Des expressions rationnelles aux automates

Thibaut Horel

19 décembre 2008

Introduction

Une des implications du théorème de Kleene affirme que pour tout langage rationnel, il existe un automate fini qui reconnaît ce langage. Les automates finis constituent donc le pendant pratique des langages rationnels, en effet on peut déterminer de façon algorithmique l'acceptation d'un mot par un automate. Dans la pratique (écriture d'un analyseur lexical, recherche dans un texte ...), les langages rationnels sont décrits par des expressions rationnelles. Le problème auquel on est confronté est donc de construire un automate fini reconnaissant le langage associé à une expression rationnelle. Il existe de très nombreuses méthodes pour cela (on en trouve dans [6] une classification assez exhaustive). On peut néanmoins distinguer deux approches :

- l'approche *pas à pas* : à partir des automates triviaux qui reconnaissent une unique lettre de l'alphabet, on donne une construction pour chaque opération rationnelle (union, concaténation, étoile) ;
- une approche plus théorique où l'on raisonne sur les langages eux-mêmes.

Nous présentons ici deux méthodes basées sur une approche théorique des langages rationnels. Notre objectif principal sera de prouver la correction des méthodes employées. ici.

On commence par un rappel sur les expressions rationnelles qui permet de fixer le vocabulaire et les notations.

1 Expressions rationnelles

Définition 1.1. — Soient A un alphabet et $\{0, 1, +, \cdot, \star, (,)\}$ des éléments qui n'appartiennent pas à A. On note $\tilde{A} = A \cup \{0, 1, +, \cdot, \star, (,)\}$. On appelle *expression rationnelle* sur l'alphabet A le plus petit ensemble de $\tilde{A}^*$ qui contient A et tel que :

- si E et F sont des expressions rationnelles, $(E) + (F)$ et $(E).(F)$ sont des expressions rationnelles ;
- si E est une expression rationnelle, $E\star$ est une expression rationnelle.

Par souci de clarté on utilisera la convention que $\star$ est prioritaire sur $\cdot$, lui-même prioritaire sur $+$ (pour réduire le nombre de parenthèses). On utilisera également la notation EF pour $E \cdot F$.

Remarque. On peut donc associer à toute expression rationnelle un arbre binaire où les feuilles sont les éléments de $A \cup \{0, 1\}$ et les noeuds les opérations $\{+, \cdot, \star\}$. Ceci permet de montrer des propriétés sur les expressions rationnelles par induction en raisonnant sur la profondeur de l'arbre, que l'on appelle également profondeur de l'expression rationnelle.

Définition 1.2. — On définit pour toute expression rationnelle E, le *langage rationnel associé* $\mathcal{L}(E)$ définit inductivement comme suit :

- $\mathcal{L}(0) = \emptyset$, $\mathcal{L}(1) = \varepsilon = 1_{A^*}$
- $\mathcal{L}(a) = \{a\}$ si $a \in A$
- $\mathcal{L}(E + F) = \mathcal{L}(E) \cup \mathcal{L}(F)$
- $\mathcal{L}(E \cdot F) = \mathcal{L}(E) \cdot \mathcal{L}(F)$
- $\mathcal{L}(E\star) = \mathcal{L}(E)\star$

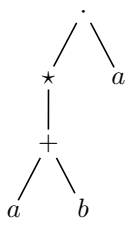

FIG. 1 – Arbre associé à l'expression rationnelle $(a+b)\star a$ sur l'alphabet $\{a,b\}$. On dit que cette expression rationnelle est de profondeur 2. Le langage associé est l'ensemble des mots qui se terminent par a.

REMARQUE. Les expressions rationnelles sont des objets purement formels, elles n'ont de sens que par leurs langages associés. Ainsi deux expressions rationnelles différentes peuvent décrire le même langage (par exemple $(a+b)$ et $(b+a)$). Cependant, on effectuera toujours dans les expressions rationnelles les simplifications suivantes pour éviter des problèmes de définition :

- $0+E=E+0=E$
- $0.E=E.0=0$
- $1.E=E.1=E$

Définition 1.3. — Si E est une expression rationnelle sur A, on définit inductivement $c(E)$, le terme constant de E comme étant l'expression rationnelle définie par :

- $c(1)=1$, $c(0)=c(a)=0$ si $a\in A$
- $c(E+F)=c(E)+c(F)$
- $c(E\cdot F)=c(E)\cdot c(F)$
- $c(E\star)=1$

si l'on effectue dans les calculs les simplifications précédemment introduites, on voit que $c(E)$ est soit 0 soit 1 et que :

$$c(E)=\begin{cases}1 & \text{si } \varepsilon\in\mathcal{L}(E)\\ 0 & \text{sinon}\end{cases}$$

Enfin pour le problème considéré, le paramètre caractéristique de la taille de l'entrée est la longueur d'une expression rationnelle définie comme suit :

Définition 1.4. — La longueur $l(E)$ d'une expression rationnelle E sur un alphabet A est le nombre d'occurrences de lettres de A dans E que l'on calcule inductivement de la façon suivante :

- $l(0)=l(1)=0$
- $l(a)=1$ si $a\in A$
- $l(E+F)=l(E\cdot F)=l(E)+l(F)$
- $l(E\star)=l(E)$

2 Dérivation d'expressions rationnelles et automate des termes dérivés

On étudie ici une transposition aux expressions rationnelles de la notion de langages quotients. Celle-ci conduit à une construction due à Antimirov d'un automate reconnaissant le langage associé à une expression rationnelle donnée.

Définition 2.1. — Soient E une expression rationnelle sur l'alphabet A et $a \in A$, on appelle **B**-dérivée de E par rapport à a et l'on note $\frac{\partial E}{\partial a}$, l'ensemble d'expressions régulières définit inductivement de la façon suivante :

$$\frac{\partial 0}{\partial a} = \frac{\partial 1}{\partial a} = \{0\} \ , \qquad \frac{\partial b}{\partial a} = \begin{cases} \{1\} & \text{si } b = a \\ \{0\} & \text{sinon} \end{cases}$$

$$\frac{\partial E + F}{\partial a} = \frac{\partial E}{\partial a} \cup \frac{\partial F}{\partial a}$$

$$\frac{\partial E \cdot F}{\partial a} = \frac{\partial E}{\partial a} \cdot \{F\} \cup \{c(E)\} \cdot \frac{\partial F}{\partial a}$$

$$\frac{\partial E\star}{\partial a} = \frac{\partial E}{\partial a} \cdot \{E\star\}$$

où l'on étend l'opération $\cdot$ à des ensemble $\mathcal{E}$ et $\mathcal{F}$ d'expressions rationnelles par la formule suivante :

$$\frac{\partial \mathcal{E}}{\partial a} = \left\{ \frac{\partial E}{\partial a} \ : E \in \mathcal{E} \right\}$$

$$\mathcal{E} \cdot \mathcal{F} = \{E \cdot F \ : E \in \mathcal{E}\,, F \in \mathcal{F}\}$$

Exemple. Soit $E = (ab + b) \star ba$. On a :

$$\frac{\partial E}{\partial a} = [(\{b\} \cup \{0\}) \cdot (ab + b) \star \cdot ba] \cup \frac{\partial ba}{\partial a} = \{b(ab + b) \star ba\}$$

De même :

$$\frac{\partial E}{\partial b} = [(\{0\} \cup \{1\}) \cdot (ab + b) \star \cdot ba] \cup \frac{\partial ba}{\partial b} = \{(ab + b) \star ba, a\} = \{E, a\}$$

On étend ensuite naturellement cette définition pour la **B**-dérivée d'un mot.

Définition 2.2. — Soit E une expression rationnelle sur l'alphabet A et $u = va$ un mot non vide sur A, on définit la **B**-dérivée de E par rapport à u par récurrence sur la longueur du mot :

$$\frac{\partial E}{\partial u} = \frac{\partial}{\partial a} \frac{\partial E}{\partial v}$$

où l'on a étendu l'opération $\frac{\partial}{\partial a}$ à un ensemble $\mathcal{E}$ d'expressions rationnelles par la formule :

$$\frac{\partial \mathcal{E}}{\partial a} = \bigcup_{E \in \mathcal{A}} \frac{\partial E}{\partial a}$$

On appelle *terme dérivé* de E toute expression rationnelle non nulle appartenant à un $\frac{\partial E}{\partial v}$ pour un certain v dans A^*.

Pour ne pas perdre de vue la « réalité » associée à ces calculs, on définit naturellement le langage associé à un ensemble d'expressions rationnelles $\mathcal{E}$ par :

$$\mathcal{L}(\mathcal{E}) = \bigcup_{E \in \mathcal{E}} \mathcal{L}(E)$$

On peut maintenant faire le lien entre les **B**-dérivées et les langages rationnels.

Proposition 2.3. —*Soit A un alphabet et L un langage sur A et E une expression rationnelle sur le même alphabet.*

1. *Si $a \in A$:*
$$\mathcal{L}\left(\frac{\partial E}{\partial a}\right) = a^{-1}\mathcal{L}(E) \tag{1}$$
2. *Si $u \in A^*$:*
$$\mathcal{L}\left(\frac{\partial E}{\partial u}\right) = u^{-1}\mathcal{L}(E) \tag{2}$$

Démonstration. On remarque que $E \mapsto \frac{\partial E}{\partial a}$ se calcule de la même manière pour les opérations $\{+, \cdot, \star\}$ que $L \mapsto a^{-1}L$ pour les opérations rationnelles $\{\cup, \cdot, \star\}$ respectivement. De plus $\mathcal{L}$ commute avec les opérations $\{\cup, \cdot, \star\}$ qui apparaissent dans les formules de dérivation ; il suffit de vérifier la première égalité pour les éléments $a \in A$, 0 et 1, ce qui est évident, puis de raisonner par récurrence sur la profondeur de E.

La deuxième partie de la proposition s'obtient par une récurrence évidente sur la longueur du mot en remarquant que $(ua)^{-1}L = a^{-1}(u^{-1}L)$, relation analogue à celle de la définition 2.2. □

On peut se demander pourquoi on n'a pas défini la **B**-dérivée d'une expression rationnelle pour qu'elle soit elle-même une expression rationnelle plutôt qu'un ensemble d'expressions rationnelles. En effet, en remplaçant $\cup$ par un $+$ dans la définition de la **B**-dérivée et en supprimant les accolades, tous les résultats précédents restent valables. Cependant ce choix conduirait à un nombre potentiellement infini d'expression dérivées. Le choix présenté ici est justifié par la proposition suivante, qui montre qu'il n'y a qu'un nombre fini de termes dérivés.

Proposition 2.4. — *Si E est une expression rationnelle, alors E possède au plus $l(E)$ termes dérivés.*

Démonstration. C'est évident par récurrence sur la profondeur de E. □

L'idée est alors de construire un automate ayant pour états les termes dérivés en calquant la construction faite pour avec les langages quotients.

Proposition 2.5. — *Si E est une expression rationnelle sur un alphabet A. On note $E_0 = E$ et $\{E_i : 1 \leq i \leq p\}$ l'ensemble des termes dérivés de E. On pose $Q = \{E_i : 0 \leq i \leq p\}$, $I = \{E_0\}$, $F = \{E_i : c(E_i) = 1\}$ et :*

$$T = \left\{(E_i, a, E_j) : a \in A\,, E_j \in \frac{\partial E_i}{\partial a}\right\}$$

Alors l'automate (Q, A, T, I, F) reconnaît le langage $\mathcal{L}(E)$.

Démonstration. Soit u un mot de A^*, on a l'équivalence :

$$u \in \mathcal{L}(E) \Leftrightarrow \varepsilon \in u^{-1}\mathcal{L}(E) \Leftrightarrow \exists E_i \in \frac{\partial E}{\partial u},\ c(E_i) = 1$$

La dernière propriété signifie exactement que E_i est un état final et qu'il existe un chemin de l'automate étiqueté par u allant de E à E_i. □

Exemple. On considère l'expression rationnelle $E = (ab + b) \star ba$. On a déjà vu :

$$\frac{\partial E}{\partial a} = \{b(ab + b) \star ba\}\ ,\ \frac{\partial E}{\partial b} = \{E, a\}$$

Puis :

$$\frac{\partial b(ab + b) \star ba}{\partial b} = \{E\}\ ,\ \frac{\partial a}{\partial a} = \{1\}$$

On note $E_1 = b(ab + b) \star ba$. On obtient alors l'automate à 4 états suivant :

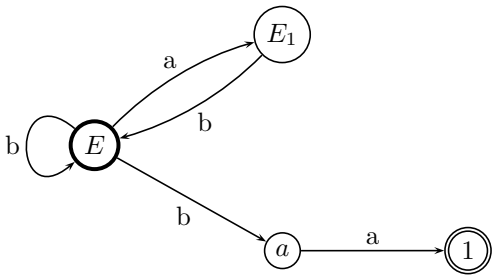

Fig. 2 – Automate des termes dérivés de l'expression $(ab + b) \star ba$.

3 Langages locaux et algorithme de Berry-Sethi

On étudie dans cette partie une sous-classe des langages rationnels particulièrement adaptée à la reconnaissance par automates, les *langages locaux*, puis on montre comment réduire le problème de la reconnaissance d'un langage rationnel à celui d'un langage local.

Soit L un langage sur un alphabet A, on définit :

- $P(L) = \{a \in A : aA^* \cap L \neq \emptyset\}$: les lettres qui sont des premières lettres de mots de L ;
- $F(L) = \{a \in A : A^*a \cap L \neq \emptyset\}$: les lettres qui sont des dernières lettres de mots de L ;
- $S(L) = \left\{u \in A^2 : A^*uA^* \cap L \neq \emptyset\right\}$: les facteurs de longueur de 2 de mots de L.

On voit facilement que pour tout langage L on a :

$$L \setminus \{\varepsilon\} \subset \left[P(L)A^* \cap A^*F(L)\right] \setminus \left[A^*[A^2 \setminus S(L)]A^*\right]$$

Cela signifie simplement que si un mot est dans L, sa première lettre est dans $P(L)$, sa dernière lettre est dans $F(L)$ et tous ses facteurs de longueur 2 sont dans $S(L)$.

Un langage est dit *local* si l'inclusion est en fait une égalité. Plus précisément on a :

Définition 3.1. — Soit L un langage sur l'alphabet A, on dit que L est *local* s'il existe P et F deux parties de A et S une partie de A^2 telles que :

$$L\setminus\{\varepsilon\} = [PA^* \cap A^*F] \setminus \left[A^*(A^2\setminus S)A^*\right]$$

Remarque. Si de telles parties existent, nécessairement $P = P(L)$, $F = F(L)$ et $S = S(L)$.

La terminologie s'explique ainsi : pour vérifier qu'un mot u appartient à un langage local L il suffit de vérifier que sa première lettre est dans P que sa dernière lettre est dans F et que tous ses facteurs de longueur 2 sont dans S. On regarde en quelque sorte le mot à travers une « fenêtre » de taille 2. On comprend alors pourquoi les langages locaux sont adaptés à la reconnaissance par automates : les facteurs de longueur 2 du mot sont les transitions de l'automate. Plus précisément on a :

Proposition 3.2. — *Soit L un langage local sur un alphabet A, on écrit $L\setminus\{\varepsilon\} = [PA^* \cap A^*F] \setminus \left[A^*(A^2\setminus S)A^*\right]$. On pose $Q = \tilde{A} = A \cup \{\varepsilon\}$, $I = \{\varepsilon\}$, $T = \{(\varepsilon, a, a) : a \in P\} \cup \{(a, b, b) : ab \in S\}$ et*

$$E = \begin{cases} F & \text{si } \varepsilon \notin L \\ F \cup \{\varepsilon\} & \text{sinon} \end{cases}$$

Alors l'automate $(Q, \tilde{A}, T, I, E)$ reconnaît L

Démonstration. Évident. □

On remarque que l'automate précédemment construit vérifie la propriété que pour toute lettre $a \in \tilde{A}$ l'ensemble $\{p : (q, a, p) \in T\}$ contient au plus un élément. Il a de plus un unique état initial et aucune transition n'aboutit à celui-ci. Ceci justifie les définitions suivantes :

Définition 3.3. — Un automate (Q, A, T, I, F) est dit *local* si

$$\forall a \in A,\ \mathrm{Card}(\{p : (q, a, p) \in T\}) \leq 1$$

En particulier, un automate local est déterministe.

Un automate est dit *standard* s'il possède un unique état initial et si cet état n'est l'arrivée d'aucune transition.

On montre maintenant une proposition similaire au théorème de Kleene relative à la classe des langages locaux.

Proposition 3.4. — *Soit L un langage, les trois propositions suivantes sont équivalentes :*

1. *L est un langage local*
2. *L est reconnu par un automate standard local*

3. L est reconnu par un automate local

Démonstration. La proposition 3.2 montre $1 \Rightarrow 2$ et $3 \Rightarrow 2$ est évident.

Pour la dernière implication, soit (Q, A, T, I, E) un automate local reconnaissant un langage L. On peut supposer sans perte de généralité que cet automate est émondé et normalisé. On note i l'état initial et on pose :

- $P = \{a \in A : \exists q \in Q, (i, a, q) \in T\}$
- $F = \{a \in A : \exists p \in Q, \exists q \in E, (p, a, q) \in T\}$
- $S = \left\{u = ab \in A^2 : \exists (p, q, r) \in Q^3, (p, a, q) \in T \text{ et } (q, b, r) \in T\right\}$

On va montrer :

$$L \setminus \{\varepsilon\} = (PA^* \cap A^*F) \setminus A^*(A^2 \setminus S)A^*$$

Soit u un mot non vide de L, l'écriture d'un chemin acceptant de l'automate étiqueté par u montre aussitôt que la première lettre de u est dans P, sa dernière lettre est dans F et que ses facteurs de longueur 2 (s'il en a) sont dans S.

Réciproquement soit u un mot dans $(PA^* \cap A^*F) \setminus A^*(A^2 \setminus S)A^*$. Écrivant $u = a_1 \dots a_n$ on montre par récurrence sur j que l'on peut trouver $(q_k)_{1 \leq k \leq j}$ des états de l'automate tels que $i \xrightarrow{a_1} q_1 \xrightarrow{a_2} \dots \xrightarrow{a_j} q_j$. a_1 est dans P donc par définition il existe $q_1 \in Q$ tel que $(i, a_1, q) \in T$. Si on a le résultat pour j, on sait qu'il existe p, q et r avec $p \xrightarrow{a_j} q \xrightarrow{a_{j+1}} r$, mais comme l'automate est local on a $q = q_j$ et on pose donc $q_{j+1} = r$. Enfin il est évident que $q_n \in F$ donc u est reconnu par l'automate et on a l'autre inclusion. □

Cette proposition permet en raisonnant sur des automates de montrer que la classe des langages locaux est stable par certaines opérations.

Proposition 3.5. — *Si A_1 et A_2 sont de parties de A telles que $A_1 \cap A_2 = \emptyset$, si L_1 est un langage local sur l'alphabet A_1 et L_2 un langage local sur l'alphabet A_2 alors les langages $L_1 \cdot L_2$ et $L_1 \cup L_2$ sont locaux. Si L est un langage local sur un alphabet A, alors $L^\star$ est local*

Démonstration. On adapte les constructions vues en cours pour construire un automate standard reconnaissant le produit et l'union de deux langages pour lesquels on a deux automates standards. On fait de même pour l'étoile. Ces constructions conservent le caractère local des automates et on a la propriété. □

Tous les langages rationnels ne sont pas locaux. On peut cependant donner un critère simple pour vérifier si un langage rationnel est local. Pour cela on a besoin de la définition suivante :

Définition 3.6. — Une expression rationnelle E sur un alphabet A est dite *linéaire* si pour tout $a \in A$ le nombre d'occurrences de a dans E est au plus 1.

Proposition 3.7. — *Si E est une expression rationnelle linéaire alors $\mathcal{L}(E)$ est local*

Démonstration. C'est évident par récurrence sur la profondeur de E car les unions et produits se font alors sur des alphabets disjoints. □

On confondra désormais E et $\mathcal{L}(E)$ pour alléger les notations.

Pour adapter la théorie des langages locaux à notre problème on va commencer par se ramener à des expressions rationnelles linéaires. On introduit l'opération de *linéarisation* :

Définition 3.8. — Soit E une expression rationnelle, on dit qu'on *linéarise* E lorsque l'on remplace chaque occurrence de lettres dans E par sa position dans l'expression rationnelle (en lisant de gauche à droite). De plus pour que cette opération soit non destructive on garde dans un tableau la lettre associée à chaque position.

Si E est une expression rationnelle, on note E' l'expression obtenue à partir de E par linéarisation. Le tableau d'association définit une fonction Pos qui à chaque position dans E' associe la lettre correspondante dans E.

Exemple. Si $E = (ab + b) \star ba$, $E' = (1 \cdot 2 + 3) \star 4 \cdot 5$ est l'expression obtenue par linéarisation à partir de E. On a le tableau d'association suivant :

n	1	2	3	4	5
$Pos(n)$	a	b	b	b	a

Si E' est une expression rationnelle linéaire sur un alphabet A, il nous faut aussi pouvoir calculer les ensembles $P(E')$, $F(E')$ et $S(E')$. En fait, au lieu de calculer ce dernier, on préfère calculer une fonction $Succ$ telle que $Succ(E', i) = \{j \in A : ij \in S(E')\}$. Tout cela se calcule facilement par récurrence sur la profondeur de E' :

- Si $E' = 0$ ou $E' = 1$, on a $P(E') = F(E') = \emptyset$ et $Succ(E', \cdot) = \emptyset$.
- Si $E' = i$ avec $i \in A$, $P(E') = F(E') = \{i\}$ et $Succ(E', \cdot) = \emptyset$.
- Si $E' = M \cdot N$:

$$P(E') = \begin{cases} P(M) & \text{si } c(M) = 0 \\ P(M) \cup P(N) & \text{sinon} \end{cases}$$

$$F(E') = \begin{cases} F(N) & \text{si } c(N) = 0 \\ F(M) \cup F(N) & \text{sinon} \end{cases}$$

$$Succ(E', i) = \begin{cases} Succ(M, i) \cup Succ(N, i) \cup P(N) & \text{si } i \in F(M) \\ Succ(M, i) \cup Succ(N, i) & \text{sinon} \end{cases}$$

- Si $E' = M + N$:

$$P(E') = P(M) \cup P(N)\ ,\ F(E') = F(M) \cup F(N)$$

$$Succ(E', i) = Succ(M, i) \cup Succ(N, i)$$

- Si $E' = M\star$:

$$P(E') = P(M)\ ,\ F(E') = F(M)$$

$$Succ(E', i) = \begin{cases} Succ(M, i) \cup P(E) & \text{si } i \in F(E) \\ Succ(M, i) & \text{sinon} \end{cases}$$

Une première méthode consiste alors à construire l'automate reconnaissant E' (comme dans la proposition 3.2) puis à remplacer les étiquettes de l'automate obtenu par les lettres correspondantes de l'expression E (à l'aide de la fonction Pos). La même lettre peut correspondre à plusieurs étiquettes, on perd donc à cette étape le caractère déterministe de l'automate. Cette méthode est

l'algorithme de Berry-Sethi. D'après la proposition 3.2 l'automate reconnaissant E' a $l(E)+1$ états (en effet E' est une expression sur l'alphabet $\{1,\ldots,l(E)\}^*$). L'automate obtenu par l'algorithme de Berry-Sethi a donc $l(E)+1$ états. Cette méthode est mise en œuvre dans l'exemple ci-dessous.

EXEMPLE. On repart de l'expression $E = (ab+b)\star ba$ ci-dessus. On a $E' = (1\cdot 2+3)\star 4\cdot 5$, $P(E') = \{1,3,4\}$, $F(E') = \{5\}$, et la fonction $c \mapsto Succ(E',c)$ est donnée par le tableau suivant :

c	1	2	3	4	5
$Succ(E',c)$	$\{2\}$	$\{1,3,4\}$	$\{1,3,4\}$	$\{5\}$	$\emptyset$

On obtient donc l'automate suivant qui reconnaît $\mathcal{L}(E')$

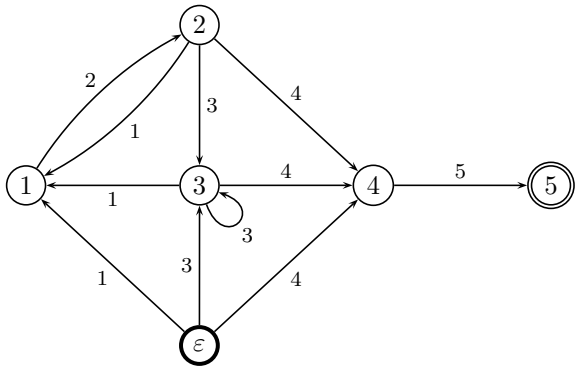

FIG. 3 – Automate local reconnaissant l'expression linéaire $E' = (1\cdot 2+3)\star 4\cdot 5$.

Puis en remplaçant les positions par les lettres correspondantes de E, on obtient l'automate (non déterministe) suivant :

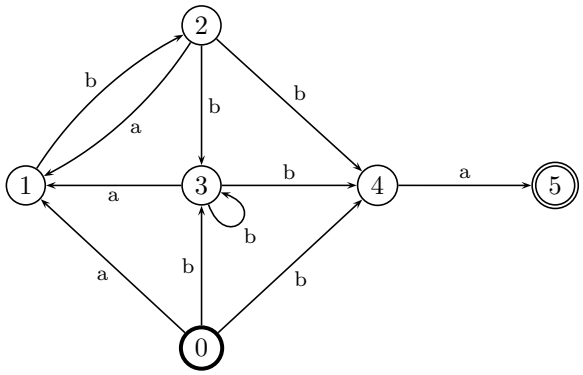

FIG. 4 – Automate reconnaissant l'expression rationnelle $E = (ab+b)\star ba$.

4 Algorithme de Glushkov

On peut obtenir directement un automate déterministe par *l'algorithme de Glushkov*. Cette algorithme est celui décrit dans [1], il est très utilisé dans les analyseurs lexicaux.

Si E est l'expression rationnelle pour laquelle on souhaite construire un automate, on commence par linéariser l'expression (comme dans la partie précédente). On ajoute ensuite le caractère 0 à la fin de l'expression rationnelle obtenue, il signifie qu'il n'y a plus rien à lire. Les états de l'automate sont alors des ensembles de lettres de E' ; ils représentent l'ensemble des caractères que l'on peut lire à un instant donné. L'état initial est $P(E')$, les états finaux sont tous les états contenant 0. Et on dit que (p, c, q) est une transition de l'automate si et seulement si :

$$q = \left\{ \bigcup_i Succ(E', i) : i \in Pos^{-1}(c) \right\}$$

On présente cette méthode dans l'exemple ci-dessous.

Exemple. On prend toujours $E = (ab + b) \star ba$. On travaille sur l'expression rationnelle linéaire $E' = (1 \cdot 2 + 3) \star 4 \cdot 5 \cdot 0$. On a $P(E') = \{1, 3, 4\}$, $F(E') = \{5\}$, et la fonction $c \mapsto Succ(E', c)$ est donnée par le tableau suivant :

c	1	2	3	4	5
$Succ(E', c)$	$\{2\}$	$\{1, 3, 4\}$	$\{1, 3, 4\}$	$\{5\}$	$\{0\}$

On a également $Pos^{-1}(a) = \{1, 5\}$ et $Pos^{-1}(b) = \{2, 3, 4\}$.

On calcule de proche en proche la table de transition :

	a	**b**
$\{1, 3, 4\}$	$\{2\}$	$\{1, 3, 4, 5\}$
$\{2\}$		$\{1, 3, 4\}$
$\{1, 3, 4, 5\}$	$\{2, 0\}$	$\{1, 3, 4, 5\}$
$\{2, 0\}$		$\{1, 3, 4\}$

On obtient l'automate (déterministe) suivant :

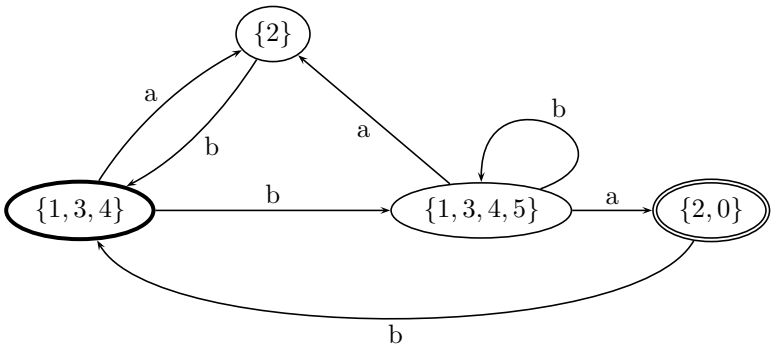

Fig. 5 – Automate reconnaissant l'expression rationnelle $E = (ab + b) \star ba$.

Remarque. On voit pour cet exemple précis que l'automate produit par l'algorithme de Glushkov a moins d'états que l'automate produit par l'algorithme de Berry et Sethi. Ce n'est pas toujours le cas (voir conclusion).

Conclusion

On a montré quelques méthodes pour construire un automate à partir d'une expression rationnelle. L'étape suivante est de savoir si un mot donné est reconnu ou non par l'automate obtenu. Ceci est très facile lorsque l'automate construit est déterministe (il s'agit d'un algorithme linéaire en la longueur du mot). Lorsque l'automate est non déterministe, on procède comme suit en associant à chaque lettre du mot un ensemble d'états de l'automate.

Soit $(Q, A, T, \{i\}, F)$ un automate non déterministe et $u = a_1 \dots a_n$, on note $E_1 = \{q : (i, a_1, q) \in T\}$. Si E_i est l'ensemble d'états associé à la lettre a_i, alors on pose :

$$E_{i+1} = \bigcup_{p \in E_i} \{q : (p, a_{i+1}, q) \in T\}$$

Il est alors clair que u est reconnu par l'automate si et seulement si E_n contient un état final de l'automate. Il s'agit en quelque sorte d'une *déterminisation paresseuse*.

On voit qu'il est plus difficile de tester l'acceptation d'un mot par un automate non déterministe. Il est donc naturel de remettre en question l'utilité de construire un automate non déterministe. Sans entrer dans une discussion quantitative on peut avancer quelques éléments de réponse.

On sait que déterminiser un automate à n états peut dans le pire des cas conduire à un automate à 2^n états. Donc construire directement un automate déterministe (comme dans l'algorithme de Glushkov) peut entraîner une croissance exponentielle du nombre d'états. Dans ce cas, tester l'acceptation d'un mot peut devenir plus rapide pour l'automate non déterministe.

On choisit alors en fonction du cadre d'application. Par exemple dans un compilateur l'automate est construit une fois pour toute et on doit tester de nombreuses fois l'acceptation d'un mot, donc on peut se permettre de perdre un peu de temps au moment de la construction de l'automate.

Références

[1] Alfred V. Aho, Monica S. Lam, Ravi Sethi et Jeffrey D. Ullman : *Compilers : Principles, Techniques & Tools*, chapitre 3, pages 147–186. Addison Wesley, seconde édition, 2006.

[2] Jean Berstel et Jean-Éric Pin : Local languages and the berry-sethi algorithm. Février 1995.

[3] Olivier Carton : *Langages formels, Calculabilité et Complexité.* Vuibert, Octobre 2008.

[4] J.-M. Champarnaud, J.-L. Ponty et D. Ziadi : From regular expressions to finite automata. *International journal of computer mathematics*, 72(4): 415–431, 1999.

[5] Jacques Sakarovitch : *Éléments de théorie des automates*, chapitre 1, pages 132–168. Vuibert, Février 2003.

[6] Bruce William Watson : *Taxonomies and Toolkits of Regular Language Algorithms.* Thèse de doctorat, Eindhoven University of Technology, 1995.