

DES EXPRESSIONS RÉGULIÈRES AUX AUTOMATES

À la fin de ce chapitre, je sais :

- ✎ expliquer le théorème de Kleene et ses conséquences
- ✎ transformer une expression régulière en automate
- ✎ montrer qu'un langage est local
- ✎ appliquer l'algorithme de Thompson et de Berry-Sethi
- ✎ décrire l'automate de Glushkov

A Théorème de Kleene

Les chapitres précédents ont permis de construire deux ensembles de langages :

1. l'ensemble des langages réguliers, c'est-à-dire dénotés par une expression régulière,
2. et l'ensemble des langages reconnaissables, c'est-à-dire reconnus par un automate fini.

Il s'agit maintenant d'établir une correspondance entre ces deux ensembles de langages.

Théorème 1 — Kleene. Un langage \mathcal{L} sur un alphabet Σ est un langage régulier si et seulement s'il est reconnaissable.

Démonstration. (\Rightarrow) Soit \mathcal{L} un langage régulier. On utilise la définition inductive des langages réguliers pour montrer que ce langage est reconnaissable.

(Cas de base) • l'ensemble vide est reconnu par un automate dont l'ensemble des états accepteurs F est vide. $\mathcal{L}_{ER}(\emptyset)$ est un langage reconnaissable.

- le mot vide est reconnu par un automate à un seul état dont l'état initial est accepteur. $\mathcal{L}_{ER}(\epsilon)$ est un langage reconnaissable.
- les lettres de l'alphabet sont des langages reconnaissables de la même manière.

(Pas d'induction) • (union) : soient \mathcal{L}_1 et \mathcal{L}_2 deux langages réguliers reconnus par deux automates \mathcal{A}_1 et \mathcal{A}_2 . Alors on a :

$$\begin{aligned}\mathcal{L}_1 \cup \mathcal{L}_2 &= \{w, w \in \mathcal{L}_1 \text{ ou } w \in \mathcal{L}_2\} \\ &= \{w, w \in \mathcal{L}_{rec}(\mathcal{A}_1) \text{ ou } w \in \mathcal{L}_{rec}(\mathcal{A}_2)\} \\ &= \mathcal{L}_{rec}(\mathcal{A}_1) \cup \mathcal{L}_{rec}(\mathcal{A}_2)\end{aligned}$$

D'après la loi de Morgan, on a $\mathcal{L}_{rec}(\mathcal{A}_1) \cup \mathcal{L}_{rec}(\mathcal{A}_2) = \overline{\overline{\mathcal{L}_{rec}(\mathcal{A}_1)} \cap \overline{\mathcal{L}_{rec}(\mathcal{A}_2)}}$. Or, les langages reconnaissables sont stables par intersection et passage au complémentaire. Ils sont donc stable pour l'union et $\mathcal{L}_{rec}(\mathcal{A}_1) \cup \mathcal{L}_{rec}(\mathcal{A}_2)$ est donc un langage reconnaissable.

- (concéténation) : soient \mathcal{L}_1 et \mathcal{L}_2 deux langages réguliers reconnus par deux automates \mathcal{A}_1 et \mathcal{A}_2 . Alors construisons l'automate \mathcal{A} en reliant les états accepteurs de \mathcal{A}_1 à l'état initial de \mathcal{A}_2 par une transition spontanée (cf. figure 3). Cet automate \mathcal{A} reconnaît alors le langage $\mathcal{L}_1.\mathcal{L}_2$.
- (fermeture de Kleene) : soit \mathcal{L} un langage régulier reconnu par un automate \mathcal{A} . Construisons l'automate \mathcal{A} associé comme sur la figure 4. Alors \mathcal{A} reconnaît le langage \mathcal{L}^* .

(Conclusion) Un langage régulier \mathcal{L} est un langage reconnaissable. Pour ce sens de la démonstration, on s'est appuyé sur l'algorithme de Thompson. Mais on peut aussi utiliser l'automate de Glushkov et l'algorithme de Berry-Sethi.

(\Leftarrow) L'algorithme de Mac Naughton-Yamada permet de calculer l'expression régulière associée à un automate fini (\rightarrow HORS PROGRAMME). On le montrera via l'élimination des états au prochain chapitre. ■

(R) Le théorème de Kleene permet d'affirmer que les langages réguliers sont stables par intersection, complémentation ce qui n'était pas évident d'après la définition des expressions régulières. Inversement, les langages reconnaissables sont stables par union, concaténation et passage à l'étoile de Kleene. Tout résultat sur un type de langage (reconnaisable ou régulier) peut se transposer à l'autre type grâce au théorème de Kleene.

Les sections qui suivent présentent les algorithmes au programme qui permettent de passer du formalisme d'une expression régulière à celui d'un automate.

B Algorithme de Thompson

L'algorithme de Thompson permet de construire un automate reconnaissant le langage dénoté par une expression régulière en utilisant des patrons de conception d'automate normalisé pour chaque cas de base et chaque opération (union, concaténation, étoile de Kleene). Cet algorithme porte également le nom de méthode compositionnelle car on compose des automates correspondants à des expressions simples.

a Patron de conception d'un cas de base

b Patron de conception de l'union

On associe à l'union de deux expressions régulières $e_1|e_2$ l'automate décrit sur la figure 2. Au démarrage de la procédure, les deux expressions possèdent un automate équivalent comme le montre la figure 1.


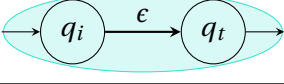
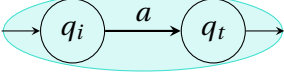
Expression régulière	Automate associé
\emptyset	
ϵ	
$a \in \Sigma$	

TABLE 1 – Automates associés aux cas de bases des expressions régulières.

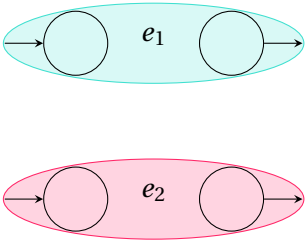


FIGURE 1 – Automates équivalents à e_1 et e_2 avant l'opération

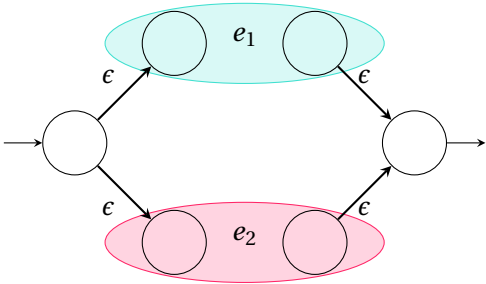
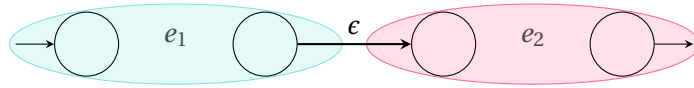


FIGURE 2 – Automate associé à l'union de deux expressions régulières $e_1|e_2$

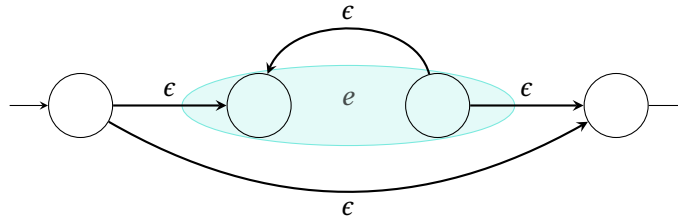
FIGURE 3 – Automate associé à la concaténation de deux expressions régulières $e_1 e_2$

c Patron de conception de la concaténation

On associe à la concaténation de deux expressions régulières l'automate décrit sur la figure 3.

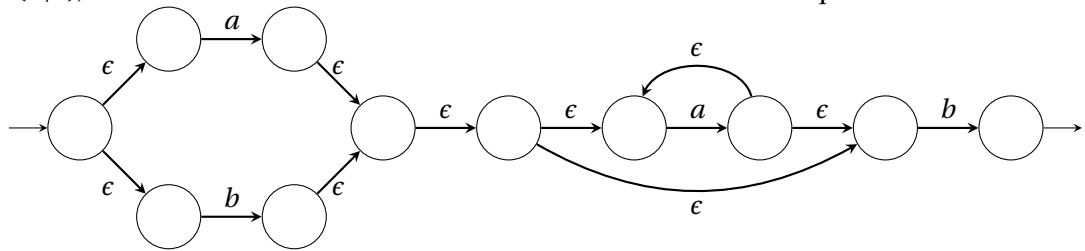
d Patron de conception de l'étoile de Kleene

On associe à la fermeture de Kleene d'une expression régulière l'automate décrit sur la figure 4.

FIGURE 4 – Automate associé à la fermeture de Kleene de e

e Application

■ **Exemple 1** — $(a|b)a^*b$. On décompose l'expression régulière en éléments simples concaténés $(a|b)$, a^* et b et on enchaîne les automates associés. L'automate équivalent est :



f Élimination des transitions spontanées

L'automate de la figure 1 comporte un certain nombre de transitions spontanées. Si, parfois, ces transitions permettent de rendre plus lisible l'automate, celles-ci multiplient cependant les états ce qui n'est pas souhaitable, surtout dans l'optique de programmer cet automate

en le déterminisant... Il faut donc trouver une méthode pour éliminer ces transitions spontanées.

(R) Même s'il est possible d'éliminer les transitions spontanées une fois qu'on a construit tout l'automate associé à une expression régulière, il est souvent souhaitable de le faire à la volée afin de ne pas aboutir à un automate illisible.

(M) Méthode 1 — Élimination des transitions spontanées Il existe deux procédures similaires, une par l'avant, une par l'arrière.

1. Fermeture par l'avant :

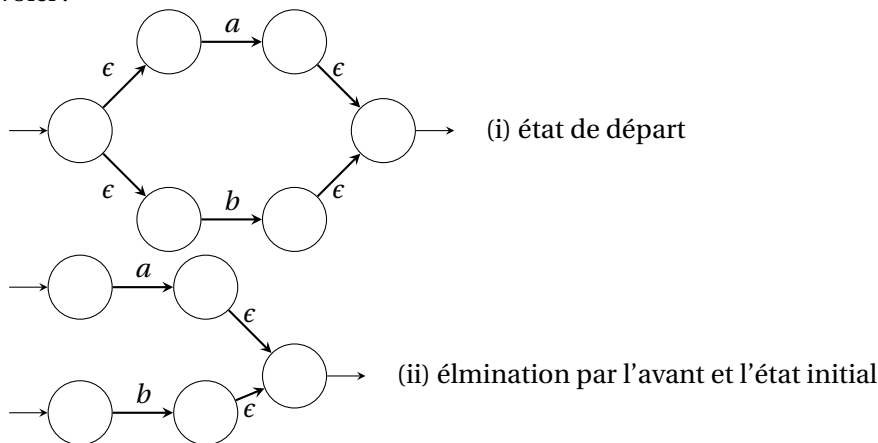
- $(p \xrightarrow{a} q \xrightarrow{\epsilon} r) \rightsquigarrow (p \xrightarrow{a} r \text{ et } p \xrightarrow{a} q)$ Pour chaque transition d'un état p à un état q portant une lettre a et pour chaque transition spontanée de q à un état r , ajouter une transition de p à r portant la lettre a . Éliminer la transition spontanée.
- $(\rightarrow p \xrightarrow{\epsilon} q) \rightsquigarrow q \in Q_{init}$ Pour chaque transition spontanée d'un état p initial à un état q , ajouter q à l'ensemble des états initiaux. Éliminer la transition spontanée.

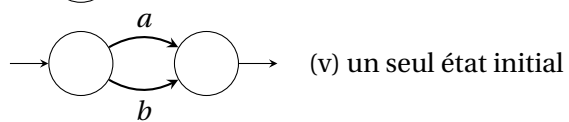
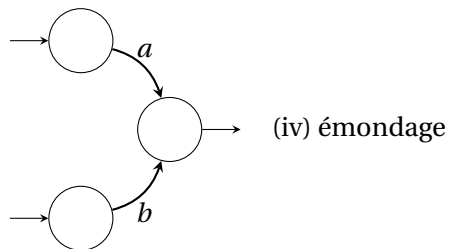
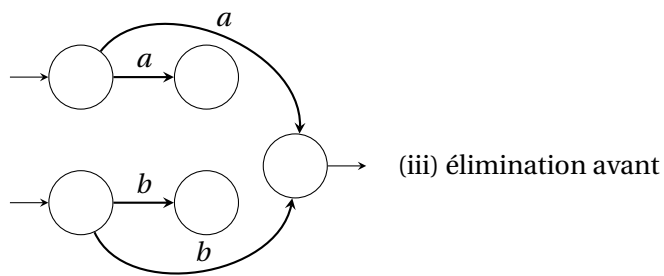
2. Fermeture par l'arrière :

- $(p \xrightarrow{\epsilon} q \xrightarrow{a} r) \rightsquigarrow (p \xrightarrow{a} r \text{ et } q \xrightarrow{a} r)$ Pour chaque transition spontanée d'un état p à un état q et pour chaque transition de q à un état r portant la lettre a , ajouter une transition de p à r portant la lettre a . Éliminer la transition spontanée.
- $(p \xrightarrow{\epsilon} q \rightarrow) \rightsquigarrow p \in F$ Pour chaque transition spontanée d'un état p à un état accepteur q , ajouter p à l'ensemble des états accepteurs. Éliminer la transition spontanée.

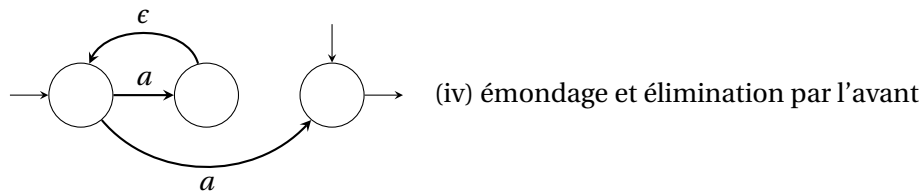
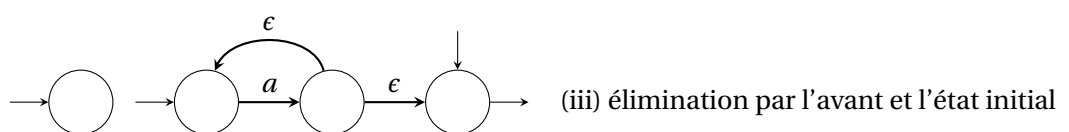
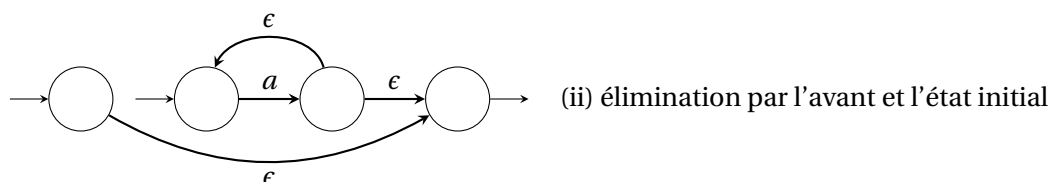
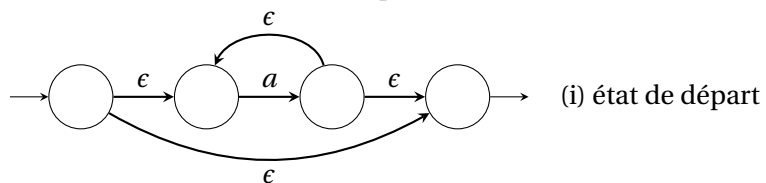
■ **Exemple 2 — Élimination des transitions spontanées de l'automate de l'exemple 1.** Pour chaque automate associé, on peut déjà appliquer la fermeture avant ou arrière.

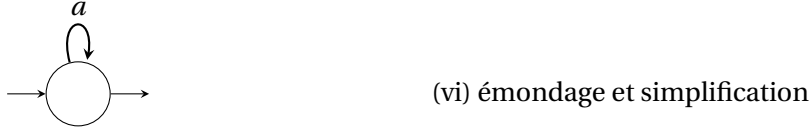
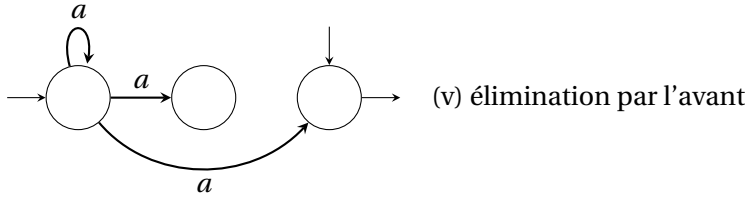
Pour l'union $a|b$, on trouve l'automate sans transitions spontanées en plusieurs étapes que voici :



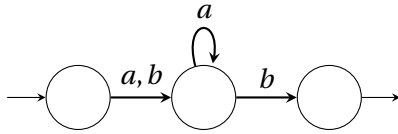


Pour la fermeture de Kleene, on procède de la même manière :





Finalement, on peut maintenant représenter l'automate normalisé (et déterministe) après élimination des transitions spontanées :



C Algorithme de Berry-Sethi et automate de Glushkov

Les notions de langage local et d'expression régulière linéaire sont introduites dans la seule perspective de construire l'automate de Glushkov[glushkov_abstract_1961] associé à une expression régulière linéaire par l'algorithme de Berry-Sethi[berry_regular_1986], conformément au programme. C'est un long développement pour une procédure finale relativement simple.

a Langages locaux

■ **Définition 1 — Ensembles** . Soit \mathcal{L} un langage sur Σ . On définit quatre ensembles de la manière suivante :

- les premières lettres des mots de \mathcal{L}

$$P(\mathcal{L}) = \{a \in \Sigma, \exists w \in \Sigma^*, aw \in \mathcal{L}\} \quad (1)$$

- les dernières lettres des mots de \mathcal{L} :

$$S(\mathcal{L}) = \{a \in \Sigma, \exists w \in \Sigma^*, wa \in \mathcal{L}\} \quad (2)$$

- les facteurs de longueur 2 des mots de \mathcal{L} :

$$F(\mathcal{L}) = \{v \in \Sigma^*, |v| = 2, \exists u, w \in \Sigma^*, uvw \in \mathcal{L}\} \quad (3)$$

- les facteurs de longueur 2 impossibles :

$$N(L) = \Sigma^2 \setminus F(L) \quad (4)$$

■ **Définition 2 — Langage local.** Un langage \mathcal{L} sur Σ est local s'il existe deux parties P et S de Σ et une partie N de Σ^2 tels que :

$$\mathcal{L} \setminus \{\epsilon\} = (P\Sigma^* \cap \Sigma^* S) \setminus (\Sigma^* N\Sigma^*) \quad (5)$$

Dans ce cas, on a nécessairement $P = P(\mathcal{L})$, $S = S(\mathcal{L})$, $N = N(\mathcal{L})$.

(R) Cette définition signifie que l'appartenance d'un mot à un langage local peut être établie uniquement en regardant la première lettre, la dernière lettre et tous les blocs de deux lettres de ce mot. On peut imaginer que, pour vérifier, on fait glisser pour comparer tous les blocs de deux lettres non autorisés (N) sur le mot. D'où le nom local : on n'a pas besoin d'examiner dans sa globalité le mot, mais uniquement chaque lettre et sa voisine. P , S et N suffisent donc pour définir un langage local.

■ **Exemple 3 — Langages locaux .** Sur l'alphabet $\Sigma = \{a, b\}$, on peut déterminer pour chacun des langages suivants les langages P , S et N et dire s'ils sont locaux :

1. $\mathcal{L}_{ER}(a^*) : P = \{a\}$, $S = \{a\}$ et $N = \{ab, ba, bb\}$.
2. $\mathcal{L}_{ER}((ab)^*) : P = \{a\}$, $S = \{b\}$ et $N = \{aa, bb\}$.

■ **Exemple 4 — Langages non locaux.** Sur l'alphabet $\Sigma = \{a, b\}$, on peut déterminer pour chacun des langages suivants les langages P , S et N et trouver un contre-exemple pour montrer qu'ils ne sont pas locaux :

1. $\mathcal{L}_{ER}(a^*(ab)^*) : P = \{a\}$, $S = \{a, b\}$ et $N = \{bb\}$. Mais on observe que le mot aba est dans $(P\Sigma^* \cap \Sigma^* S) \setminus (\Sigma^* N\Sigma^*)$ mais pas dans \mathcal{L} .
2. $\mathcal{L}_{ER}(a^*|(ab)^*) : \text{idem}$

(R) Le dernier exemple montre que les langages locaux ne sont pas stables par union et concaténation.

Théorème 2 — L'ensemble des langages locaux est stable par intersection.

Démonstration. Soient \mathcal{L}_1 et \mathcal{L}_2 deux langages locaux, et $\mathcal{L} = \mathcal{L}_1 \cap \mathcal{L}_2$. On pose $P = P(\mathcal{L})$, $S = S(\mathcal{L})$ et $N = N(\mathcal{L})$, les ensembles définis comme en 1. Alors on a :

$$\begin{aligned} \mathcal{L} \setminus \{\epsilon\} &= (\mathcal{L}_1 \setminus \{\epsilon\}) \cap (\mathcal{L}_2 \setminus \{\epsilon\}) \\ &= ((P(\mathcal{L}_1)\Sigma^* \cap \Sigma^* S(\mathcal{L}_1)) \setminus (\Sigma^* N(\mathcal{L}_1)\Sigma^*)) \cap ((P(\mathcal{L}_2)\Sigma^* \cap \Sigma^* S(\mathcal{L}_2)) \setminus (\Sigma^* N(\mathcal{L}_2)\Sigma^*)) \\ &= (P(\mathcal{L}_1)\Sigma^* \cap \Sigma^* S(\mathcal{L}_1) \cap P(\mathcal{L}_2)\Sigma^* \cap \Sigma^* S(\mathcal{L}_2)) \setminus (\Sigma^* N(\mathcal{L}_1)\Sigma^* \cup \Sigma^* N(\mathcal{L}_2)\Sigma^*) \\ &= (P\Sigma^* \cap \Sigma^* S) \setminus (\Sigma^* N\Sigma^*). \end{aligned}$$

Donc \mathcal{L} est bien local. ■

Théorème 3 — L'union de deux langages locaux définis sur deux alphabets disjoints est un langage local.

Démonstration. Soit Σ_1 et Σ_2 les alphabets **disjoints** sur lesquels sont définis \mathcal{L}_1 et \mathcal{L}_2 , deux langages locaux et $\mathcal{L} = \mathcal{L}_1 \cup \mathcal{L}_2$ défini sur $\Sigma = \Sigma_1 \cup \Sigma_2$. On pose $P = P(\mathcal{L})$, $S = S(\mathcal{L})$ et $N = N(\mathcal{L})$, les ensembles définis comme en 1.

On doit montrer l'inclusion $(P(\mathcal{L})\Sigma^* \cap \Sigma^* S(\mathcal{L})) \setminus (\Sigma^* N(\mathcal{L})\Sigma^*) \subset \mathcal{L}$ puisque l'inclusion réciproque est toujours vraie.

Considérons donc un mot $w \in (P(\mathcal{L})\Sigma^* \cap \Sigma^* S(\mathcal{L})) \setminus (\Sigma^* N(\mathcal{L})\Sigma^*)$ que l'on décompose en lettres $w = a_1 \dots a_n$. Montrons que $w \in \mathcal{L}$

- Soit $a_1 \in P(\mathcal{L}) = P(\mathcal{L}_1) \cup P(\mathcal{L}_2)$, on peut supposer sans perte de généralité que $a_1 \in P(\mathcal{L}_1)$, alors $a_1 \in \Sigma_1$.
- $a_1, a_2 \in \Sigma^2 \setminus N(\mathcal{L}) = F(\mathcal{L}_1) \cup F(\mathcal{L}_2)$, or $a_1 \in \Sigma_1$ et les alphabets Σ_1 et Σ_2 sont disjoints, donc nécessairement $a_1 a_2 \in F(\mathcal{L}_1)$ et $a_2 \in \Sigma_1$.
- De proche en proche, on montre que $a_i a_{i+1} \in F(\mathcal{L}_1)$ et $a_i \in \Sigma_1$ pour tout i .
- Enfin, $a_n \in S(\mathcal{L}) = S(\mathcal{L}_1) \cup S(\mathcal{L}_2)$ et $a_n \in \Sigma_1$ donc $a_n \in S(\mathcal{L}_1)$.
- Finalement $w \in (P(\mathcal{L}_1)\Sigma^* \cap \Sigma^* S(\mathcal{L}_1)) \setminus (\Sigma^* N(\mathcal{L}_1)\Sigma^*) = \mathcal{L}_1$ car \mathcal{L}_1 est local.

En partant de l'hypothèse que $a_1 \in \Sigma_2$, on en aurait conclu que $w \in \mathcal{L}_2$. On en déduit que $w \in \mathcal{L}_1 \cup \mathcal{L}_2$ et, donc, $\mathcal{L} = \mathcal{L}_1 \cup \mathcal{L}_2$ est un langage local. ■

Théorème 4 — La concaténation de deux langages locaux définis sur deux alphabets disjoints est un langage local.

Démonstration. Soit Σ_1 et Σ_2 les alphabets **disjoints** sur lesquels sont définis \mathcal{L}_1 et \mathcal{L}_2 , deux langages locaux et $\mathcal{L} = \mathcal{L}_1 \mathcal{L}_2$ défini sur $\Sigma = \Sigma_1 \Sigma_2$. On pose $P = P(\mathcal{L})$, $S = S(\mathcal{L})$ et $N = N(\mathcal{L})$, les ensembles définis comme en 1.

On doit montrer l'inclusion $(P(\mathcal{L})\Sigma^* \cap \Sigma^* S(\mathcal{L})) \setminus (\Sigma^* N(\mathcal{L})\Sigma^*) \subset \mathcal{L}$ puisque l'inclusion réciproque est toujours vraie.

$$P(\mathcal{L}_1 \mathcal{L}_2) = \begin{cases} \{a \in \Sigma_2, \exists w \in \Sigma_2^*, aw \in \mathcal{L}_2\} & \text{si } \epsilon \in \mathcal{L}_1 \\ \{a \in \Sigma_1, \exists w \in \Sigma_1^* \Sigma_2^*, aw \in \mathcal{L}_1 \mathcal{L}_2\} & \text{sinon} \end{cases}$$

$$S(\mathcal{L}_1 \mathcal{L}_2) = \begin{cases} \{a \in \Sigma_1, \exists w \in \Sigma_1^*, wa \in \mathcal{L}_1\} & \text{si } \epsilon \in \mathcal{L}_2^* \\ \{a \in \Sigma_2, \exists w \in \Sigma_1^* \Sigma_2^*, wa \in \mathcal{L}_1 \mathcal{L}_2\} & \text{sinon} \end{cases}$$

$$F(\mathcal{L}_1 \mathcal{L}_2) = \{v \in \Sigma_1^* \Sigma_2^*, |v| = 2, \exists u, w \in \Sigma_1^* \Sigma_2^*, uvw \in \mathcal{L}_1 \mathcal{L}_2\}$$

On considère un mot $w \in (P(\mathcal{L})\Sigma^* \cap \Sigma^* S(\mathcal{L})) \setminus (\Sigma^* N(\mathcal{L})\Sigma^*)$ et on le décompose en lettres $w = a_0 \dots a_n$. On traite différents cas :

- Si $a_0 \in \Sigma_2$, alors $\epsilon \in \mathcal{L}_1$ et $a_2 \in P(\mathcal{L}_2)$. De proche en proche on montre que $a_i a_{i+1} \in F(\mathcal{L}_2)$, $a_i \in \Sigma_2$ pour tout i et $a_n \in S(\mathcal{L}_2)$, donc $w \in \mathcal{L}_2$ car \mathcal{L}_2 est local.

- Si $a_0 \in \Sigma_1$, alors $a_0 \in P(\mathcal{L}_1)$. Notons $a_0 \dots a_k$ le plus long préfixe de w qui soit dans Σ_1^* . On montre de proche en proche que $a_i a_{i+1} \in F(\mathcal{L}_1)$ pour $i < k$. Puis deux cas se présentent :
 - Si $k = n$ alors $a_n \in S(\mathcal{L}_1)$, donc $w \in \mathcal{L}_1$ car \mathcal{L}_1 est local.
 - Si $k < n$, on a $a_k a_{k+1} \in S(\mathcal{L}_1)P(\mathcal{L}_2)$ donc $a_k \in S(\mathcal{L}_1)$ et $a_{k+1} \in P(\mathcal{L}_2)$. On prouve alors que $a_0 \dots a_k \in \mathcal{L}_1$ et $a_{k+1} \dots a_n \in \mathcal{L}_2$ avec les mêmes arguments.
- Finalement, $w \in \mathcal{L}_1 \mathcal{L}_2$ et \mathcal{L} est local. ■

Théorème 5 — La fermeture de Kleene d'un langage local est un langage local.

Démonstration. On pose $P = P(\mathcal{L})$, $S = S(\mathcal{L})$ et $N = N(\mathcal{L})$, les ensembles définis comme en 1. On a :

$$P(\mathcal{L}^*) = \{a \in \Sigma, \exists w \in \Sigma^*, aw \in \mathcal{L}^*\}$$

$$S(\mathcal{L}^*) = \{a \in \Sigma, \exists w \in \Sigma^*, wa \in \mathcal{L}^*\}$$

$$F(\mathcal{L}^*) = \{v \in \Sigma^*, |v| = 2, \exists u, w \in \Sigma^*, uvw \in \mathcal{L}^*\}$$

On considère un mot $u = a_0 \dots a_n \in (P(\mathcal{L}^*)\Sigma^* \cap \Sigma^*S(\mathcal{L}^*)) \setminus (\Sigma^*N(\mathcal{L}^*)\Sigma^*)$ et on cherche à montrer qu'il appartient à \mathcal{L}^* .

De la même manière que précédemment, si $a_0 \in \Sigma$ alors $a_0 \in P(\mathcal{L})$. Les facteurs de longueur 2 de w sont dans $F(\mathcal{L})$ et $a_n \in \mathcal{L}$. On en déduit une décomposition de w en mots dans $(P(\mathcal{L})\Sigma^* \cap \Sigma^*S(\mathcal{L})) \setminus (\Sigma^*N(\mathcal{L})\Sigma^*)$ donc dans \mathcal{L} car \mathcal{L} est local. Finalement, $w \in \mathcal{L}^*$, car qui peut le plus peut le moins et \mathcal{L}^* est local. ■

b Expressions régulières linéaires

Les langages définis par des expressions régulières ne sont donc pas toujours locaux. En revanche les langages définis par une expression régulière linéaire le sont.

■ **Définition 3 — Expression régulière linéaire.** Une expression régulière e sur Σ est linéaire si toute lettre de Σ apparaît **au plus une fois** dans e .

■ **Exemple 5 — Expression régulière linéaire.** L'expression $(ab)^*$ est linéaire mais pas $(ab)^*a^*$.

Théorème 6 — Toute expression régulière linéaire dénote un langage local.

Démonstration. On procède par induction structurelle et en utilisant les propriétés des langages locaux.

Cas de base : \emptyset , ϵ et $a \in \Sigma$ sont des expressions régulières linéaires.

- $\mathcal{L}_{ER}(\emptyset) = \emptyset$ et on a bien $(P\Sigma^* \cap \Sigma^*S) \setminus (\Sigma^*N\Sigma^*) = \emptyset \setminus (\Sigma^*\Sigma^2\Sigma^*) = \emptyset$
- $\mathcal{L}_{ER}(\epsilon) = \{\epsilon\}$ et on a bien $(P\Sigma^* \cap \Sigma^*S) \setminus (\Sigma^*N\Sigma^*) = \epsilon \setminus (\Sigma^*\Sigma^2\Sigma^*) = \{\epsilon\}$

- $\mathcal{L}_{ER}(a) = \{a\}$ et on a bien $(P\Sigma^* \cap \Sigma^* S) \setminus (\Sigma^* N\Sigma^*) = \{a\} \setminus (\Sigma^* \Sigma^2 \Sigma^*) = \{a\}$

Pas d'induction :

(union) Soit Σ_1 et Σ_2 des alphabets **disjoints**. Soient e_1 et e_2 deux expressions régulières linéaires définies respectivement sur Σ_1 et Σ_2 . Alors $e_1|e_2$ est régulière et linéaire et la sémantique des expressions régulières permet d'affirmer que $\mathcal{L}_{ER}(e_1|e_2) = \mathcal{L}_{ER}(e_1) \cup \mathcal{L}_{ER}(e_2)$. Or, l'union de deux langages locaux dont les alphabets sont disjoints est un langage local. Donc $\mathcal{L}_{ER}(e_1|e_2)$ est un langage local.

(concaténation) on procède de même en utilisant la concaténation de deux langages locaux.

(union) on procède de même en utilisant la fermeture de Kleene de deux langages locaux.

Finalement, les expressions régulières linéaires dénotent des langages locaux. ■

(R) La réciproque de ce théorème est fausse : par exemple, le langage $L(aa^*)$ est local mais aa^* n'est pas une expression régulière linéaire.

c Automate locaux

Automates locaux

■ **Définition 4 — Automate local.** Un automate fini déterministe $\mathcal{A} = (Q, \Sigma, q_i, \delta, F)$ est local si pour toute lettre $a \in \Sigma$, il existe un état $q \in Q$ tel que toutes les transitions étiquetées par a arrivent dans q .

Théorème 7 — Tout langage local \mathcal{L} est reconnaissable par un automate local. De plus, si \mathcal{L} ne contient pas le mot vide ϵ , alors l'automate est normalisé.

Démonstration. Soit \mathcal{L} un langage local. On pose $P = P(\mathcal{L})$, $S = S(\mathcal{L})$ et $N = N(\mathcal{L})$, les ensembles définis comme en 1.

On considère l'automate $\mathcal{A} = (Q, \Sigma \cup \{\epsilon\}, q_0, \delta, S \cup \{\epsilon\})$ et la fonction δ définie par :

$$\forall a \in P, \delta(q_0, a) = q_a \quad (6)$$

$$\forall a_1 a_2 \in F, \delta(q_{a_1}, a_2) = q_{a_2} \quad (7)$$

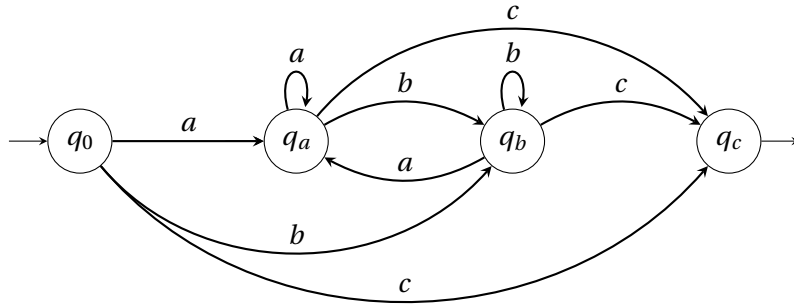
$$(8)$$

Par construction de cet automate, un mot $w = a_0 a_1 \dots a_n$ est reconnu si et seulement si :

- $a_0 \in P$,
- $\forall i \in \llbracket 0, n-1 \rrbracket, a_i a_{i+1} \in F$
- et $a_n \in S$.

Comme \mathcal{L} est local, on a bien $\mathcal{L}_{rec}(\mathcal{A}) = \mathcal{L}$. Si \mathcal{L} ne contient pas le mot vide, il suffit de l'exclure des états accepteurs et on obtient un automate normalisé. ■

■ **Exemple 6 — Automate associé à l'expression régulière linéaire $(a|b)^*c$.** Le langage dénoté par cette expression régulière est local (le montrer!). On construit l'automate défini lors de la démonstration du théorème 7. Comme ce langage ne comporte pas le mot vide, l'automate est normalisé.



d Automate de Glushkov et algorithme de Berry-Sethi

On cherche maintenant un algorithme pour transformer une expression régulière (pas nécessairement linéaire) en un automate fini.

M **Méthode 2 — Algorithme de Berry-Sethi** Pour obtenir un automate fini local reconnaissant le langage $\mathcal{L}_{ER}(e)$ à partir d'une expressions régulière e sur un alphabet Σ :

1. Linéariser l'expression e : cela consiste à numérototer toutes les lettres qui apparaissent afin de créer une expression rationnelle linéaire e' .
2. Déterminer les ensembles P , S et F associés au langage local $\mathcal{L}(e')$.
3. Déterminer un automate **local** \mathcal{A} reconnaissant $\mathcal{L}_{ER}(e')$ à partir de P, S et F . On peut associer ses états aux lettres de l'alphabet de e' . L'état initial est relatif au mot vide : s'il appartient au langage, on fait de cet état un état accepteur. Toutes les transitions qui partent de l'état initial conduisent à un état associé à une lettre de P . Les états accepteurs sont associés aux éléments de S . Les facteurs de deux lettres déterminent les autres transitions.
4. Supprimer les numéros sur les transitions et faire réapparaître l'alphabet initial Σ .

L'automate obtenu est nommé automate de Glushkov[glushkov_abstract_1961]. C'est un automate **local et sans transitions spontanées**. Il n'est pas nécessairement déterministe mais on peut le déterminer facilement en utilisant la procédure de déterminisation d'un AFND (cf méthode ??). Il possède $|\Sigma_e|$ états où Σ_e est l'alphabet étendu obtenu en faisant le marquage. $|\Sigma_e|$ correspond donc au nombre total de lettres dans e en comptant les répétitions. Dans le pire des cas, le nombre de transitions de l'automate est en $O(|\Sigma_e|^2)$.

M **Méthode 3 — Linéarisation de l'expression régulière** À partir de l'expression régulière de départ, on numérote à partir de 1 chaque lettre de l'expression dans l'ordre de lecture.

Si une lettre apparaît plusieurs fois, elle est numérotée autant de fois qu'elle apparaît avec un numéro différent.

Par exemple, la linéarisation de $ab|(ac)^*$ est $a_1b_1|(a_2c_1)^*$.

R La linéarisation d'une expression est en fait un marquage par une fonction

$$m : \{a_1, a_2, \dots, b_1, \dots\} \longrightarrow \Sigma \quad (9)$$

que l'on utilise également à la fin de l'algorithme de Berry-Sethi 2 pour supprimer les numéros.

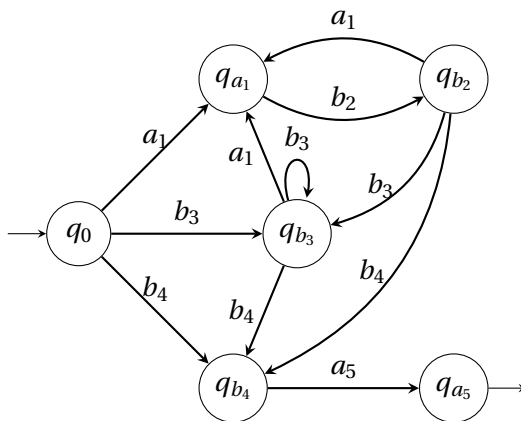
La complexité de l'algorithme de Berry-Sethi est quadratique dans le pire des cas. Si n est le nombre de lettres rencontrées dans l'expression linéarisée, on a :

- la linéarisation est en $O(n)$,
- la construction des ensembles P , S , et F est linéaire en $O(n)$,
- la construction de l'automate avec les ensembles précédents est au pire quadratique en $O(n^2)$ (on construit n chemins qui peuvent être de longueur n),
- la suppression des marquages est linéaire en $O(n)$.

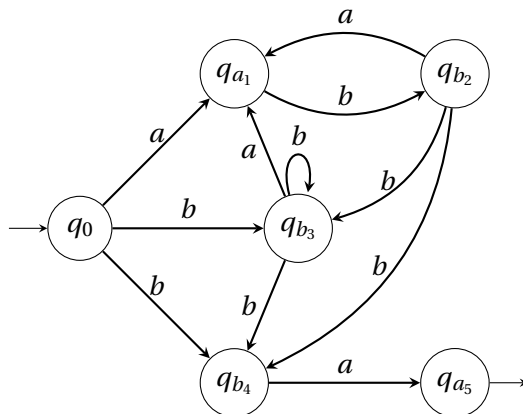
■ **Exemple 7 — Construire l'automate de Glushkov associé à l'expression régulière $(ab|b)^*ba$.**

On applique l'algorithme de Berry-Sethi :

1. Linéarisation : $(ab|b)^*ba \longrightarrow (a_1b_2|b_3)^*b_4a_5$,
2. Construction des ensembles associés au langage local de l'expression linéarisée :
 - $P = \{a_1, b_3, b_4\}$
 - $S = \{a_5\}$
 - $F = \{a_1b_2, b_2a_1, b_2b_4, b_3b_3, b_3b_4, b_3a_1, b_4a_5, b_2b_3\}$
3. Construction de l'automate local associé :



Suppression des marquages des transitions (numéros) :

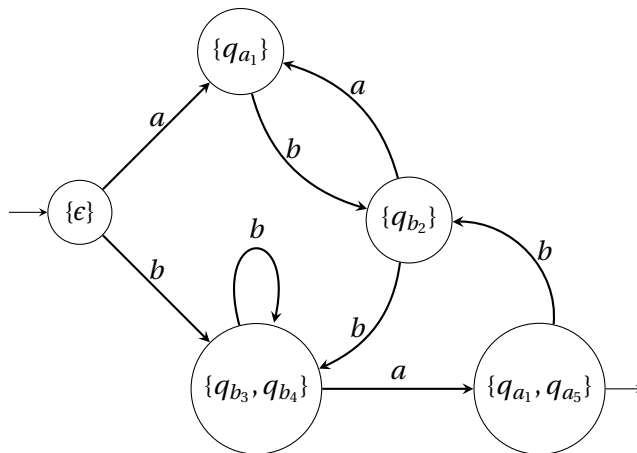


Déterminisation :

On construit la fonction de transition de proche en proche à partir de l'état initial :

	$\downarrow q_0$	$\{q_{a_1}\}$	$\{q_{b_2}\}$	$\{q_{b_3}, q_{b_4}\}$	$\uparrow \{q_{a_1}, q_{a_5}\}$
a	$\{q_{a_1}\}$		$\{q_{a_1}\}$	$\{q_{a_1}, q_{a_5}\}$	
b	$\{q_{b_3}, q_{b_4}\}$	$\{q_{b_2}\}$	$\{q_{b_3}, q_{b_4}\}$	$\{q_{b_3}, q_{b_4}\}$	$\{q_{b_2}\}$

ce qui se traduit par l'AFD :



D Comparaison Thompson / Berry-Sethi

Étape	Thompson	Berry-Sethi
Préparation	Aucune	Linéariser l'expression Construire les ensembles P , S et F
Automate	Construction directe en $O(n)$	Construction en $O(n^2)$
Finition	Supprimer les transitions spontanées	Aucun

TABLE 2 – Comparaison des algorithmes de Thompson et Berry-Sethi.