Automates finis (II)

Montaigne 2023-2024

– mpi23@arrtes.net –

Deux formalismes permettent de définir un langage régulier.

- Par une approche **déclarative**, les **expressions régulières** offrent une syntaxe pour définir un langage sans donner toutefois les moyens de tester l'appartenance d'un mot à ce langage.
- Par une approche **opérationnelle**, les **automates** peuvent être considérés comme des programmes élémentaires permettant de tester l'appartenance d'un mot à un langage.

Théorème de Kleene

Le **théorème de Kleene** établit l'équivalence de ces formalismes : tout langage définissable par une expression régulière est aussi définissable par un automate et réciproquement.

Théorème 1

Soit Σ un alphabet. Les ensembles $Rec(\Sigma)$ et $Reg(\Sigma)$ sont égaux.

La suite de ce chapitre présente une preuve de ce résultat.

De l'expression régulière à l'automate

Langage local

Il existe de nombreux algorithmes pour construire un automate à partir d'une expression régulière. Nous verrons en exercice l'algorithme de Thompson qui permet de définir l'automate inductivement sur la structure de l'expression régulière.

L'algorithme de Berry-Sethi est une autre construction qui convertit une expression régulière en un automate généralement non déterministe sans transition spontanée.

L'automate ainsi construit, appelé **automate de Glushkov** nécessite d'abord la définition d'un type de langage particulier appelé **langage local**.

Langage local

Définition 2 (langage local)

Soit Σ un alphabet et $L \subseteq \Sigma^*$ un langage. On note :

- $P(L) = \{a | a \in \Sigma, \{a\}\Sigma^* \cap L \neq \varnothing\}$
- $S(L) = \{a | a \in \Sigma, \Sigma^*\{a\} \cap L \neq \varnothing\}$
- $F(L) = \{v | v \in \Sigma \times \Sigma, \Sigma^*\{v\}\Sigma^* \cap L \neq \varnothing\}$
- $N(L) = \Sigma \times \Sigma \setminus F(L)$

L est un langage **local** si et seulement si :

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

Langage local

Les ensembles $P(L)$ et $S(L)$ sont respectivement l'ensemble des lettres pouvant apparaître en première et dernière position d'un mot de L.

L'ensemble $F(L)$ est l'ensemble des facteurs de taille 2 des mots de L, c'est-à-dire l'ensemble des couples de lettres qui peuvent apparaître à la suite dans un mot de L.

L'ensemble $N(L)$ est le complémentaire sur $\Sigma \times \Sigma$ de $F(L)$.

Langage local

Un langage est **local** s'il peut s'écrire comme l'ensemble des mots qui commencent par une lettre dans $P(L)$, terminent par une lettre dans $S(L)$ et ne contiennent aucun facteur de $N(L)$.

Cette dernière propriété subtile est une double négation : l'ensemble $N(L)$ est défini comme un complémentaire et apparaît ensuite sous une différence ensembliste. On l'illustre sur quelques exemples.

Exemple

Soit $\Sigma = \{a, b, c\}$ et $L_1 = \mathcal{L}(a^* bc^*)$.

- Les mots de L_1 commencent par a ou b : $P(L_1) = \{a, b\}$.
- Les mots de L_1 finissent par b ou c : $S(L_1) = \{b, c\}$.
- Tout mot de L_1 est de la forme $a \dots abc \dots c$:
  $F(L_1) = \{aa, ab, bc, cc\}$.
- $N(L_1) = \{ac, ba, bb, ca, cb\}$.

Exemple

Si on considère tous les ensembles de mots commençant par a ou b et finissant par b ou c, qu'on en retire tous les mots contenant un facteur dans $N(L_1)$, on obtient bien L_1.

Parcourons les lettres d'un mot :

- si la lettre courante est un a, la suivante peut être un a ou un b (aa et ab sont autorisés), mais pas un c car ac est interdit ;
- si la lettre courante est un b, la suivante peut être un c, mais pas un b car bb est interdit, ni un a car ba est interdit ;
- si la lettre courante est un c, la suivante ne peut être qu'un c car ca et cb sont interdits.

Si on commence par a ou b et qu'on applique les règles ci-dessus jusqu'à produire un b ou un c, on produit bien un mot de L_1.

Donc le langage L_1 est **local**.

Exemple

Soit $\Sigma = \{a, b, c\}$ et $L_2 = \mathcal{L}(ba^*bc^*b)$.

- Les mots commencent par b : $P(L_2) = \{b\}$.
- Les mots finissent par b : $S(L_2) = \{b\}$.
- Tout mot est de la forme $ba \dots abc \dots cb$:
  $F(L_2) = \{ba, aa, ab, bc, cc, cb\}$.
- $N(L_2) = \{ac, bb, ca\}$.

Le mot $bcbab$ ne possède aucun facteur interdit, commence et finit par b, mais n'est pas un mot du langage.

Donc le langage L_2 est **non local**.

Automate local

À un **langage local**, il est possible d'associer un **automate fini déterministe** en considérant uniquement $P(L)$, $S(L)$ et $F(L)$.

Définition 3 (automate local)

Soit Σ un alphabet et $L \subseteq \Sigma^*$ un langage local. On note $\mathrm{Loc}(L)$ l'automate défini par $\mathrm{Loc}(L) = (Q_L, \Sigma, q_0, F_L, \delta_L)$ et :

- $Q_L = \{q_a \mid a \in \Sigma\} \cup \{q_0\}$
- $F_L = \{q_a \mid a \in S(L)\} \cup \{q_0 \mid \varepsilon \in L\}$
- $\delta_L : \quad Q_L \times \Sigma \to Q_L$

$$(q_0, a) \mapsto q_a \qquad \forall a \in P(L)$$

$$(q_a, b) \mapsto q_b \qquad \forall a, \forall b, ab \in F(L)$$

Automate local

Cette construction crée un **état initial** q_0 et **un état pour chaque lettre** de l'alphabet.

Les **états acceptants** sont tous les états correspondant à une lettre de $S(L)$ auxquels on ajoute q_0 si le mot vide fait partie du langage L.

La **fonction de transition** a une forme bien particulière : une transition lisant la lettre a va toujours dans l'état q_a.

Il y a une transition sortant de l'état initial pour chaque lettre de $P(L)$ (on peut commencer le mot par chacune de ces lettres). Il y a une transition sortant d'un état q_a pour toute lettre b telle que ab est dans $F(L)$.

Automate local

Théorème 4 (reconnaissance par un automate local)

Soit Σ un alphabet et $L \subseteq \Sigma^*$ un langage local. Alors :

$$L = \mathcal{L}(Loc(L))$$

Démonstration

$L \subseteq \mathcal{L}(\text{Loc}(L))$ Si $v = a_1 \dots a_n \in L$, alors $a_1 \in P(L)$, $a_n \in S(l)$ et $\forall 1 \leq i < n, a_i a_{i+1} \in F(L)$. Ainsi, $q_0 \xrightarrow{a_1} \dots \xrightarrow{a_n} q_{a_n}$ est un chemin acceptant pour $\text{Loc}(L)$, et donc $v \in \text{Loc}(L)$. Si $v = \varepsilon$, alors $q_0 \in F_L$ et $\varepsilon \in \text{Loc}(L)$.

$\mathcal{L}(\text{Loc}(L)) \subseteq L$ Si $v = a_1 \dots a_n \in \mathcal{L}(\text{Loc}(L))$, il existe un chemin acceptant $q_0 \xrightarrow{a_1} q_{a_1} \dots \xrightarrow{a_n} q_{a_n}$. Par construction de l'automate, $a_1 \in P(L)$ (transition sortant de q_0). De même, $a_n \in S(L)$ (car q_{a_n} est un état acceptant). Et, $\forall 1 \leq i < n, \delta_L(q_{a_i}, a_i) = q_{a_{i+1}}$ et $a_i a_{i+1} \in F(L)$. Donc $v \in L$. Si $v = \varepsilon$ est reconnu par $\text{Loc}(L)$, alors q_0 est forcément acceptant et donc $\epsilon \in L$.

Linéarisation d'une expression régulière

Définition 5 (expression régulière linéaire)

Soit Σ un alphabet. L'expression régulière r sur Σ est dite linéaire si tout symbole de Σ y apparaît au plus une fois.

Les expressions linéaires représentent des **langages locaux**.

On peut linéariser une expression régulière r en associant à chaque occurrence d'une lettre un indice distinct et en traitant les lettres indicées comme des symboles différents.

Par exemple, l'expression $a(ab)^*|b^*a$ peut être linéarisée en $a_1(a_2b_3)^*|b_4^*a_5$. On utilise des indices croissants par convention mais ce n'est pas obligatoire tant que deux occurrences d'un même symbole ont des indices distincts.

Automate de Glushkov

L'**algorithme de Berry-Sethi** construit un NFA à partir d'une expression régulière. Il passe par l'intermédiaire d'un automate local, automate déterministe. Mais une dernière étape, appelée effacement des indices, peut rendre l'automate non déterministe. L'automate obtenu est appelé **automate de Glushkov**.

Définition 6 (algorithme de Berry-Sethi)

Soit r une expression régulière sur un alphabet Σ.

1. Linéariser r pour obtenir r'.
2. Calculer $P(\mathcal{L}(r'))$, $S(\mathcal{L}(r'))$ et $F(\mathcal{L}(r'))$.
3. Construire $\mathcal{A} = \mathrm{Loc}(\mathcal{L}(r'))$.
4. Effacer les indices des symboles des transitions de $\mathcal{A}$.

Exemple

Soit $r = (a(ab)^*) + (b^*a)$.

Linéarisation de r : $r' = a_1(a_2b_3)^*|b_4^*a_5$.

Calcul des ensembles :

$$P(\mathcal{L}(r')) = \{a_1, b_4, a_5\}$$
$$S(\mathcal{L}(r')) = \{a_1, b_3, a_5\}$$
$$F(\mathcal{L}(r')) = \{a_1a_2, a_2b_3, b_3a_2, b_4b_4, b_4a_5\}$$

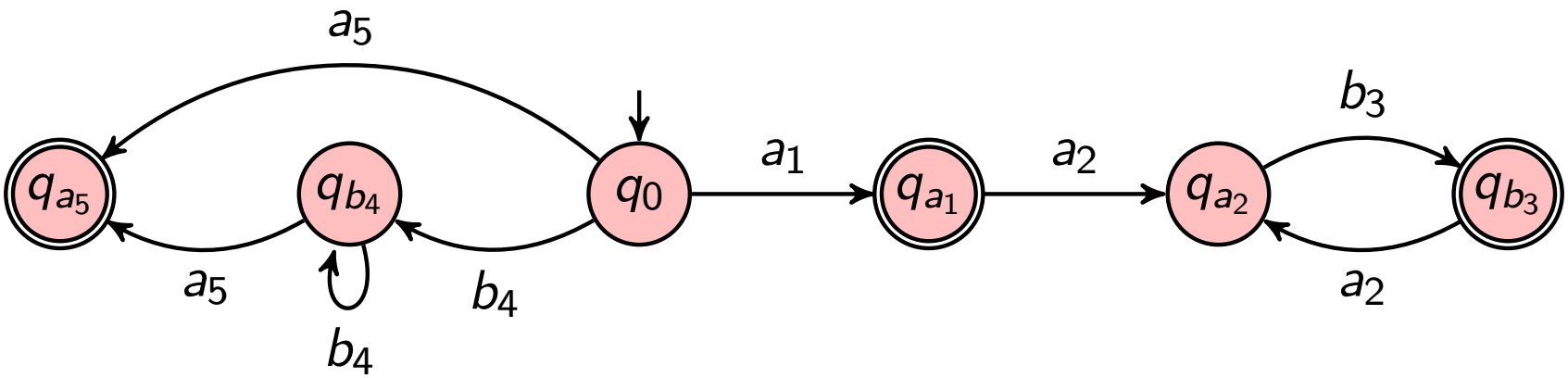
Exemple

Calcul de l'automate $\mathrm{Loc}(\mathcal{L}(r')) = (Q, \Sigma, q_0, F, \delta)$ avec :

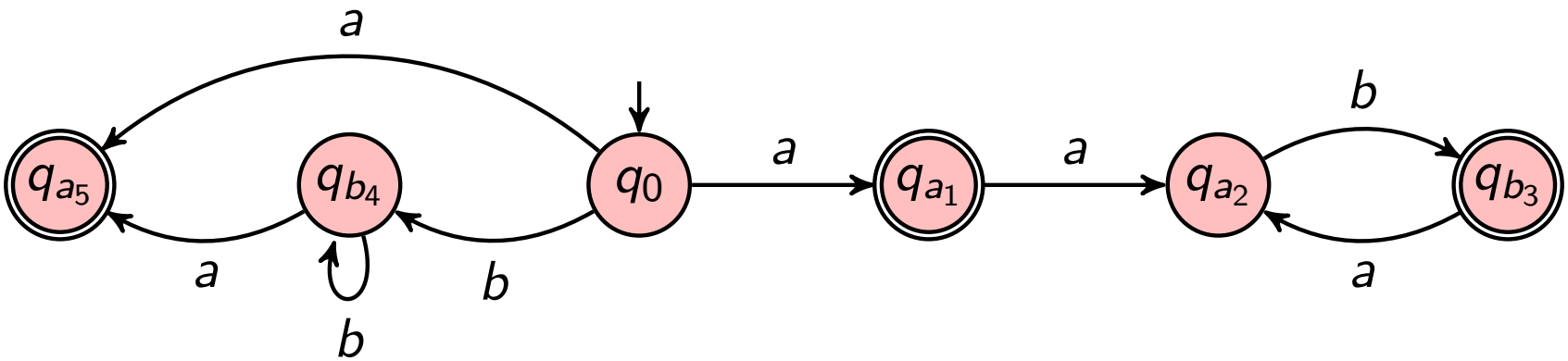
- $Q = \{q_0, q_{a_1}, q_{a_2}, q_{b_3}, q_{b_4}, q_{a_5}\}$
- $F = \{q_{a_1}, q_{b_3}, q_{a_5}\}$
- δ : $(q_0, a_1) \mapsto q_{a_1}$ $(q_{a_2}, b_3) \mapsto q_{b_3}$
  $(q_0, b_4) \mapsto q_{b_4}$ $(q_{b_3}, a_2) \mapsto q_{a_2}$
  $(q_0, a_5) \mapsto q_{a_5}$ $(q_{b_4}, b_4) \mapsto q_{b_4}$
  $(q_{a_1}, a_2) \mapsto q_{a_2}$ $(q_{b_4}, a_5) \mapsto q_{a_5}$

Exemple

Construction de l'automate local.



Effacement des indices des transitions.



Exemple

Soit $r = (a|b)^* c$.

Cette expression est déjà linéaire : $r' = (a_1|b_2)^* c_3$.

Calcul des ensembles :

$$P(\mathcal{L}(r')) = \{a_1, b_2, c_3\}$$
$$S(\mathcal{L}(r')) = \{c_3\}$$
$$F(\mathcal{L}(r')) = \{a_1 a_1, a_1 b_2, a_1 c_3, b_2 a_1, b_2 b_2, b_2 c_3\}$$

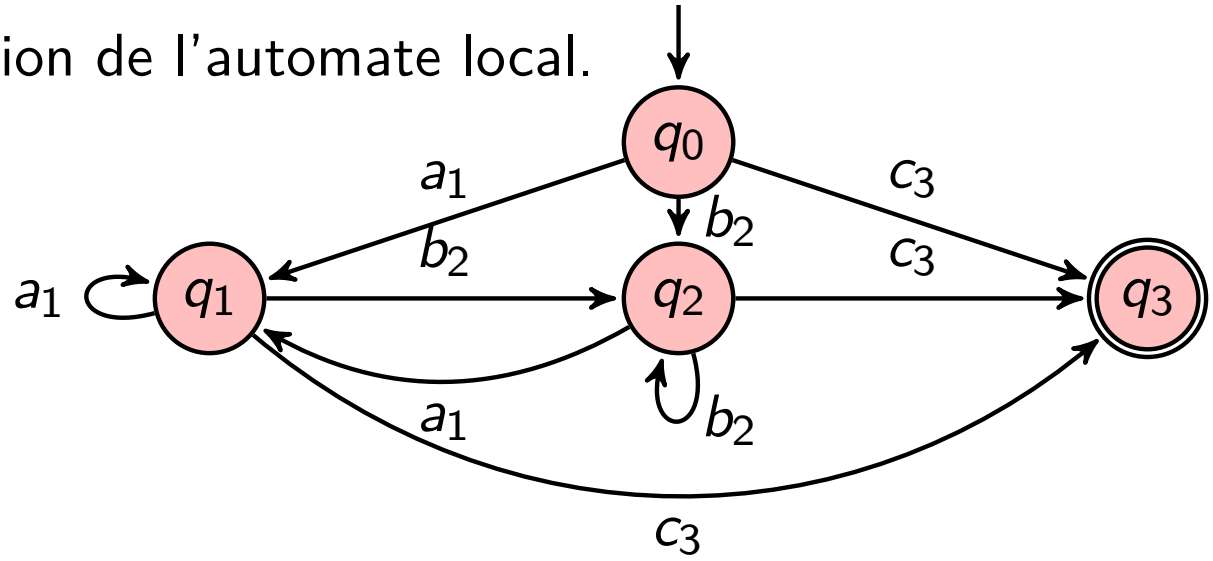
Exemple

Calcul de l'automate $\mathrm{Loc}(\mathcal{L}(r')) = (Q, \Sigma, q_0, F, \delta)$ avec :

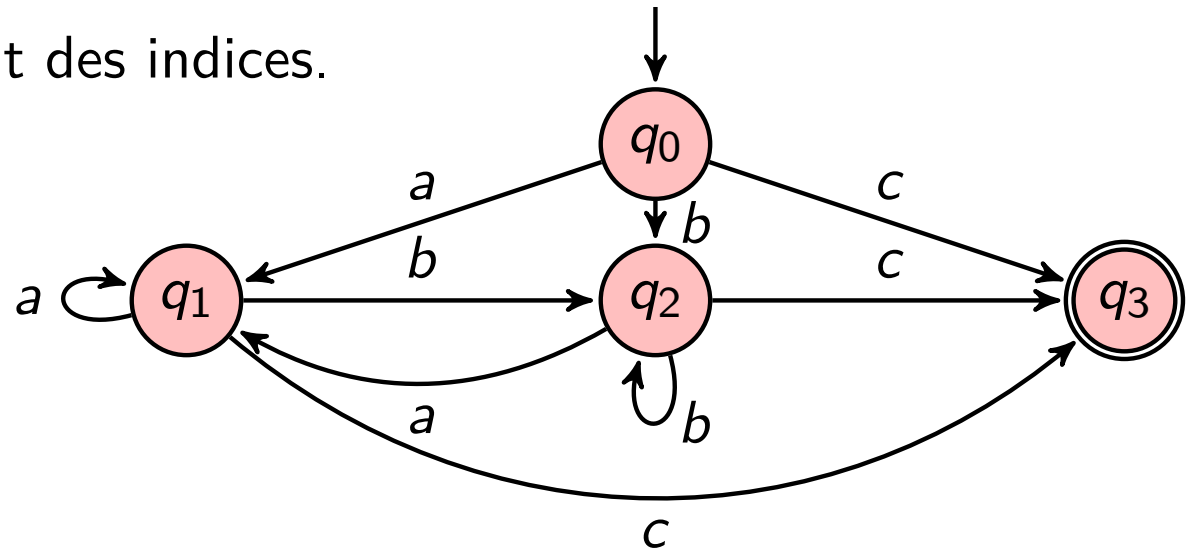
- $Q = \{q_0, q_{a_1}, q_{b_2}, q_{c_3}\}$
- $F = \{q_{c_3}\}$
- δ : $(q_0, a_1) \mapsto q_{a_1}$ $(q_{a_1}, a_1) \mapsto q_{a_1}$ $(q_{b_2}, a_1) \mapsto q_{a_1}$
  $(q_0, b_2) \mapsto q_{b_2}$ $(q_{a_1}, b_2) \mapsto q_{b_2}$ $(q_{b_2}, b_2) \mapsto q_{b_2}$
  $(q_0, c_3) \mapsto q_{c_3}$ $(q_{a_1}, c_3) \mapsto q_{c_3}$ $(q_{b_2}, c_3) \mapsto q_{c_3}$

Exemple

Construction de l'automate local.



Effacement des indices.



Exercice

Construire un automate de Glushkov qui reconnaît l'expression régulière suivante.

$$r = \Big(\big((ab) \,|\, \big(b^2c\big)^*\big)\, ab\Big)^* \,|\, \big(b^2c\big)^*$$

Expression linéarisée.

$$r' = \Big(\big((a_1b_1) \,|\, (b_2b_3c_1)^*\big)\, a_2b_4\Big)^* \,|\, (b_5b_6c_2)^*$$

Exercice

Expression linéarisée.

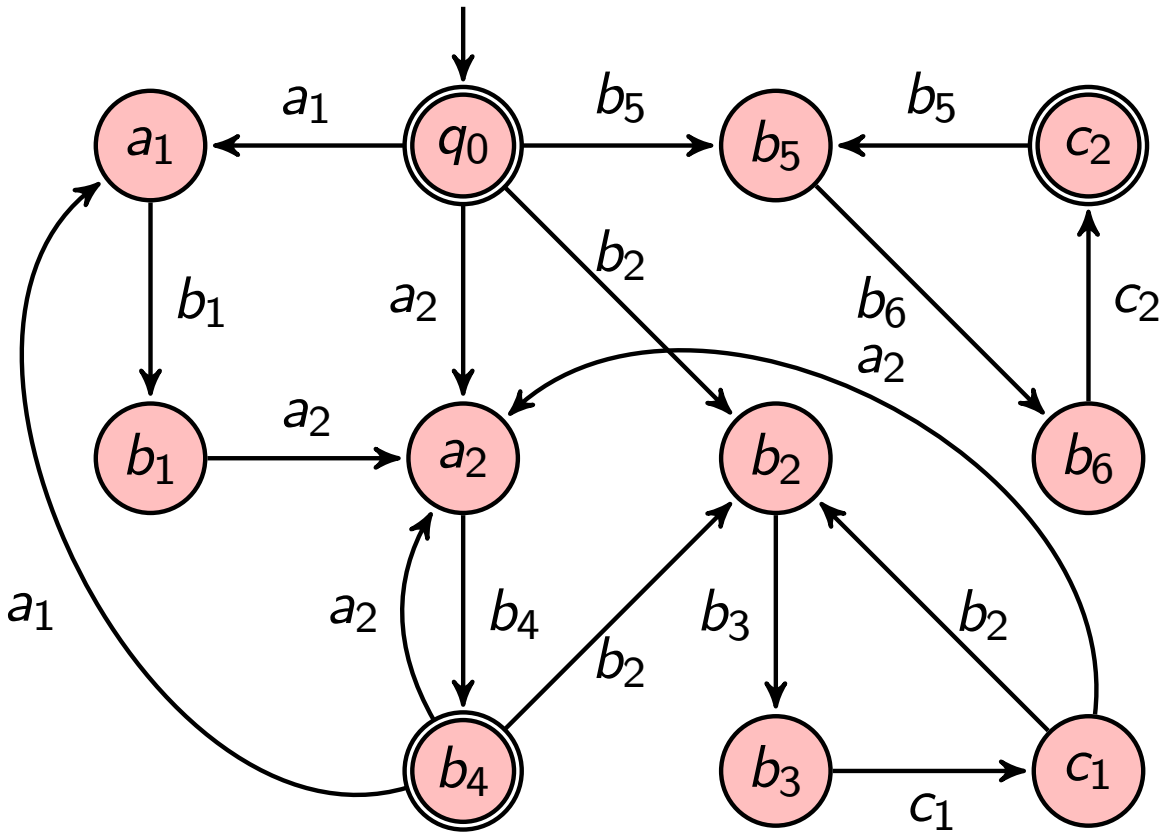
$$r' = \Big(\big((a_1 b_1) \,|\, (b_2 b_3 c_1)^*\big)\, a_2 b_4\Big)^* \,|\, (b_5 b_6 c_2)^*$$

Ensembles P, F, S.

$$P = \{a_1, a_2, b_2, b_5\}$$
$$F = \{a_1 b_1, b_1 a_2, b_2 b_3, b_3 c_1, b_4 a_1, b_4 b_2,$$
$$b_4 a_2, c_1 b_2, c_1 a_2, a_2 b_4, b_5 b_6, b_6 c_2, c_2 b_5\}$$
$$S = \{b_4, c_2\}$$

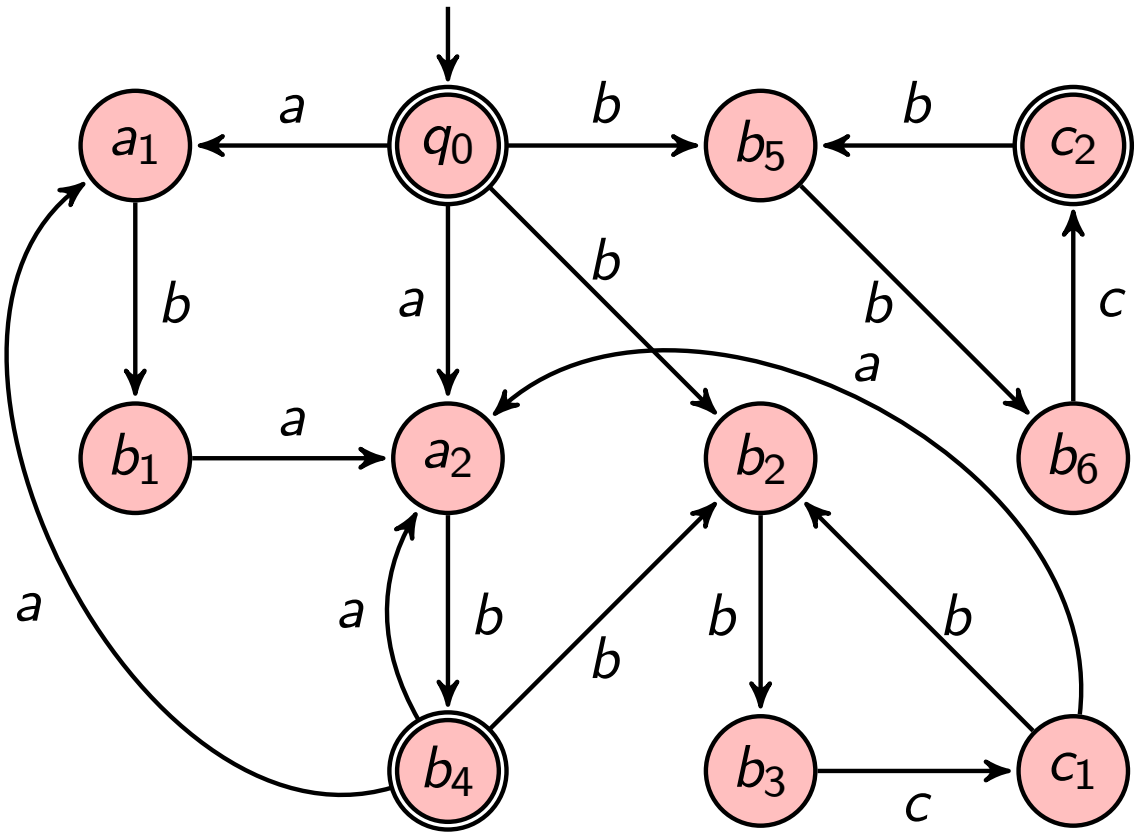
Exercice

Construction de l'automate local.



Exercice

Automate de Glushkov

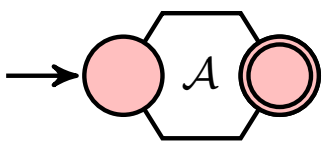


Algorithme de Thompson

À partir d'une expression régulière, l'algorithme de Berry-Sethi aboutit généralement à la construction d'un NFA. L'**algorithme de Thompson** construit un ε-NFA.

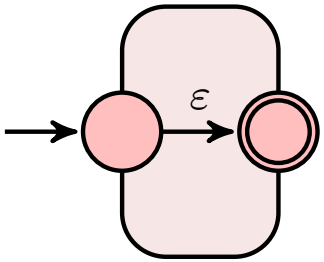
Cet algorithme associe des **règles de construction d'automates** aux **opérations élémentaires** de construction des expressions régulières (concaténation, union, étoile). Puis il **combine** les automates à l'aide d'ε-transitions.

Dans la suite du document, un automate $\mathcal{A}$ est représenté de manière compacte sous la forme suivante.

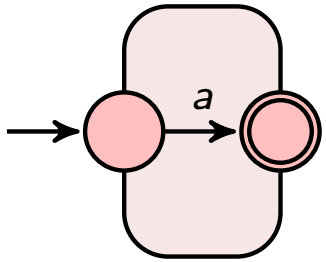


Algorithme de Thompson

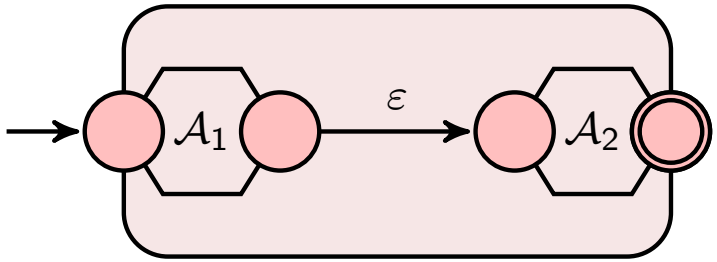
Les règles de constructions sont les suivantes.



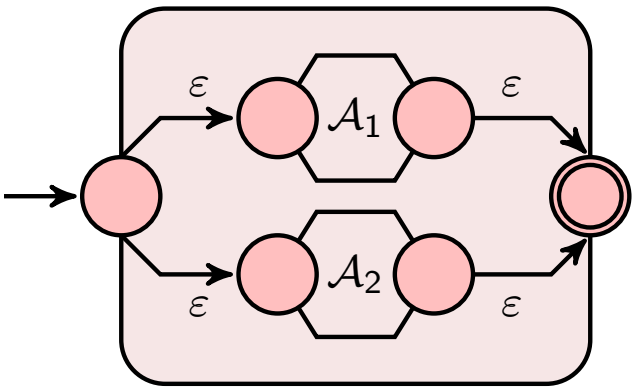
Mot vide $\mathcal{A}_\varepsilon$



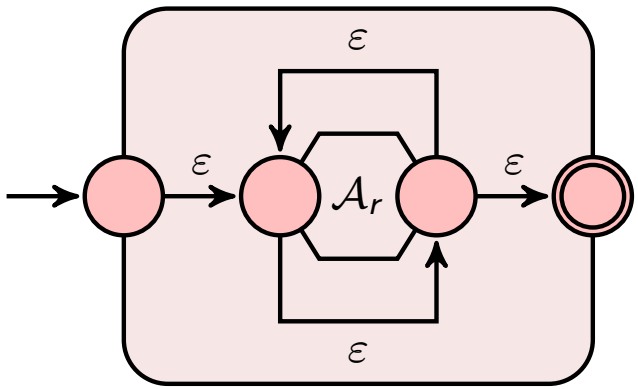
Lettre $\mathcal{A}_a$



Concaténation $\mathcal{A}_{r_1 r_2}$



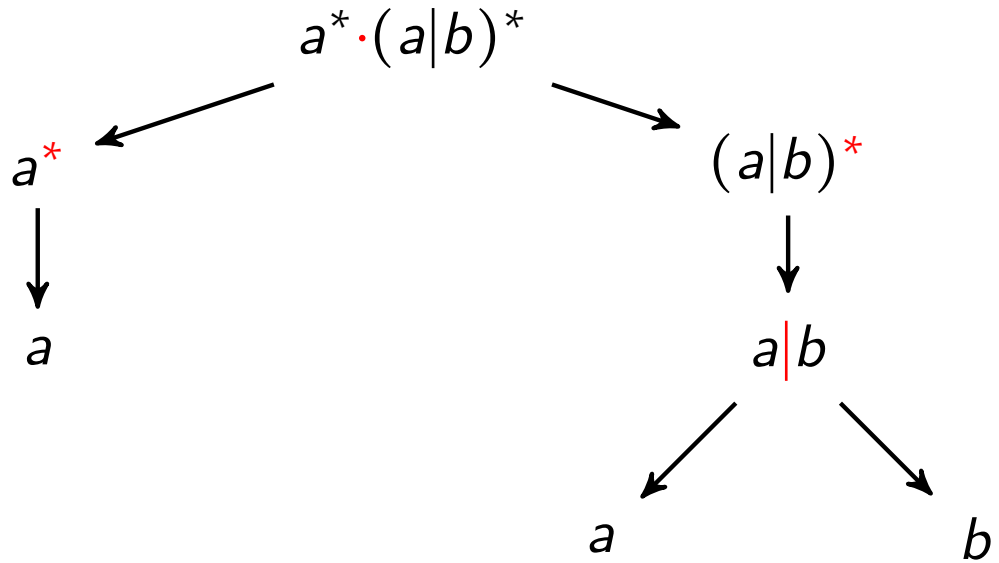
Union $\mathcal{A}_{r_1|r_2}$



Étoile de Kleene $\mathcal{A}_{r^*}$

Exemple

Illustrons la construction d'un automate associé à l'expression régulière suivante $a^* \cdot (a|b)^*$ en utilisant l'arbre syntaxique associé à la décomposition de cette dernière.

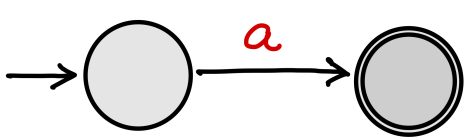


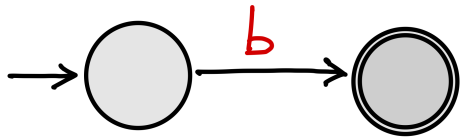
On applique alors les règles de Thompson en partant des feuilles puis on remonte progressivement dans l'arbre.

Il n'est pas nécessaire d'étiqueter les états pendant la construction.

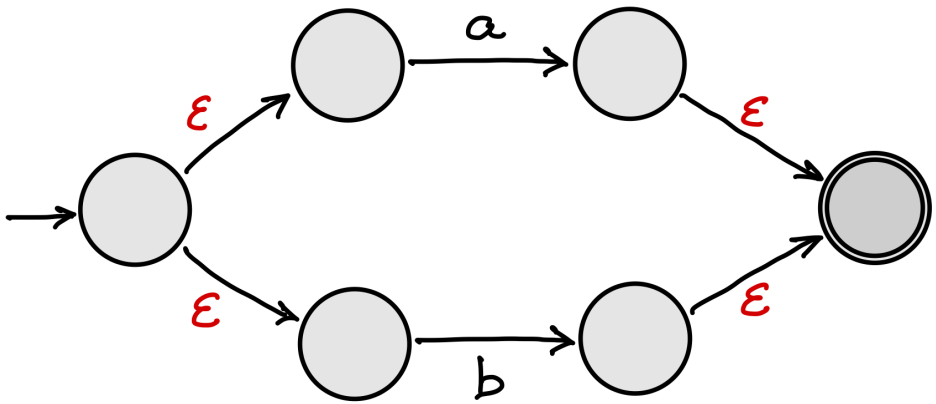
Exemple

Construction de a et b



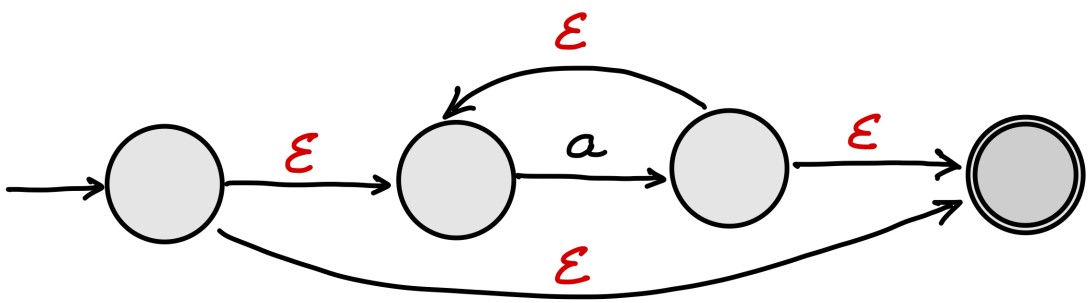


Construction de $(a|b)$



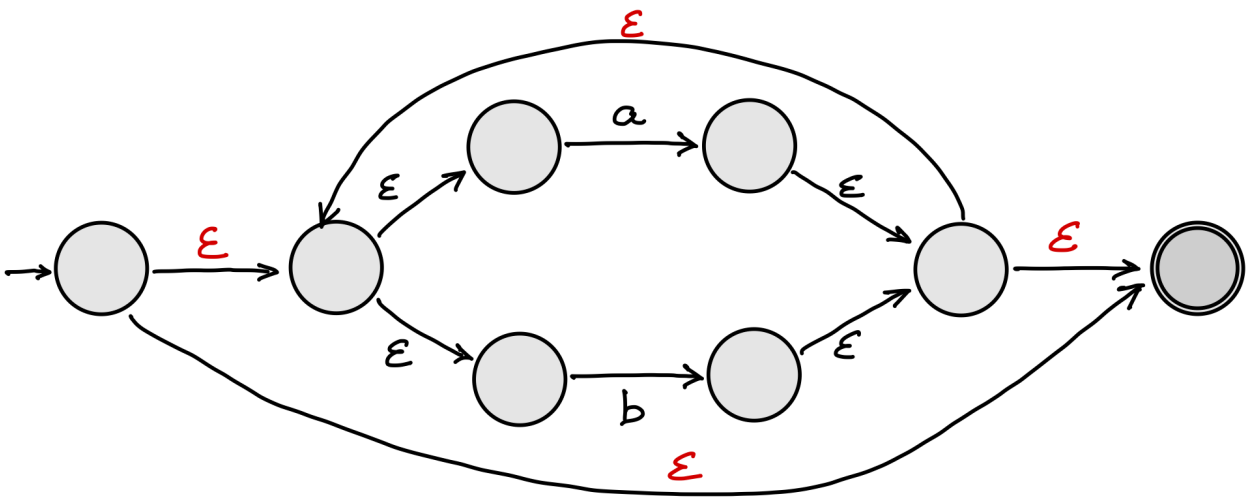
Exemple

Construction de a^*



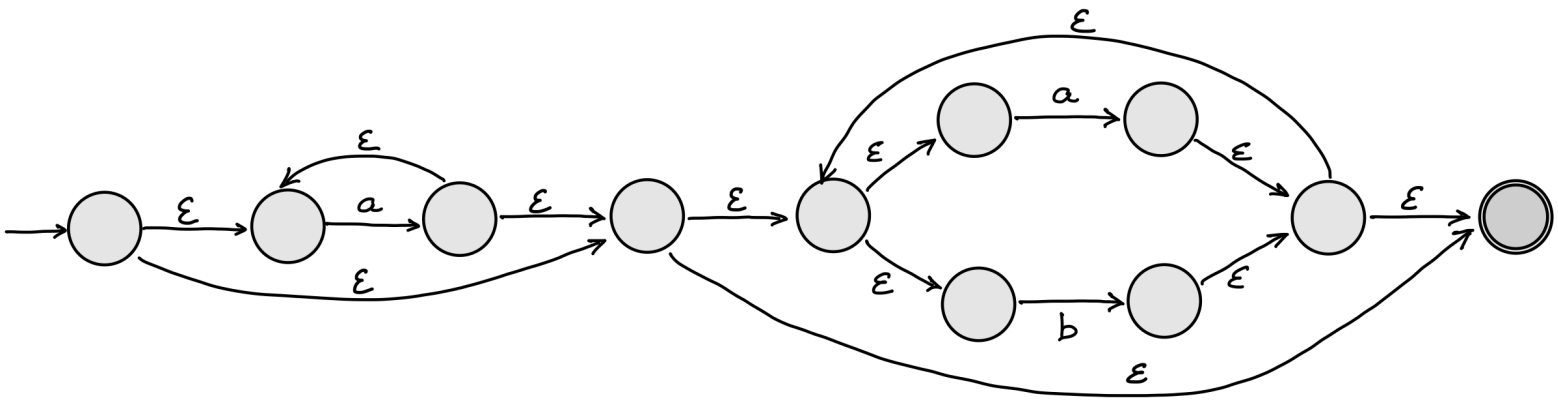
Exemple

Construction de $(a|b)^*$



Exemple

Construction de $a^*(a|b)^*$



Exercice

À l'aide de l'algorithme de Thompson, construire les automates associés aux expression régulière suivantes.

- $a^*(a|b)b$
- $(a^*b^*)^*$
- $(\varepsilon|a|aa)a^*$
- $(a|bb)^*(b|aa)^*$
- $b(ab)^*|(ba)^*b$
- $(a|b)^*(aaa|bbb)(a|b)^*$

De l'automate à l'expression régulière

Automate généralisé

Un automate généralisé est un automate dont les transitions sont étiquetées par des expressions régulières plutôt que par des symboles.

Définition 7 (automate généralisé)

Un **automate généralisé** est un quintuplet $\mathcal{A} = (Q, \Sigma, q_0, F, \delta)$:

- Q est un ensemble d'états ;
- Σ est un alphabet ;
- $q_0 \in Q$ est l'état initial ;
- $F \subseteq Q$ est l'ensemble des états acceptants ;
- $\delta \subseteq Q \times \mathrm{Reg}(\Sigma) \times Q$ est la relation de transition de l'automate.

Élimination des états

Un **automate généralisé** est utilisé comme **structure de données** auxiliaire pour stocker petit à petit les expressions régulières partielles générées à partir d'un automate initial pour lequel on cherche à déterminer une expression régulière qui dénote le langage qu'il reconnaît.

En pratique, on procède par **élimination des états** en suivant un **algorithme dit de Brzozoswki et McCuskey**.

Algorithme d'élimination des états

Soit $\mathcal{A} = (Q, \Sigma, q_0, F, \delta)$ un automate, déterministe ou non, avec ou sans transition spontanée.

1 - Initialisation

Créer l'automate généralisé $\mathcal{A}' = (Q \cup \{q_i, q_f\}, \Sigma, q_i, \{q_f\}, \delta')$ avec

$$\delta' = \delta \cup \{(q_i, \varepsilon, q_0)\} \cup \{(q, \varepsilon, q_f) \mid q \in F\}$$

Cet automate a un seul état initial q_i, sans transition entrante, et un seul état acceptant q_f, sans transition sortante.

Algorithme d'élimination des états

2 - Procédures auxiliaires

Élimination des transitions.
Tant qu'il existe deux transitions distinctes $q \xrightarrow{r_1} q'$ et $q \xrightarrow{r_2} q'$ dans δ, les supprimer et les remplacer par une unique transition $q \xrightarrow{r_1|r_2} q'$.

Élimination d'un état.
Pour tous $p \xrightarrow{r_1} q$ et $q \xrightarrow{r_2} s$ dans δ :

- ajouter $p \xrightarrow{r_1 r^* r_2} s$ à δ si $q \xrightarrow{r} q$
  ajouter $p \xrightarrow{r_1 r_2} s$ sinon ;
- supprimer $p \xrightarrow{r_1} q$ et $q \xrightarrow{r_2} s$ de δ.

Supprimer q de Q.

Algorithme d'élimination des états

3 - Boucle principale

Pour chaque état $q \in Q$,

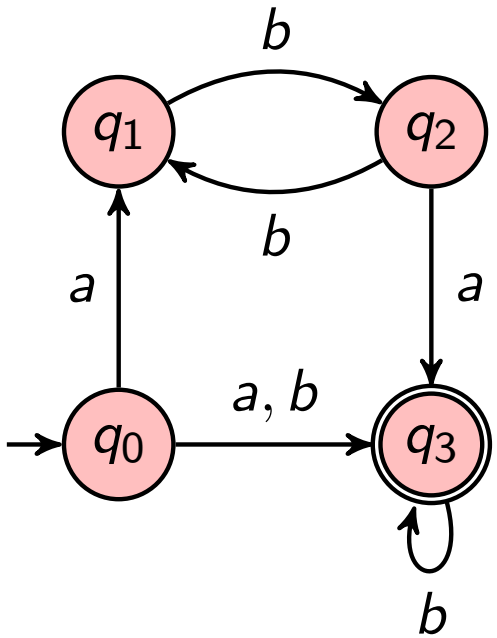
- éliminer toutes les transitions possibles ;
- éliminer l'état q.

4 - Résultat

Une fois l'automate réduit à $q_i \xrightarrow{r} q_f$, renvoyer r.

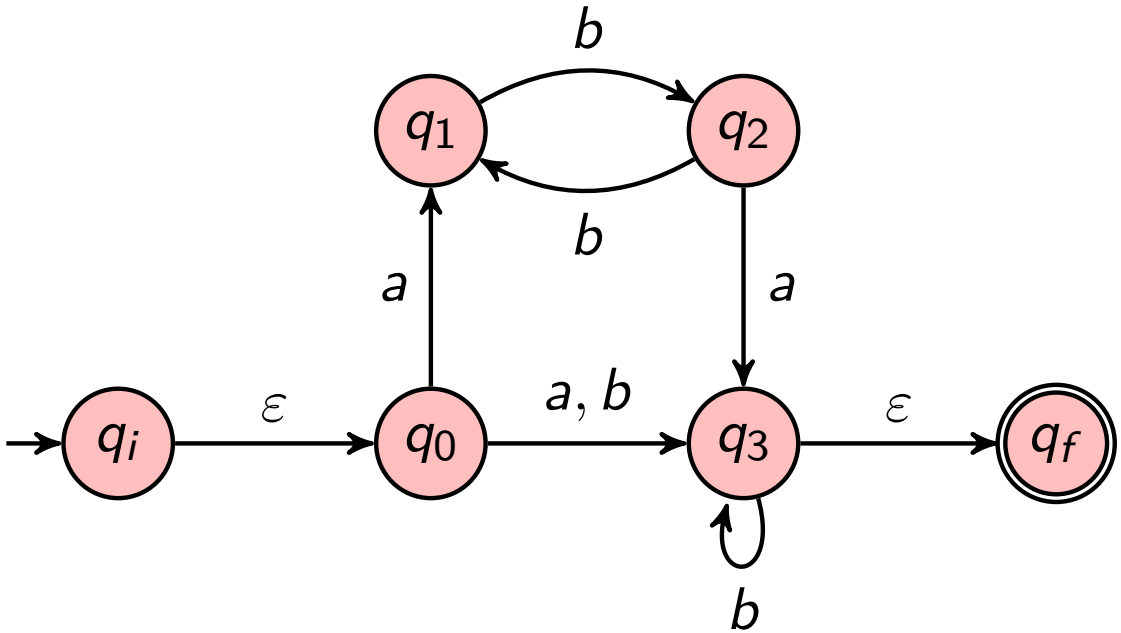
Exemple

Considérons l'automate $\mathcal{A}$ sur l'alphabet $\Sigma = \{a, b\}$, représenté par son graphe (complété avec q_i et q_f).



Exemple

Initialisation



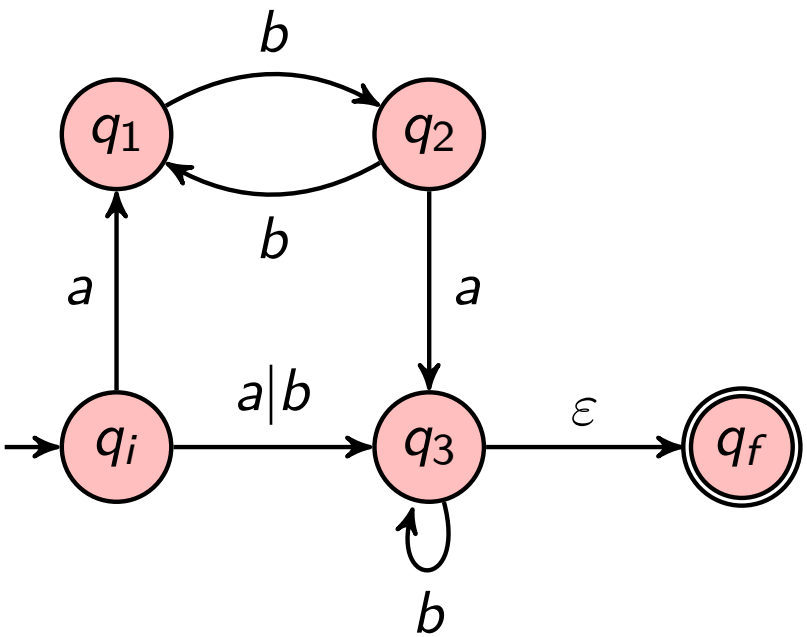
Exemple

Élimination de la transition $q_0 \longrightarrow q_3$

On élimine les transitions $q_0 \xrightarrow{a} q_3$ et $q_0 \xrightarrow{b} q_3$ pour les remplacer par une seule transition $q_0 \xrightarrow{a|b} q_3$.

Élimination de q_0

On considère ensuite $q_i \xrightarrow{\varepsilon} q_0$, $q_0, \xrightarrow{a} q_1$ et $q_i \xrightarrow{\varepsilon} q_0$, $q_0, \xrightarrow{a|b} q_3$.
On crée $q_i \xrightarrow{a} q_1$ et $q_i \xrightarrow{a} q_3$ et on supprime q_0.



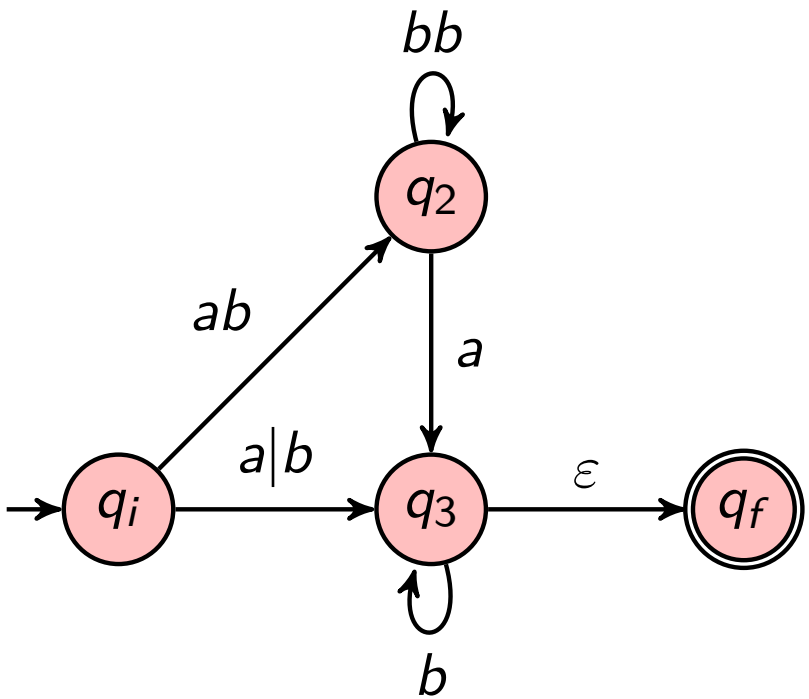
Exemple

Élimination de q_1

Pas de transitions à éliminer.

On considère $q_i \xrightarrow{a} q_1, q_1 \xrightarrow{b} q_2$ et $q_2 \xrightarrow{b} q_1, q_1 \xrightarrow{b} q_2$.

On crée $q_i \xrightarrow{ab} q_2$ et $q_2 \xrightarrow{bb} q_2$ et on supprime q_1.



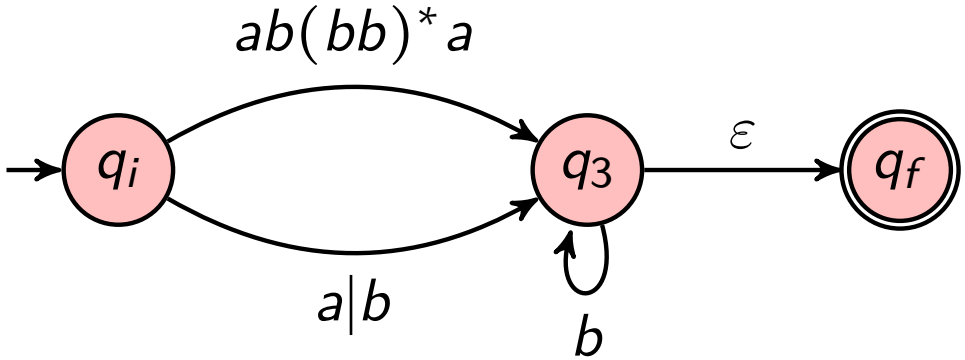
Exemple

Élimination de q_2

Pas de transition à éliminer.

On considère $q_i \xrightarrow{ab} q_2, q_2 \xrightarrow{a} q_3$.

On crée $q_i \xrightarrow{ab(bb)^*a} q_3$ (on n'oublie pas la boucle sur q_2 qui se transforme en $(bb)^*$).

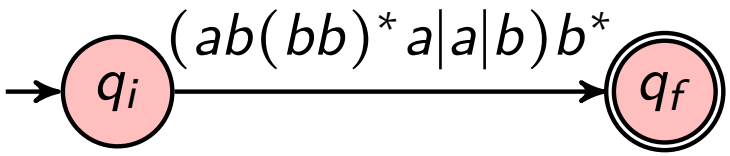


Exemple

Élimination de q_3

On élimine les transitions $q_i \xrightarrow{abb^*a} q_3$ et $q_i \xrightarrow{a|b} q_3$ que l'on remplace par $q_i \xrightarrow{abb^*a|a|b} q_3$.

On considère cette transition et $q_3 \xrightarrow{\varepsilon} q_f$ pour obtenir $q_i \xrightarrow{(ab(bb)^*a|a|b)b^*\varepsilon} q_f$.



L'automate est réduit aux états q_i et q_f, l'expression recherchée est :

$$(ab(bb)^*a|a|b)b^*$$

Remarques

La taille des expressions régulières renvoyées par cet algorithme peut, dans le pire cas, être exponentielle en le nombre d'états de l'automate de départ (certains cas sont inévitables).

Le choix de l'ordre des sommets peut influencer la taille finale de l'expression. Dans l'exemple, nous avons simplifié la présentation en choisissant les états dans l'ordre $q_0, \ldots, q_n$, mais on peut choisir à chaque itération n'importe quel état autre que q_i et q_f.

De nombreuses simplifications algébriques et heuristiques sont utilisées en pratique pour obtenir des expressions plus courte et plus lisibles.

Cet algorithme est cependant suffisant pour achever la preuve du théorème de Kleene : **pour tout automate fini $\mathcal{A}$, il existe une expression régulière r telle que $\mathcal{L}(r) = \mathcal{L}(\mathcal{A})$.**

Algorithme par résolution d'équations

Considérons un état q_i d'un automate $\mathcal{A} = (Q, \Sigma, I, F, \delta$ où I désigne l'ensemble des états initiaux, F l'ensemble des états acceptants. Soit X_q l'ensemble des mots qui font passer de l'état q à un état acceptant. Le langage reconnu par $\mathcal{A}$ est alors :

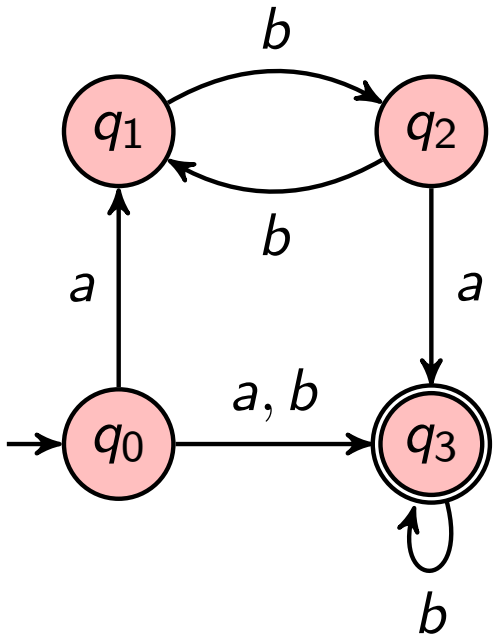
$$\mathcal{L}(\mathcal{A}) = \bigcup_{q \in I} X_q$$

Si $A_{p,q}$ désigne l'ensemble des étiquettes des transitions de l'état p vers l'état q, on a également :

$$X_p = \begin{cases} \bigcup_{q \in I} A_{p,q} X_q & \text{si } p \text{ n'est pas acceptant ;} \\ \{\varepsilon\} \cup \bigcup_{q \in I} A_{p,q} X_q & \text{si } p \text{ est acceptant ;} \end{cases}$$

Algorithme par résolution d'équations

Illustrons cette mise en équation avec l'automate vu plus haut.



Comme souvent dans ce genre de calculs, on confond expressions régulières et langages correspondants afin d'alléger les notations. On note également X_i le langage reconnu depuis l'état q_i.

Algorithme par résolution d'équations

Le système d'équation est alors le suivant.

$$\begin{cases} X_0 &= aX_1 \mid (a|b)X_3 \\ X_1 &= bX_2 \\ X_2 &= bX_1 \mid aX_3 \\ X_3 &= \varepsilon \mid bX_3 \end{cases}$$

Le langage reconnu par l'automate est celui donné par X_0.
Pour résoudre ce système, on fait appel au **lemme d'Arden**.

Lemme d'Arden

Théorème 8 (version simplifiée)

Soient K et L deux langages sur le même alphabet Σ.
K^*L est solution de l'équation $X = KX \cup L$.
Si $\varepsilon \notin K$, il s'agit de la seule solution.

Dans le cas général, à savoir si $\varepsilon \in K$, le lemme établit que X est solution de $KX \cup L$ si et seulement s'il existe un langage Y tel que $X = K^*(L \cup Y)$.

Ce résultat étant HP, nous nous limitons à la version simplifiée du lemme pour répondre aux besoins les plus courants !

Lemme d'Arden

Démonstration

Existence. On raisonne par double inclusion. On remarque tout d'abord que K^*L est solution de l'équation.

$$K(K^*L) \cup L = K^+L \cup L = (K^+ \cup \{\varepsilon\})L = K^*L$$

Ce qui montre que $X \subseteq K^*L$.
Réciproquement, si X est solution alors $L \subseteq X$ et $KX \subseteq X$. Par récurrence, pour tout entier naturel n, on établit $K^nL \subseteq X$ et donc $K^*L \subseteq X$.

Unicité. Supposons l'existence d'un mot $w \in X \setminus K^*L$ solution de l'équation qui soit de taille minimale. Puisque $w \notin L$ alors $w \in KX$. Si on suppose $\varepsilon \notin K$, ce mot peut s'écrire $w = kx$ avec $k \in K$, $|k| \geqslant 1$, et $x \in X$. x étant plus court que w, il appartient à K^*L. Ce qui entraîne que w appartient aussi à K^*L, résultat en contradiction avec l'hypothèse. Donc si $\varepsilon \notin K$, K^*L est l'unique solution de l'équation.

Exemple

En appliquant ce résultat au système précédent, on trouve tout d'abord :

$$X_3 = b^*$$

Ce qui permet d'écrire :

$$X_2 = bX_1 \mid ab^*$$

puis :

$$X_1 = b(bX_1 \mid ab^*) = bbX_1 \mid bab^* = (bb)^* bab^*$$

Et enfin :

$$X_0 = a\,(bb)^*\, bab^* \mid (a|b)b^*$$

Notons que la détermination de l'expression finale de X_2 n'est pas indispensable, l'objectif étant de trouver X_0.

Remarques

L'expression régulière $a\,(bb)^*\,bab^* \mid (a|b)b^*$ est à rapprocher de celle obtenue par la mise en œuvre de l'algorithme d'éliminations, à savoir $(ab(bb)^*a|a|b)b^*$. Ces deux expressions sont équivalentes (le montrer).

Cette dernière observation permet d'envisager l'utilisation du lemme d'Arden pour établir l'équivalence d'expressions régulières. Par exemple, en partant de l'automate ci-dessous et en résolvant le système d'équations obtenu de trois façons différentes, on établit l'équivalence des trois expressions régulières.

$$(a|b)^* \qquad (a^*b)^*a^* \qquad (b^*a)^*b^*$$