Rapport sur les expressions rationnelles (régulières) et les automates

Praden Florian

4 juin 2007

Table des matières

1 Langages rationnels et expressions rationnelles

1.1 Langages rationnels

Les langages rationnels sont des langages définis de façon inductive. Soit Σ un alphabet, $\mathcal{L}$ l'ensemble des langages rationnels sur Σ est la plus petite classe qui satisfasse les propriétés suivantes :

- $\varepsilon \in \mathcal{L}$
- Pour tout $c \in \Sigma, \{c\} \in \mathcal{L}$
- Si $L_1, L_2 \in \mathcal{L}$ alors
  - $L_1 \cup L_2 \in \mathcal{L}$
  - $L_1.L_2 \in \mathcal{L}$
- Si $L \in \mathcal{L}$ alors $L^* \in \mathcal{L}$ avec $L^* = \bigcup_{n\in\mathbb{N}} L^k$ et $\begin{cases} L^0 = \{\varepsilon\} \\ L^k = LL^{k-1} \quad \text{si} \quad k \neq 0 \end{cases}$

avec ε désignant le mot vide.

1.2 Expressions régulières

Un motif p est défini ainsi :

- () est un motif dit motif vide
- Si c est un caractère ($c \in \Sigma$)
- Si p_1, p_2 sont des motifs, alors
  - $p_1|p_2$ est un motif (l'alternative)
  - p_1p_2 est un motif (la concaténation)
- Si p est un motif, alors
  - $p*$ est un motif
  - (p) est un motif

Une expression régulière sur Σ est définie par un motif. Ce sont les expressions régulières de base. Sur cette construction, on peut ajouter d'autres formes de caractères spéciaux. On a alors les expressions régulières étendues, que l'on rencontre dans beaucoup de logiciels. On note $\mathcal{R}$ l'ensemble des expressions régulières.

1.3 Expressions régulières étendues

Sur la base des expressions régulières définies ci-dessus, on peut ajouter certaines autres opérations. Chacune des opérations suplémentaires peut être implémentée par les expressions régulières de base. Elles n'augmentent donc pas la classe des expressions régulières classiques.

On a donc les opérations suivantes en plus :

Expression régulière étendue	Expression régulière classique
.	Σ
$p?$	$(p\|)$
$p+$	$p(p*)$
$p\{n\}$	$\underbrace{pp\cdots p}_{n \text{ fois}}$
$p\{n,\}$	$\underbrace{pp\cdots p}_{n \text{ fois}}(p*)$
$p\{n,m\}$	$\underbrace{pp\cdots p}_{n \text{ fois}}(\|p\|pp\|\cdots\|\underbrace{pp\cdots p}_{m-n \text{ fois}})$
$[c_1c_2c_3\cdots c_n]$	$(c_1\|c_2\|\cdots\|c_n)$
$[c_1 - c_n]$	$(c_1\|c_2\|\cdots\|c_n)$
$[\hat{}\, c_{i_1}c_{i_2}\cdots c_{i_k}]$	$(c_{i_{k+1}}\|c_{i_{k+2}}\|\cdots\|c_{i_m})/m = \|\Sigma\|$

L'opérateur $[c_1 - c_n]$ suppose un ordre sur Σ.

De plus, on peut définir les classes de caractères :

Soit $\{d_0, d_1, \ldots, d_k\} = \Theta \in \mathfrak{P}(\Sigma)$. On a alors [: Θ :], qui est défini par $(d_0|d_1|\cdots|d_k)$.

Les expressions régulières étendues forment donc un langage commode pour une recherche de texte avancée.

1.4 Interêt de l'utilisation d'expression régulière

Les expressions régulières sont très souvent utilisées dans le monde l'informatique. En effet, cela permet de rechercher de façon assez précise une sous-chaîne d'un texte. Les interêts sont multiples :

- La recherche de textes dans des éditeurs (Pattern Matching)
- La description de langage (rationnel)
- La description de mot clef d'une grammaire (cf. lex et yacc)
- La réponse automatique (exemple : procmail , . . .)
- grep, find, . . .
- etc. . .

Dans la plupart de ces cas, la rapidité de la recherche est importante : en effet, le parsage d'un mail par procmail ne doit pas être trop long, car sinon, adieu les mails importants. Pareil si on veut rechercher du texte dans un fichier : on ne va pas attendre une heure que le logiciel trouve notre sous-chaîne correcte.

1.5 «Équivalence» entre langage rationnel et expression régulière

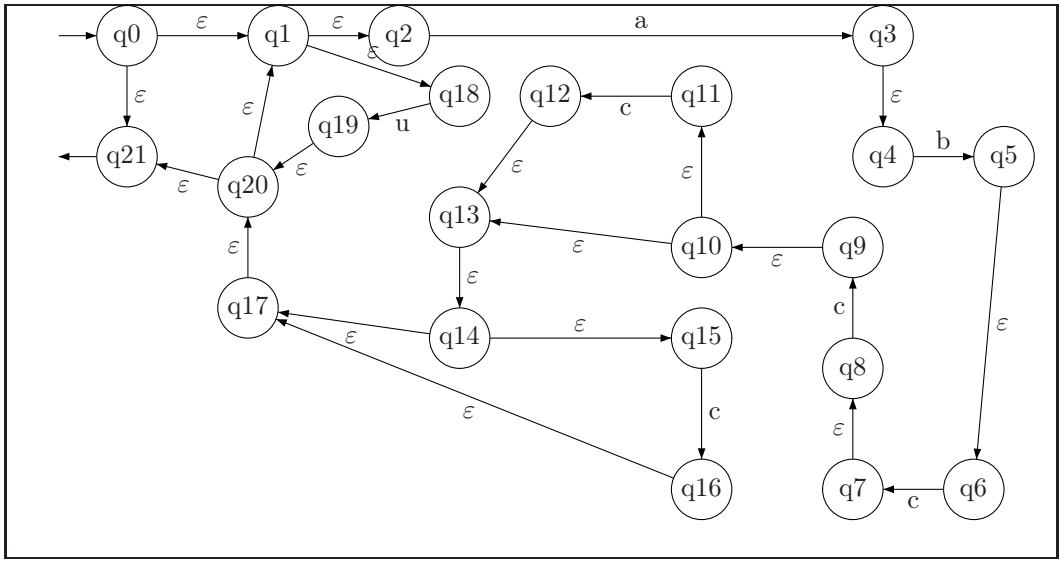
Un langage rationnel est donc potentiellement un ensemble infini de mots, mais tout langage rationnel peut être construit à partir d'un ensemble de singletons par une suite finie d'opérations '.', '+' et '*'. On peut définir le langage rationnel de l'expression régulière p par : $\mathcal{L}(p) = \mu(p)$ avec $\mu : \mathcal{R} \to \mathcal{L}$
μ est défini par :

- $\mu(()) = \{\varepsilon\}$
- $\mu(c) = \{c\}$
- Si p_1, p_2 sont des motifs, alors
  - $\mu(p_1|p_2) = \mu(p_1) \cup \mu(p_2)$
  - $\mu(p_1 p_2) = \mu(p_1)\mu(p_2)$
- Si p est un motif, alors
  - $\mu(p*) = \mu(p)^*$
  - $\mu((p)) = \mu(p)$

Dans la suite de l'exposé, on ne fera plus de différence entre expression régulière, et expression rationnelle (ce n'est qu'une convention de notation des opérateurs).

Cependant, deux expressions régulières différentes peuvent avoir le même langage :
Exemple :
$\mu((a|aa|aaaa*)) = \mu(aa*)$ et $(a|aa|aaaa*) \neq aa*$

Exemple d'expression régulière (abc{2,4} | u)* et automate associé



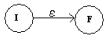
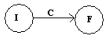
Construction par la méthode de Thompson.

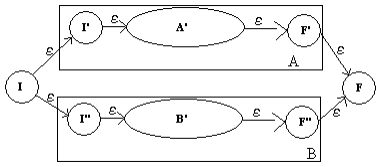
2 Construction de l'automate reconnaissant l'expression régulière p par l'algorithme de Thompson

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

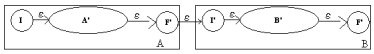
Cela consiste à construire l'automate avec des ε-transitions.

On construit l'automate récursivement :

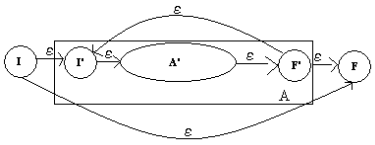
- ε est reconnu par 
- c est reconnu par 
- Si p_1, p_2 sont des motifs reconnus par les automates A et B, alors
  - $(p_1|p_2)$ l'est par



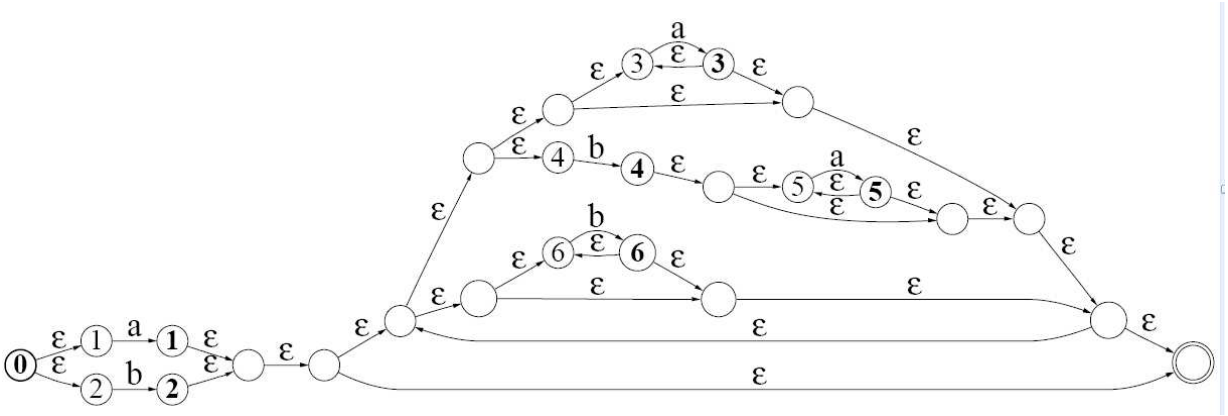
  - (p_1p_2) l'est par



- Si p est un motif reconnu par l'automate A, alors $p*$ l'est par



Voici un exemple de construction avec l'expression $(a|b)(a^*|ba^*|b^*)^*$



On voit donc que la construction est assez mauvaise au niveau du nombre d'états.

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.

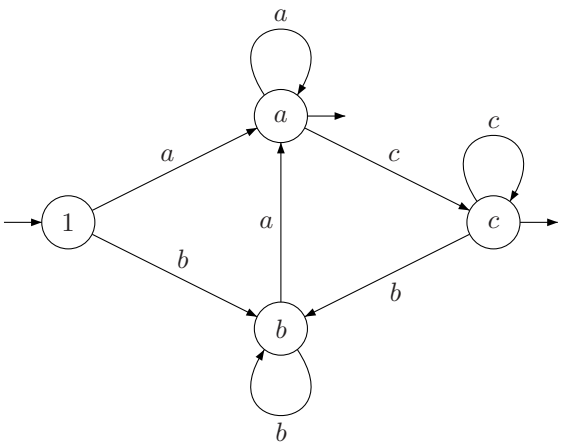


FIG. 1 – Automate local

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

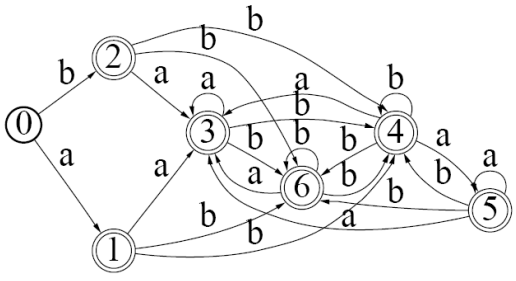
Cela consiste en un parcours linéaire (de gauche à droite) de l'expression, et de mettre à jour les différents ensemble P, S et $F = A^2 \setminus N$:

```
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 : $(a|b)(a^*|ba^*|b^*)^*$



3.3 Optimisation de Brüggemann-Klein

Cette optimisation se base sur la décomposition

$$Follow(p,x) = \ Follow(p_f,x) \setminus First(p_f) \ \sqcup First(p_f)$$

qui permet de réaliser l'union comme réunion disjointe.

Définition 3.4. E est sous forme normale de fermeture si $\forall H/H^*$ sous expression de E , $\forall c \in Last(H), Follow(H,c) \cap First(H) = \emptyset$

Cela permet à l'opération de fermeture d'être réalisée par des unions disjointes.
Il faut donc construire :
- $\dot{E}$ sous forme normale de fermeture par récurence sur E :
  - $E = \emptyset$ ou ε ou $c \to \dot{E} = E$;
  - $E = E_g \mid E_d \to \dot{E} = \dot{E_g} \mid \dot{E_d}$;
  - $E = E_g \cdot E_d \to \dot{E} = \dot{E_g} \cdot \dot{E_d}$;
  - $E = E_f^* \to \dot{E} = \dot{E_f}^{\circ^*}$.
- $A_g(E^\circ)$ qui est égale à $A_g(E)$.

On définit H° récursivement par :
- $H = \emptyset$ ou $\varepsilon \to H^\circ = \emptyset$;
- $H = c \to H^\circ = c$;
- $H = H_g \mid H_d \to H^\circ = H_g^\circ \mid H_d^\circ$;
- $H = H_g \cdot H_d \to H^\circ = \begin{cases} H_g \cdot H_d & \text{si } Null_{H_g} = \emptyset \text{ et } Null_{H_d} = \emptyset \\ H_g^\circ \cdot H_d & \text{si } Null_{H_g} = \emptyset \text{ et } Null_{H_d} = \{\varepsilon\} \\ H_g \cdot H_d^\circ & \text{si } Null_{H_g} = \{\varepsilon\} \text{ et } Null_{H_d} = \emptyset \\ H_g^\circ \mid H_d^\circ & \text{si } Null_{H_g} = \{\varepsilon\} \text{ et } Null_{H_d} = \{\varepsilon\} \end{cases}$
- $H = H_f^* \to H^\circ = H_f^\circ$.

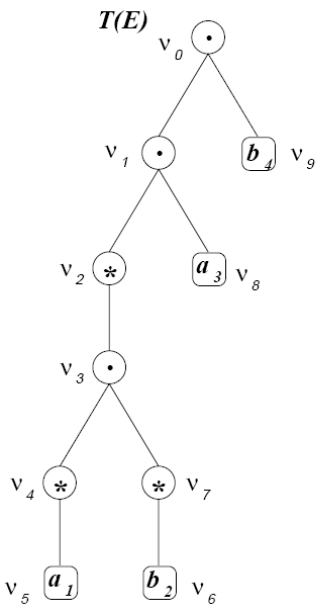
On peut transformer E en E° en temps linéaire grâce à la définition récursive de E°.
Grâce à cette optimisation, on a le théorème suivant :

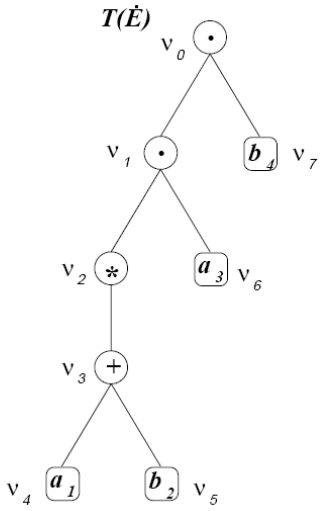
Théorème 3.5. *L'automate de Glushkov $A_g(E)$ d'une expression rationnelle E peut être calculé en $\Omega(n)$, en passant par $\widetilde{E}$*

Exemple avec $E = (a^*b^*)^*ab$ et donc $\widetilde{E} = (a|b)^*ab$

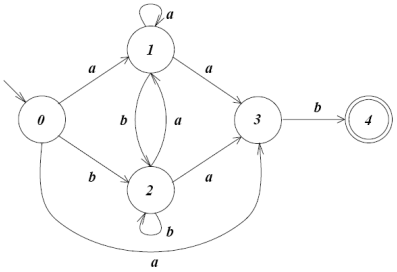
$$\begin{aligned}
\widetilde{E} = \ & \widetilde{(a^*b^*)^*ab} = & \widetilde{(a^*b^*)^*}ab = & & \widetilde{(a^*b^*)^*}ab \\
= \ & \widetilde{(a^*b^*)}^{\circ^*}ab = & (\widetilde{(a^*)}(b^*))^{\circ^*}ab = & & (a^{\circ^*}b^{\circ^*})^{\circ^*}ab \\
= \ & (a^*b^*)^{\circ^*}ab = & (a^{*^\circ}b^{*^\circ})^*ab = & & (a^\circ b^\circ)^*ab \\
\widetilde{E} = \ & (a \mid b)^*ab
\end{aligned}$$

Arbre représentant E et $\widetilde{E}$:





Et l'automate $A_g(E)$



3.4 Optimisation de Chang-Paige

Cette optimisation se base sur l'autre décomposition

$$Follow(p, x) = Follow(p_f, x) \sqcup \ First(p_f) \setminus Follow(p_f, x)$$

qui permet de réaliser l'union comme réunion disjointe.

On remplace ainsi la définition de la fonction de transition δ de Glushkov pour l'étoile (qu'il définit ainsi $\delta_{p^*} = \delta_p \cup (Last(p) \times First(p))$) par $\delta_{p^*} = \delta_p \cup NRed(p)$ où $NRed(p) = (Last(p) \times First(p)) \setminus \delta_p$.

Ceci permet de ne considérer que des unions disjointes.

On définit ainsi $NRed$ par récurrence :

- $E = \emptyset$ ou $\varepsilon \to NRed(E) = \emptyset$;
- $E = c \to NRed(E) = First(c) \times Last(c)$;
- $E = E_g \mid E_d \to NRed(E) = NRed(E_g) \cup NRed(E_d) \cup Last(E_g) \times First(E_d) \cup Last(E_d) \times First(E_g)$;
- $E = E_g \cdot E_d \to NRed(E) = Null_{E_g} \cdot Nred(E_d) \cup Null_{E_d} \cdot Nred(E_g) \cup Last(E_d) \times First(E_g)$;
- $E = E_f^* \to NRed(E) = \emptyset$.

4 Construction d'Antimirov

4.1 Termes dérivés

Le but de cette nouvelle notion sera justifé par la proposition 4.1 et surtout 4.2, qui relie les termes dérivés de E aux résiduels du langage $\mathcal{L}(E)$.

L'appellation de «terme dérivé» provient de la similitude avec la dérivation d'une expression formelle.

Cependant, Antimirov l'a légèrement modifiée permettant de ne plus avoir une infinité de termes dans certains cas pathologiques.

On définit la $\mathbb{B}$-dérivée d'une expression E sur Σ par rapport à $a \in \Sigma$ (que l'on note $\partial_a(E)$) récursivement par :

- $\partial_a(0) = \partial_a(1) = 0$
- $\forall a, b \in \Sigma, \partial_a(b) = \begin{cases} 1 & \text{si} \quad b = a \\ 0 & \text{sinon} \end{cases}$
- $\partial_a(E_g|E_d) = \{\partial_a(E_g)\} \cup \{\partial_a(E_d)\}$
- $\partial_a(E_g \cdot E_d) = \{[\partial_a(E_g)] \cdot E_d\} \cup \quad Null_{E_g} \cdot \partial_a(E_d)$
- $\partial_a(E_f^*) = \left\{[\partial_a(E_f)] \cdot E_f^*\right\}$

Définition 4.1. Soit E une expression régulière. On étend la définition de dérivée à un mot par :
Si $g, f, a \in (\Sigma^*)^2 \times \Sigma$ tel que $g = f \cdot a$ alors $\partial_{fa}(E) = \partial_a(\partial_f(E))$

Si $\mathcal{E}$ est un ensemble d'expression, on notera $\mathcal{L}(\mathcal{E})$ la réunion des $\mathcal{L}(E)$ pour $E \in \mathcal{E}$

Proposition 4.1. $\forall E \in Rat(\Sigma), \forall a \in \Sigma, \quad \mathcal{L}(\partial_a(E)) = a^{-1}\mathcal{L}(E)$

Preuve
On raisonne par induction structurelle sur l'expression E :

- La propriété est triviale pour $E = 0, 1$ ou $b \in \Sigma$
- Si $E = E_g|E_d$, on doit montrer que $a^{-1}\mathcal{L}(E) = a^{-1}\mathcal{L}(E_g) \cup a^{-1}\mathcal{L}(E_d)$, ce qui est vrai.
- Si $E = E_f^*$, on doit montrer que $a^{-1}\mathcal{L}(E_f)^* = a^{-1}\mathcal{L}(E_f)\mathcal{L}(E_f^*)$, car un mot de $\mathcal{L}(E_f^*)$ commençant par a est la concaténation d'un mot de $\mathcal{L}(E_f)$ commençant par a avec un mot de $\mathcal{L}(E_f^*)$.
- Si $E = E_g \cdot E_d$, on doit montrer que $a^{-1}\mathcal{L}(E_g \cdot E_d) = a^{-1}\mathcal{L}(E_g) \cdot E_d \cup a^{-1}\mathcal{L}(E_d)$, ce qui est vrai

■

Proposition 4.2. $\forall E \in Rat(\Sigma), \forall f \in \Sigma^*, \mathcal{L}(\partial_f(E)) = f^{-1}\mathcal{L}(E)$

Preuve On raisonne par récurrence sur la longueur du f. Si $f = \varepsilon$, il n'y a rien à démontrer. Si $\mathcal{L}(\partial_f(E)) = f^{-1}\mathcal{L}(E)$ alors $\mathcal{L}(\partial_{fa}(E)) = \mathcal{L}(\partial_a(\partial_f(E))) = a^{-1}\mathcal{L}(\partial_f(E))$ d'après la proposition précédente, ce qui vaut $a^{-1}f^{-1}\mathcal{L}(E) = (fa)^{-1}\mathcal{L}(E)$. ■

Définition 4.2. On note $l(E)$ pour une expression régulière E sur E, le nombre d'occurences des lettres de Σ.

On remarque que $l(E)$ se calcule bien par induction sur l'expression regulière E

Proposition 4.3. *Une expression regulière E a au plus $l(E)$ termes dérivés sans compter le* 0.

Preuve On raisonne par induction structurelle sur E.
C'est vrai si $E = 0, 1$ ou $a \in \Sigma$.
Si $E = E_g|E_d$, les termes dérivés de E sont ceux de E_g (au plus $l(E_g)$) et ceux de E_d (au plus $l(E_d)$). Donc au total il y en a au plus $l(E_d) + l(E_g) = l(E)$.
Si $E = E_g \cdot E_d$, les termes dérivés de E sont ceux de E_g concaténés avec E_d (au plus $l(E_g)$) et éventuellement ceux de E_d (au plus $l(E_d)$). Donc au total il y en a au plus $l(E_d) + l(E_g) = l(E)$.
Si $E = E_f^*$, les termes dérivés de E sont ceux de E_f concaténés avec E_f^* (au plus $l(E_f)$). Donc au total il y en a au plus $l(E_f) = l(E)$. ■

Théorème 4.4. *Soit E une expression régulière. Les termes dérivés de E forment un ensemble P d'expressions $\{K_p\}_{p \in P}$ telle que*

- $Card(P) \leqslant l(E)$
- $\forall a \in \Sigma, \exists Q^{(a)} \subset P, \{H_p^{(a)}\}_{p \in P} \subset \mathfrak{P}(P)$ *tel que :*
  - $\partial_a(E) = \quad_{p \in Q^{(a)}} K_p$
  - $\forall p \in P \quad \partial_a(K_p) = \quad_{j \in H_p^{(a)}} K_j$

Preuve La preuve se fait de manière analogue à la preuve précédente(par induction structurelle), en travaillant sur les ensembles déduits de la récurrence, pour en construire de nouveaux, qui satisfont aux conditions de la proposition. ■

4.2 Construction de l'automate

Grâce au théorème d'Antimirov (cf. Théorême 5.1), on a une construction d'un automate reconnaissant $\mathcal{L}(E)$.

Définition 4.3. Soient E une expression régulière, et P_E l'ensemble de ses termes dérivés. On pose $P' = P_E \cup \{E\}, E = K_i$ et $Q^{(a)} = H_i^{(a)}$ avec $i \in P'$.

L'automate $\mathcal{A}_E = \langle P', \{E\}, F_E, \delta_E \rangle$ défini par :

$\delta_E = \left\{(K_p, a, K_q \partial_a(F)) | p, a \in P' \times \Sigma, q \in H_p^{(a)}\right\}$

et

$F_E = \{K_p \in P' | Null_{K_p}\}$

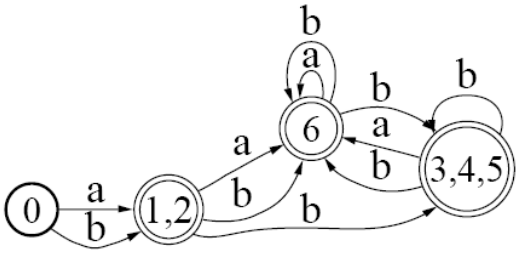
est appelé automate des dérivés de E

Proposition 4.5. *Soit une expression regulière E. L'automate des dérivés de E reconnaît $\mathfrak{L}(E)$:*

$\mathfrak{L}(A_E) = \mathfrak{L}(E)$

Preuve D'après la proposition 4.2, l'automate des dérivés n'est rien d'autre que l'automate des résiduels de $\mathfrak{L}(E)$, qui reconnaît $\mathfrak{L}(E)$. ■

Voici la même expression régulière que dans le précédent exemple (construction de Glushkov), avec la construction d'Antimirov : $(a + b)(a^* + ba^* + b^*)^*$



5 Comparaison des diverses constructions & remarques

Aucun des automates considérés n'est déterministe. Il faut donc simuler la déterminisation en gérant une liste d'états atteints.

La simulation d'un automate déterminisé est variable en complexité, et dépend de son nombres d'états.

5.1 Thompson vs. Glushkov vs. Antimirov

On note par m la longueur de l'expression régulière étudiée et n le nombre de symboles de l'alphabet dans l'expression E.

On résume dans un petit tableau les résultats des différentes méthodes :

Algorithme	Complexité en temps
Thomson	m
Glushkov	$O(mn)$
Antimirov	$O(m\log(m)+mn)$

Cependant, bien que les constructions de Glushkov et d'Antimirov soient plus longues à calculer, il faut prendre en compte que pour l'algorithme de Thompson, il faut ensuite faire la fermeture des ε-transitions.

En conclusion, bien que l'algorithme d'Antimirov soit plus long, on obtient un automate sans ε-transition et presque minimal (cf. Exemple)

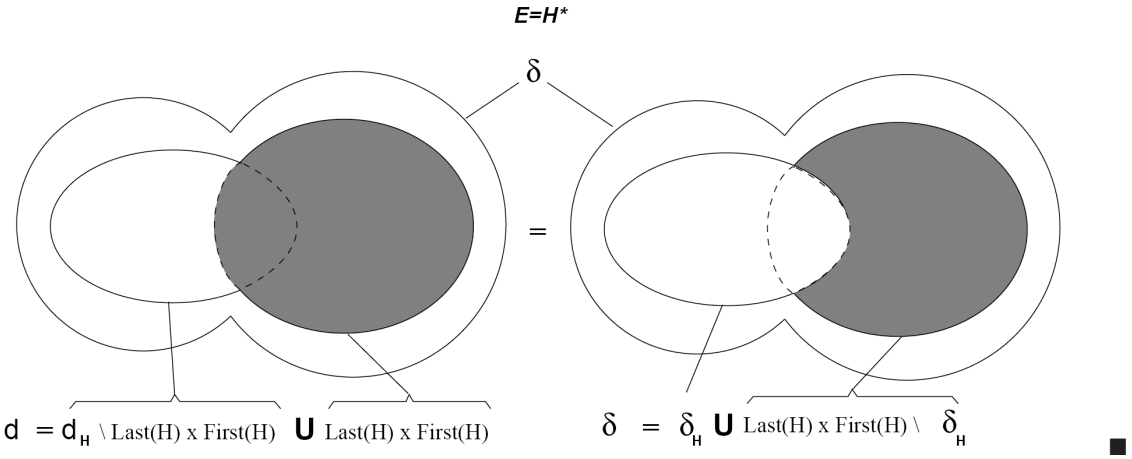
5.2 Comparaison de la méthode de Chang-Paige et Brüggemann-Klein

Les deux méthodes d'optimisation permettent de ne considérer que des unions disjointes, ce qui est plus rapide au niveau de l'implémentation (recherche de double en moins).

Pour pouvoir comparer les approches de Chang et Paige et de A. Brüggemann-Klein , nous désignerons 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.

Proposition 5.1. *On a $d = \delta$ pour toute expression régulière E.*

Preuve La preuve se fait par induction sur E.
Si $E = 0, 1, a \in \Sigma, E_g|E_d$ ou $E_g \cdot E_d$, c'est la même construction.
Dans le cas il l'on a $E = H^*$, on voit que d et δ correspondent aux deux décompositions possibles d'une union en réunion disjointe (cf section 3.3 et 3.4) comme illustré par le diagramme suivant.



■

5.3 Quelques propriétés intéressantes

Dans cette section, on énoncera sans démonstration, pour la curiosité du lecteur, les liens entre les automates produits par les diverses construction.

Pour les démonstrations, se référer à la bibliographie.

Définition 5.1. Soit E une expression régulière. On note

- $A_T(E), A_G(E)$ et $A_A(E)$ les automates produits par les constructions de Thomson, Glushkov et Antimirov.

- $Close_\varepsilon(A)$ la fermeture par ε-transition de A
- $\overline{T}$ (où T est un automate avec ε-transitions) l'automate dont certaines ε-transitions sont marquées, pour le rendre déterministe. C'est
- $\overline{dema}(\overline{T})$ est l'opération inverse : enlever le marquage des ε-transitions de $\overline{T}$.
- $min(A)$, la minimisation de A

Proposition 5.2. *Soit E une expression régulière, on a : $A_G(E) = Close_\varepsilon(A_T(E))$.*

Proposition 5.3. *Soit E une expression régulière, on a : $A_A(E) = \overline{dema}(min(Close_\varepsilon(\overline{A_T(E)})))$.*

Les égalités des algorithmes ci-dessus montrent une manière simple de construire $A_A(E)$ et $A_G(E)$. Cependant, l'intérêt est de construire ces automates de façon plus efficace.

6 Conclusion

On a effectué un tour des méthodes de construction d'automates à partir d'une expression régulière. On a vu que l'on pouvait toujours partir de l'automate de Thompson, de le déterminiser et puis le minimiser.

Cependant, ce n'est pas la méthode la plus efficace. On a donc vu que l'algorithme de Glushkov permet déjà de construire un automate plus petit que celui de Thompson. Puis la construction d'Antimirov permet d'avoir encore un automate beaucoup plus petit. En effet, cet algorithme est équivalent à une minimisation sur une expression régulière légèrement transformée.

Cependant , la théorie des expressions régulières est encore en plein développement, et de nombreuses optimisations sont implémentées améliorant ainsi la rapidité de transformation de l'expression.

Je remercie Tony Ly pour sa première relecture, Boris Yakobowski pour sa patience, sa correction de mon LaTeX, de mon orthographe, et en particulier Rémy Oudompheng pour le LaTeX, la reformulation de mes phrases, et ses relectures constantes.

7 Bibliographie

Une petite bibliographie des liens et livres utilisés :

- J. Sakarovitch, *Éléments de théorie des automates*. Vuibert, 2003.
- G. Beauquier, J. Berstel, and P. Chrétienne, *Éléments d'algorithmique*. Masson, 1993.
- D. Perrin, *Finite automata*, in Handbook of Theoretical Computer Science (J. van Leeuwen, ed.), vol. B, chapt. 1, pp. 1-57, Elsevier, 1990.
- D. Ziadi, J.-L. Ponty et J.-M. Champarnaud, *Passage d'une expression rationnelle à un automate fini non-déterministe*, Bulletin of the Belgian Mathematical Society Simon Stevin, 4-2(1997), pp. 177–203
- D. Ziadi and J.-M. Champarnaud, *An optimal parallel algorithm to convert a regular expression into its Glushkov automaton*, Theoret. Comp. Sc., 215(1999), pp. 69–87
- J.-M. Champarnaud, J.-L. Ponty and D. Ziadi, *From Regular Expressions to Finite Automata*, Intern. J. Computer Math., 72 (1999), pp. 415–431
- J.-M. Champarnaud and D. Ziadi, *Canonical Derivatives, Partial Derivatives and Finite Automaton Constructions*, Theoret. Comp. Sc., 289(2002), pp. 137–163
- J.-M. Champarnaud and G. Duchamp, *Derivatives of rational expressions and related theorems*, Theoret. Comp. Sc., 313-1(2004), pp. 31–44
- J.-M. Champarnaud, F. Coulon and T. Paranthoën, *Compact and Fast Algorithms for Safe Regular Expression Search*, Intern. Journal Computer Math., 81-4(2004), pp. 383–401