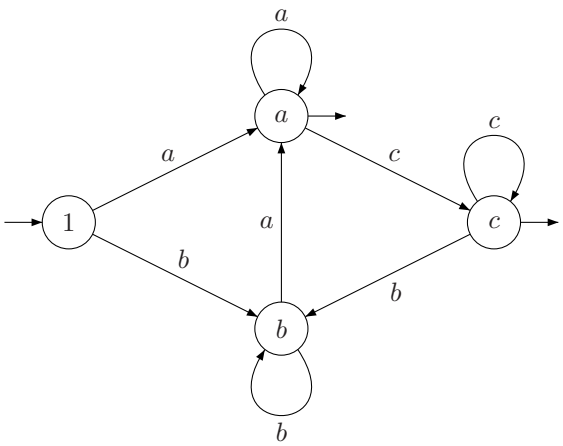

FIG. 1 – Automate local

On a ainsi démontré un sens de la proposition suivante, nous permettant de justifier la construction de Glushkov.

Proposition 3.2. *Il y a équivalence entre :*

1. *L est un langage local.*
2. *L est reconnu par $\mathcal{A} = \langle Q, \{I\}, F, \delta\rangle$, automate local sur Σ.*

Preuve
$1 \implies 2$ d'après la proposition 3.1
$2 \implies 1$ Soit $\mathcal{A} = \langle Q, \{I\}, F, \delta\rangle$ un automate reconnaissant L. On définit
- $P = \{c \in \Sigma / \exists d \in Q / (I, c, d) \in \delta\}$
- $S = \{c \in \Sigma / \exists f, p \in F \times Q / (p, c, f) \in \delta\}$
- $N = \{ab / a, b \in \Sigma /$
- $K = (P\Sigma^* \cap A^*S) \setminus \Sigma^* N \Sigma^*$

Soit $u = a_1 \cdots a_n$ un mot non vide reconnu par $\mathcal{A}$. Il existe un chemin $q_0 \xrightarrow{a_1} q_1 \xrightarrow{a_2} q_2 \cdots q_{n-1} \xrightarrow{a_n} q_n$.
On a donc $a_1 \in P$, et comme $q_n \in F$, on a $a_n \in S$. De plus $\forall i \in [\![1, n]\!], a_i a_{i+1} \notin N$.
En conclusion, $u \in K$, et donc $L \setminus \{\varepsilon\} \subset K$.
Soit $u = a_1 \cdots a_n \in K$ non nul. Par définition de K, on a $a_1 \in P$, $a_n \in S'$. On définit $q_1 = q_0 \cdot a_1$.
Comme $a_1 a_2 \notin N$, $\exists s_0, s_1, s_2 | s_0 \xrightarrow{a_1} s_1 \xrightarrow{a_2} s_2$, et que $\mathcal{A}$ est un automate local avec $q_0 \cdot a_1 = s_0 \cdot a_1$, on a $q_1 = s_1$. On a donc aussi $q_0 \xrightarrow{a_1} q_1 \xrightarrow{a_2} q_2$.
On montre ainsi par récurrence que

$$\forall i \in [\![1, n]\!], q_{i-1} \xrightarrow{a_i} q_i \xrightarrow{a_{i+1}} q_{i+1}$$

Comme $\mathcal{A}$ est local et que $a_n \in S$, on a $q_{n-1} \cdot a_n \in F$ et donc $q_0 \xrightarrow{a_1} q_1 \xrightarrow{a_2} q_2 \cdots q_{n-1} \xrightarrow{a_n} q_n$ est un chemin reconnu par $\mathcal{A}$
En conclusion $L = K$, et donc L est un langage local.

■

Comme les langages rationnels, les langages locaux sont stables par produit, union et étoile :

Proposition 3.3. *Étant donnés L_1 et L_2 langages locaux sur Σ_1 et Σ_2 tels que $\Sigma_1 \cup \Sigma_2 \subseteq \Sigma$ et $\Sigma_1 \cap \Sigma_2 = \emptyset$, alors $L_1 L_2$ et $L_1 \cup L_2$ sont aussi des langages locaux.*

Preuve Grâce à la proposition 3.2, On se ramène à démontrer que l'union, le produit ou l'étoile d'automates locaux est encore un automate local.
Comme on travaille sur deux ensembles d'alphabet disjoints, la méthodes de construction d'un automate produit, étoilé ou union, préservent le caractère local des automates.
On démontre ainsi les deux propositions précédentes. ■

Définition 3.3. Une expression rationnelle p est dite *linéaire* si $\forall x \in \Sigma, |p|_x \leqslant 1$ où $|\cdot|_x$ est le nombre d'occurences du caractère x dans l'expression régulière considérée.

Exemple d'expression linéaire : $p = (e_1|e_3)(e_6^*|e_9 e_{10}^*|e_{13}^*)^*$ est une expression linéaire car chaque caractère de Σ a une occurence d'au plus 1.

L'intérêt de cette notion est justifié par la proposition suivante :

Proposition 3.4. *Toute expression rationnelle linéaire représente un langage local.*

Preuve La preuve se fait par induction structurelle sur l'expression rationnelle linéaire. En effet par définition, le langage qu'elle définit est obtenu par union, produit et étoile de langages (locaux) de la forme $\{e_i\}$, où les e_i sont les lettres apparaissant dans l'expression régulière. Ces langages sont formés sur des alphabets disjoints puisque toutes les lettres sont différentes, la proposition 3.3 s'applique donc. ■

Comme on a pas forcément une expression régulière linéarisée, on utilise la méthode suivante :
- On initialise $\dot{p} = p$.
- Pour chaque caractère $c \in \Sigma$ de p, on note i sa position. Alors, on définit $\mu(i) = c = p[i]$, $\dot{p}[i] = e_i$.

L'expression régulière $\dot{p}$ est alors linéaire, et on revient à p en remplaçant e_i par $\mu(i)$.

Pour l'automate, il suffit de remplacer les étiquettes e_i par $\mu(e_i)$, et on obtient ainsi un automate reconnaissant L définit par p, non déterministe.

Cette opération consiste simplement à remplacer les caractères redondants de p, par de nouveaux caractères non redondants.
Exemple : si $p = (a|b)(a^*|ba^*|b^*)^*$ alors $\dot{p} = (e_1|e_3)(e_6^*|e_9 e_{10}^*|e_{13}^*)^*$

3.2 Algorithme de Glushkov

L'algorithme de Glushkov permet de trouver un automate non déterministe, mais avec un nombre d'états réduit plus réduit que l'algorithme de thompson.

3.2.1 Initialisation

On commence par linéariser l'expression régulière obtenue. On met en forme d'arbre l'expression ainsi obtenue. On applique l'algorithme suivant pour trouver P, L, et F.

3.2.2 Construction de P,L et F

Cela consiste en un parcours linéaire (de gauche à droite) de l'expression, et de mettre à jour les différents ensemble P, S et $F = A^2 \setminus N$:

```
GLUSHKOF(p)
 1  switch
 2    case p = ∅ :
 3         First(p) ← ∅;
 4         Last(p) ← ∅;
 5         Null_p ← ∅
 6    case p = ε :
 7         First(p) ← ∅;
 8         Last(p) ← ∅;
 9         Null_p ← {ε}
10    case p = x :
11         First(p) ← {x}t;
12         Last(p) ← {x};
13         Null_p ← ∅;
14         Follow(p, x) ← ∅
15    case p = p_g|p_d :
16         First(p) ← First(p_d) ⊔ First(p_g);
17         Last(p) ← Last(p_d) ⊔ Last(p_g);
18         Null_p ← Null_{p_d} ⊔ Null_{p_g};
19         for c ∈ Last(p_g)
20         do Follow(p, c) ← Follow(p_g, c) ⊔ First(p_d)
21    case p = p_g · p_d :
22         First(p) ← Null_{p_g} · First(p_d) ⊔ First(p_g);
23         Last(p) ← Null_{p_g} · Last(p_d) ⊔ Last(p_g);
24         Null_p ← Null_{p_d} ∩ Null_{p_g};
25         for c ∈ Last(p_g)
26         do Follow(p, c) ← Follow(p_g, c) ⊔ First(p_d)
27    case p = p_f^* :
28         First(p) ← First(p_f);
29         Last(p) ← Last(p_f);
30         Null_p ← {∅};
31         for c ∈ Last(p_g)
32         do Follow(p, c) ← Follow(p_f, c) ∪ First(p_f)
```

Toutes les unions sont disjointes sauf la dernière, donc on peut réaliser celles-ci en $O(1)$ simplement par concaténation de listes.

On verra dans les optimisations de l'algorithme de Glushkov, que la dernière union peut être faite en temps constant.

Voici le même exemple d'expression, avec la construction de Glushkov : $(a|b)(a^*|ba^*|b^*)^*$