Langages et automates

Définitions générales

Alphabet : Un alphabet est un ensemble fini, non vide, de symboles (appelés aussi lettres ou caractères). On le note généralement Σ. Le nombre de symboles qu'il contient est appelé sa cardinalité et dénoté par |Σ|.
Exemple : Le langage machine est basé sur l'alphabet Σ = {0, 1}. |Σ|=2

Mot (ou chaîne) : Un mot sur un alphabet Σ est une suite finie de symboles de Σ.
Exemple :

- Le mot 1011 est défini sur l'alphabet {0, 1}
- Le mot 6.1239 est défini sur l'alphabet {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, .}

Longueur : La longueur d'un mot w notée |w| est le nombre de symboles de ce mot.
Exemple :

- Le mot w = 010 est de longueur |w| = 3.
- Le mot u = 01110 est de longueur |u| = 5.
- Le **mot vide** noté ε est le seul mot de longueur nulle, |ε|=0.

On note :

- Σ^* l'ensemble de tous les mots (y compris le mot vide ε) définis sur Σ.
- Σ^+ représente l'ensemble des mots sur l'alphabet Σ de longueur ≥ 1.
- Σ^n représente l'ensemble des mots sur l'alphabet Σ de longueur n.

Opérations sur les mots

Concaténation : La concaténation de deux mots u et v de Σ^* est un mot noté u.v ou uv et défini par : $u = u_1u_2\ldots u_n$, $v = v_1v_2\ldots v_n$, $w = u.v = uv = u_1\ldots u_nv_1\ldots v_n$, $|uv|=|u|+|v|$
Cette opération est associative, non commutative, et ε est l'élément neutre.

Propriété de la concaténation

Soient w, w_1 et w_2 trois mots définis sur l'alphabet Σ :

- $|w1.w2| = |w1| + |w2|$
- $(w1.w2).w3 = w1.(w2.w3)$ (la concaténation est associative)
- $w.\varepsilon = \varepsilon.w = w$ (ε est un élément neutre pour la concaténation)

Exposant (puissance)

La puissance n-ième d'un mot est définie par : $w^n = w\ldots w$ (n fois), $w^0 = \varepsilon$, $w^{n+1} = w^nw$.

Mot miroir

Soit $w = a_1a_2...a_n$ un mot définit sur l'alphabet Σ. On appelle mot miroir de w et on le note par w^R le mot obtenu en écrivant w à l'envers, c'est-à-dire : $w^R = a_n...a_2a_1$. Il est donc facile de voir que $(w^R)^R = w$.

Mots conjugués

Les mots w_1 et w_2 sont dits conjugués s'il existe deux mots u et v tels que : $w_1 = uv$ et $w_2 = vu$.

Préfixe et suffixe

Un mot $u \in \Sigma^*$ est préfixe du mot $w \in \Sigma^*$ s'il existe $v \in \Sigma^*$ tel que w = uv. Le mot v est dit suffixe du mot w.

Projection

Soit le mot u ∈ Σ* et un alphabet $\overline{\Sigma} \subseteq \Sigma$. La projection de u sur $\overline{\Sigma}$, notée u ↑ $\overline{\Sigma}$, est la suite obtenue de u en gardant les symboles qui appartiennent à $\overline{\Sigma}$ et effaçant les autres symboles.

Exemple : Soit Σ = {a, b, c} et $\overline{\Sigma}$ = {a, b}. On considère le mot u = cacbcca, sa projection sur $\overline{\Sigma}$ est obtenue en gardant les a et b et en effaçant tous les c et donc u ↑ $\overline{\Sigma}$ = aba.

Langage

Un langage L sur Σ est un sous-ensemble de Σ^*, c'est-à-dire, L ∈ P(Σ^*) ou L ⊆ Σ^*. Sa cardinalité, à savoir le nombre de mots qu'il contient, est notée |L|.

Exemple :

- Langage des nombre binaires définies sur l'alphabet Σ ={0, 1} (langage infini)
- Langage des mots de longueur 2 défini sur l'alphabet Σ ={a, b}, L={aa, ab, ba, bb}
- L= ∅ est le langage vide, il ne contient aucun mot

Opérations sur les langages

Soit L, L_1 et L_2 trois langages définis sur l'alphabet Σ.

- Union (somme) : $L_1 \cup L_2 = L_1 + L_2 = \{w \in \Sigma^* \mid w \in L_1 \text{ ou } w \in L_2\}$
- Intersection : $L_1 \cap L_2 = \{w \in \Sigma^* \mid w \in L1 \text{ et } w \in L2\}$
- Concaténation (produit) : $L = L_1.L_2 = L_1L_2 = \{w \in \Sigma^* \mid \exists\, w_1 \in L_1 \text{ et } \exists\, w_2 \in L_2 : w = w_1w_2\}$
- Différence : $L_1\text{-}L_2 = \{w \in \Sigma^* \mid w \in L_1 \text{ et } w \notin L_2\}$
- Complément : Le complémentaire d'un langage L est noté $\Sigma^* \setminus L = \{w \in \Sigma^* \mid w \notin L\}$
- Puissance : $L^{n+1} = L^n L$ avec $L^0 = \varepsilon$
- Clôture de Kleene (Kleene star) ou étoile de L : La clôture de Keene d'un langage L est l'ensemble des mots construits par une concaténation finie des mots de L notée :

  $$L^* = \bigcup_{k=0}^{+\infty} L^k = L^0 \cup L^1 \cup L^2 \cup \cdots \cup L^n \cup \cdots$$

  $L^* = \{w \in \Sigma^* \mid \exists\, k \in N, \exists\, w_1, w_2, \ldots, w_k \in L : w = w_1w_2\ldots w_k\}$.

  Ce langage contient un nombre infini de mots, qui sont les répétitions indéfinies de mots du langage L.
- Fermeture plus : $L^+ = \bigcup_{k=1}^{+\infty} L^k = L^1 \cup L^2 \cup \cdots \cup L^n \cup \cdots$
- Préfixe-clôture : on appelle la préfixe-clôture de L le langage $\overline{L}$ contenant tous les préfixes des mots de L : $\overline{L} = \{w_1 \in \Sigma^* \mid \exists\, w_2 \in \Sigma^* : w_1w_2 \in L\}$. On a par définition $L \subseteq \overline{L}$. Un langage L est dit préfixe-clos si $L = \overline{L}$.

Exemple : Soient les langages L_1 = {ab,ba} , L_2 = {c,cc} et L_3 = {ac} . Nous avons :

- L_1L_2 = {abc,abcc,bac,bacc}
- $L_1L_2L_3$ = {abcac,abccac,bacac,baccac}
- L_1^2 = {abab,abba,baab,baba}
- $\overline{L_1}$ = {ε, a, ab, b, ba}

Propriétés des opérations sur les langages

Soit L, L_1, L_2 et L_3 quatre langages définis sur l'alphabet Σ :

- $L^* = L^+ + \{\varepsilon\}$
- $L^+ = L^* L = L L^*$
- $L_1.(L_2.L_3) = (L_1.L_2).L_3$
- $L_1.(L_2 + L_3) = (L_1.L_2) + (L_1.L_3)$
- $L.L \neq L$

- $L_1.(L_2 \cap L_3) \neq (L_1 \cap L_2).(L_1 \cap L_3)$
- $L_1.L_2 \neq L_2.L_1$
- $(L^*)^* = L^*$
- $L^*.L^* = L^*$;
- $L_1.(L_2.L_1)^* = (L_1.L_2)^*.L_1$
- $(L_1 + L_2)^* = (L_1^* L_2^*)^*$
- $L_1^* + L_2^* \neq (L_1 + L_2)^*$

Langages réguliers (rationnels)
En théorie des langages, les langages rationnels ou langages réguliers ou encore langages reconnaissables peuvent être décrits de plusieurs façons équivalentes :

– Ce sont les langages décrits par les expressions régulières ou rationnelles, d'où le nom de langages réguliers ;
– Ce sont les langages obtenus, à partir des lettres et de l'ensemble vide, par les opérations rationnelles (union, produit et l'étoile de Kleene), d'où le nom de langages rationnels ;
– Ce sont les langages reconnus par des automates finis, d'où le nom de langages reconnaissables.

Les expressions régulières (ou rationnelles) sont une notation qui indique comment un ensemble régulier est construit à partir des ensembles réguliers élémentaires.

Remarque :
– Tout langage fini est rationnel.
– Si L est rationnel, alors L* est rationnel.
– Si L_1 et L_2 sont rationnels, alors $L_1 + L_2$ et L_1L_2 sont rationnels

Expressions régulières
Les expressions régulières (ER) pour un alphabet Σ sont les expressions formées par les règles suivantes :
- Règle 1 : ∅ et chaque lettre de Σ est une ER
- Règle 2 : si α et β sont des ER, alors (αβ), (α+β) et α^* sont des ER
- Règles 3 : On ne peut obtenir des expressions régulières qu'en appliquant de manière finie les règles 1 et 2 énoncées ci-dessus.

Le langage L(ER) représenté par l'expression régulière ER est défini ainsi :
- $L(\emptyset) = \emptyset$, $L(\varepsilon) = \{\varepsilon\}$
- $L(a) = \{a\}$ pour tout $a \in \Sigma$
- $L(\alpha\beta) = L(\alpha).L(\beta)$
- $L(\alpha+\beta) = L(\alpha)+L(\beta)$
- $L(\alpha^*) = L(\alpha)^*$

Exemple :
Soit l'alphabet Σ = {a, b, c}
(a + b + c)* : L'ensemble de tous les mots définis à partir de l'alphabet Σ
$(a + b + c)^*a$: L'ensemble de tous les mots finissant par le symbole a
$(a + b + c)\ (a + b + c)^*$: L'ensemble de tous les mots non vides définis à partir de l'alphabet Σ
$(a + b)^*b(a + b)^*$: Définit l'ensemble des mots composés par a et b contenant au moins un b.
0 + 10* : C'est le langage formé du mot 0 et des mots composés d'un 1 suivi d'un nombre quelconque de 0 : {0, 1, 10, 100, . . .}.

Remarque :
Ordre de priorité sur les opérations : priorité (étoile) > priorité (concaténation) > priorité (union). L'expression 0 + 10* est donc équivalente à (0 + (1(0)*)).

Quelques équivalences utiles :
(ab)*a = a(ba)* ; a^+=aa*=a*a ; a*=(ε + a)* =a*a*=(a*)* ;
(a + b)* = (a*b*)* = (a*+b*)* = (a*b)*a* ;

Automates Finis
On rencontre couramment des automates finis (ou automates à états finis) dans de nombreux appareils qui réalisent des actions déterminées en fonction des évènements qui se présentent.
Exemples :
- Un distributeur automatique de boissons qui délivre l'article souhaité quand le montant introduit est approprié,
- Les ascenseurs qui savent combiner les appels successifs pour s'arrêter aux étages intermédiaires,
- Les feux de circulation capables de s'adapter aux voitures en attente,
- Des digicodes qui analysent la bonne suite de chiffres,

On distingue deux types d'automates finis :
- AFD : automate fini déterministe
- AFN : automate fini non déterministe

Automate fini déterministe
Un automate fini déterministe (AFD) est définit par un quintuplet A = (∑, Q, δ, q_0, F) où :
- ∑ est un ensemble fini de symboles appelé alphabet.
- Q est un ensemble fini dont les éléments sont appelés états.
- δ est une relation de Q × ∑ → Q appelée fonction de transition ou ensemble des transitions de A.
- q_0 est un état de Q appelé état initial.
- F est un sous-ensemble de Q appelé ensemble des états finaux (ou états marqués) de A.

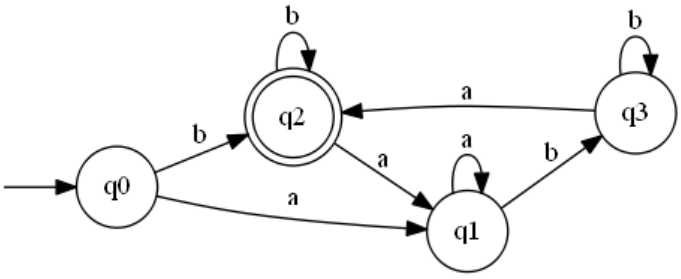
Un automate fini peut être représenté de deux manières : soit par une table définissant la fonction de transition soit par un graphe orienté.

Représentation par table de transition
Le nombre de ligne de la table est égale au nombre d'états dans l'automate de telle sorte que chaque ligne correspond à un état. Les colonnes correspondent aux différents symboles de l'alphabet. Si l'automate est dans l'état q_i et que le symbole a est le prochain à lire, alors l'entrée (q_i, a) de la table donne l'état auquel l'automate passera après avoir lu a. Notons qu'il faut préciser l'état initial et les états finaux (l'état initial est précédé d'une flèche « → », l'état final d'une étoile « * »)

Représentation graphique
Un automate fini peut être représente graphiquement comme un graphe orienté dont les sommets (des cercles) sont les états et les arcs sont les transitions. Une transition $\delta(q_i, a)=q_j$ est représentée par un arc reliant les sommets q_i et q_j, étiqueté par a. L'état initial est désigné par une flèche entrante au sommet correspondant tandis que les états finaux sont marqués par un double cercle.

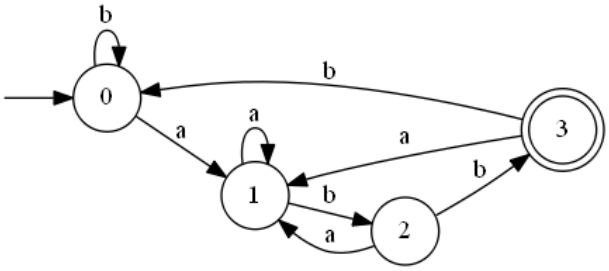
Exemple :

	a	b
→ q_0	q_1	q_2
q_1	q_1	q_3
*q_2	q_1	q_2
q_3	q_2	q_3



Exemple :

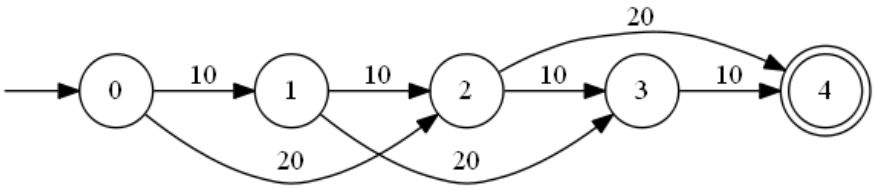
	a	b
→0	1	0
1	1	2
2	1	3
*3	1	0



Exemple : Distributeur de café

Le café coûte 40 DA et les pièces acceptées sont celles de 10 et 20 DA. Il n'y a pas de retour de monnaie.

L'alphabet Σ = {10, 20} ; Ensemble des états : $Q = \{0,1,2,3,4\}$; Ensemble des états finaux : $F = \{4\}$; Etat initial : 0.



Remarque : Dans un AFD, les conditions suivantes sont vérifiées :

- $\forall q_i \in Q, \forall a \in \Sigma$, il existe au plus un état q_j tel que $\delta(q_i, a) = q_j$. C'est-à-dire, pour chaque état, il existe au maximum une transition issue de cet état possédant le même symbole.
- L'automate ne comporte pas des ε-transitions (epsilon-transitions)
- Il existe un seul état initial

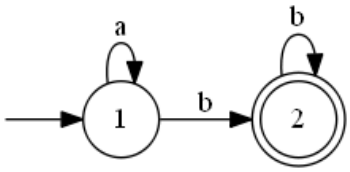
Remarque : Un AFD ne comportant pas d'états finaux est appelé **accepteur** : $A = (\Sigma, Q, \delta, q_0)$.

Automate fini complet – complétion

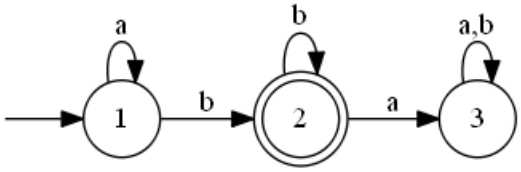
Un automate fini est complet si δ est défini sur tout $Q \times \Sigma$, i.e., on peut transiter depuis chaque état vers un autre état sur tous les symboles de Σ.

Si un automate n'est pas complet, on peut le compléter en lui ajoutant un nouvel état, qui n'est pas un état final, appelé état puits dans lequel aboutirons toutes les transitions manquantes.

Exemple : Soit l'automate fini incomplet suivant :



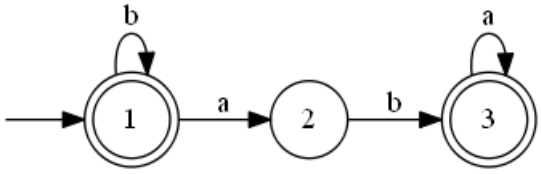
On peut construire l'automate fini complet suivant (l'état 3 est l'état puits) :



Langage reconnu (accepté) par un AFD

Le langage L(A) reconnu par l'automate A est l'ensemble des mots permettant d'atteindre un état final à partir de l'état initial : $L(A) = \{\omega \in \Sigma^* \mid \delta(q_0, \omega) \in F\}$

Exemple : On considère l'automate fini déterministe suivant :



Calcul sur le mot *baba* : $1 \xrightarrow{b} 1 \xrightarrow{a} 2 \xrightarrow{b} 3 \xrightarrow{a} 3$ (accepté) ce mot est accepté parce que le calcul se termine sur l'état 3 qui est un état final.

Calcul sur le mot *bba* : $1 \xrightarrow{b} 1 \xrightarrow{b} 1 \xrightarrow{a} 2$ (rejeté) ce mot n'est pas accepté parce que le calcul se termine sur l'état 2 qui n'est pas un état final.

Calcul sur le mot *abab* : $1 \xrightarrow{a} 2 \xrightarrow{b} 3 \xrightarrow{a} 3$ (impasse) ce mot n'est pas accepté parce que l'automate se bloque.

Remarque : Si l'automate se bloque pendant la lecture d'un mot ou n'atteint pas l'état final, ce mot n'est pas reconnu par l'automate

Langage généré par un AFD

Le langage généré par un automate A = (∑, Q, δ, q_0) contient tous les mots qui permettent de passer de l'état initial de l'automate à n'importe quel état :

$L_g(A) = \{\omega \in \Sigma^* \mid \delta(q_0, \omega) \in Q\}$, i.e. $\delta(q_0, \omega)$ est définie.

Le langage généré par un automate est toujours préfixe-clos.

Propriété d'un automate fini déterministe

Un automate fini déterministe est :

- Accessible (ou réalisable) si tous ses états sont accessibles. Un état q est accessible s'il existe un chemin partant de l'état initial qui arrive dans l'état q.
- Co-accessible (ou co-réalisable) si tous ses états sont co-accessibles. Un état q est co-accessible s'il existe un chemin partant de q et qui arrive dans un état final.
- Non-bloquant si tous ses états sont non bloquants. Un état q est bloquant s'il est accessible mais pas co-accessible.
- Emondé (utile) s'il est accessible et co-accessible.

Automates finis non déterministes (AFN)

Un automate fini non déterministe (AFN) est définit par un quintuplet A = (∑, Q, δ, I, F) où :

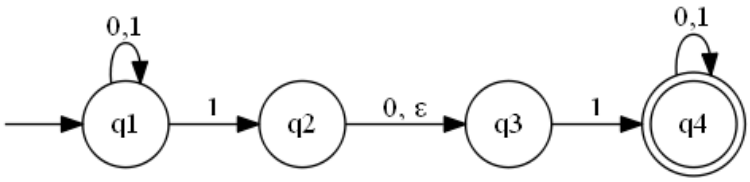
- ∑ est un ensemble fini de symboles appelé alphabet.
- Q est un ensemble fini dont les éléments sont appelés états.
- δ est une relation de Q × (∑ ∪ {ε}) → P(Q) (l'ensemble des parties de Q) appelée fonction de transition ou ensemble des transitions de A.

- I est un sous-ensemble de Q appelé ensemble des états initiaux de A.
- F est un sous-ensemble de Q appelé ensemble des états finaux de A.

Un AFN admet des transitions étiquetées par ε et des états dont plus d'une transition sortante ont la même étiquette. Un mot w est reconnu par un AFN s'il existe un état final dans $\delta(q_0,w)$.

Exemple : On considère l'AFN suivant

	0	1	ε
→ q_1	$\{q_1\}$	$\{q_1, q_2\}$	∅
q_2	$\{q_3\}$	∅	$\{q_3\}$
q_3	∅	$\{q_4\}$	∅
*q_4	$\{q_4\}$	$\{q_4\}$	∅



Est-ce que cet automate accepte le mot 011 ?

$$^{1}\nearrow q_2 \xrightarrow{\varepsilon} q_3 \text{ (rejeté)}$$

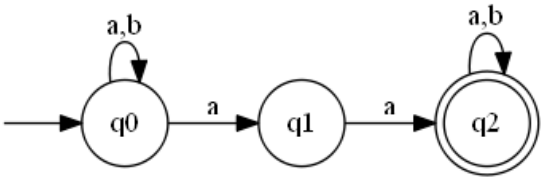
$$q_1 \xrightarrow{0} q_1 \xrightarrow{1} q_1 \xrightarrow{1} q_1 \text{ (rejeté)}$$

$$\searrow^{1} q_2 \xrightarrow{\varepsilon} q_3 \xrightarrow{1} q_4 \text{ (accepté)}$$

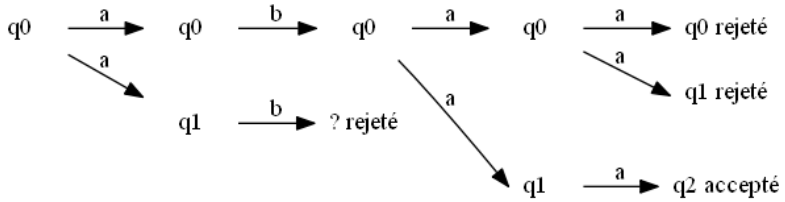
Le mot 011 est accepté parce que l'un des chemins aboutit à l'état final q_4.

Exemple : Soit l'AFN suivant

	a	b
→ q_0	$\{q_0, q_1\}$	$\{q_0\}$
q_1	$\{q_2\}$	∅
*q_2	$\{q_2\}$	$\{q_2\}$



Est ce que cet automate accepte le mot abaa ?



Le mot abaa est accepté parce que l'un des chemins aboutit à l'état final q_2.

Remarque : Il est important de comprendre qu'un AFN évalue simultanément tous les chemins possibles. Si au moins l'un de ces chemins conduit à un état final, l'automate déclare avoir reconnu le mot.

Langage reconnu (accepté) par un AFN

Le langage L(A) reconnu par un AFN A est l'ensemble des mots permettant d'atteindre un état final à partir d'un état initial : $L(A) = \{\omega \in \Sigma^* \mid \delta(q_0, \omega) \cap F \neq \varnothing\}$

Remarque **:**

- Pour tout automate fini A, il existe une expression régulière dénotant le langage L(A).
- Pour toute expression régulière ER, il existe un automate fini qui reconnaît le langage défini par ER.

Passage de l'automate fini vers l'expression régulière

Etant donné un automate fini A, on veut construire une expression régulière dénotant le langage L(A). Le principe est d'exprimer les langages de chaque état q dans un système d'équations d'inconnues L_q. Pour chaque état, on écrit une équation de la forme :

- Si l'état q n'est pas un état final : $L_q = \sum_{\left(q \xrightarrow{a} p\right) \in \delta} a\, L_p$
- Si l'état q est un état final, alors ε appartient à L_q : $L_q = \sum_{\left(q \xrightarrow{a} p\right) \in \delta} a\, L_p + \varepsilon$

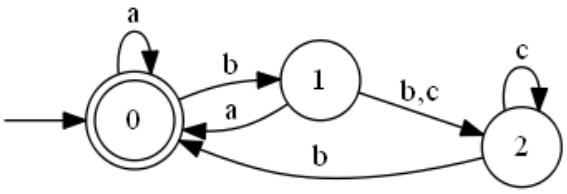
On obtient un système de N équations avec N inconnues (N est le nombre d'états de l'automate). On cherche une expression régulière pour L_{q0}.

Lemme d'Arden

L'équation récursive suivante : L = X L + M, où L est le langage cherché et X et M sont deux langages connus, a une unique solution qui est : $L = X^* M$

Exemple :

1) Langage reconnu par cet AFD.



$L_0 = a\, L_0 + b\, L_1 + \varepsilon$;

$L_1 = a\, L_0 + (b + c)\, L_2$;

$L_2 = b\, L_0 + c\, L_2$;

Ce qui donne :

$L_2 = c^* b\, L_0$

$L_1 = a\, L_0 + (b + c)\, L_2 = a\, L_0 + (b + c) c^* b\, L_0 = \left(a + (b + c) c^* b\right) L_0$

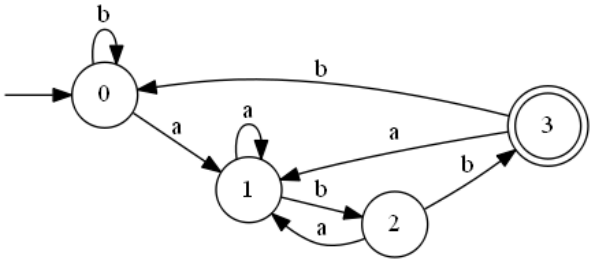
$L_0 = a\, L_0 + b\left(a + (b + c) c^* b\right) L_0 + \varepsilon = \left(a + b\left(a + (b + c) c^* b\right)\right) L_0 + \varepsilon$

$\Rightarrow L_0 = \left(a + b\left(a + (b + c) c^* b\right)\right)^*$

Alors, le langage reconnu par cet AFD est donné par l'expression régulière :

$\left(a + b\left(a + (b + c) c^* b\right)\right)^*$

2) Langage reconnu par cet AFD.



$L_0 = b\, L_0 + a\, L_1$;

$L_1 = a\, L_1 + b\, L_2$;

$L_2 = a\, L_1 + b\, L_3$;

$L_3 = a\, L_1 + b\, L_0 + \varepsilon$

Ce qui donne :

$L_2 = (a + ba)\, L_1 + bb\, L_0 + b$

$L_1 = \left(a + b(a + ba)\right) L_1 + bbb\, L_0 + bb \Rightarrow L_1 = \left(a + b(a + ba)\right)^* (bbb\, L_0 + bb)$

$L_0 = b\, L_0 + a\left(\left(a + b(a + ba)\right)^* (bbb\, L_0 + bb)\right)$

$= \left(b + a\left(a + b(a + ba)\right)^* bbb\right) L_0 + a\left(a + b(a + ba)\right)^* bb$

$\Rightarrow L_0 = \left(b + a\left(a + b(a + ba)\right)^* bbb\right)^* a\left(a + b(a + ba)\right)^* bb$

Alors, le langage reconnu par cet AFD est donné par l'expression régulière :

$\left(b + a\left(a + b(a + ba)\right)^* bbb\right)^* a\left(a + b(a + ba)\right)^* bb$

Remarques sur les AFN

- Un mot est accepté par un automate non déterministe s'il existe une dérivation qui accepte ce mot. Nous laissons l'automate « choisir » la bonne dérivation.
- Si un langage L est accepté par un AFND alors il existe un AFD acceptant L.
- Un langage est régulier ssi il est accepté par un automate fini.
- Le non déterminisme permet de construire facilement des automates utilisant la disjonction et l'itération, mais ne reconnaît pas plus de langages que les automates déterministes.
- Les automates finis déterministes et non déterministes reconnaissent la même classe de langages.
- Ce résultat est à la fois surprenant et pratique.
    - Il est surprenant car les automates non déterministes semblent plus puissants que les automates déterministes.
    - Il est pratique car il est souvent plus simple de décrire un langage à l'aide d'un automate non déterministe.

Passage de l'expression régulière vers l'automate fini

Tout langage défini par une expression régulière est aussi défini par un automate fini.

Algorithme de Glushkov

C'est un algorithme qui permet de construire un automate fini (pas nécessairement déterministe et sans ε-transitions) à partir d'une expression régulière.
Entrée : une expression régulière : (aba + b)* + a*

Etape 1 : Linéariser l'expression régulière.
On remplace toutes les lettres de l'expression par des symboles distincts (x_i pour la lettre en *i*-ème position) : $\left(x_1x_2x_3 + x_4\right)^* + x_5^*$.

Etape 2 : Déterminer

- *Premier* : l'ensemble des symboles pouvant commencer un mot.
- *Dernier* : l'ensemble des symboles pouvant terminer un mot.
- *Suivant* : pour tout symbole x_i, l'ensemble des symboles pouvant suivre x_i.

Etape 3 : Construction de l'AFN

- L'ensemble des états :
    - un état initial q_0.
    - un état q_i par symbole x_i.
- L'ensemble des états finaux :
    - l'état initial q_0 si ε appartient au langage.
    - un état q_i pour tout x_i appartenant à Dernier.
- La fonction de transition :
    - une transition de l'état initial q_0 vers l'état q_i pour tout x_i appartenant à Premier et étiquetée par la lettre correspondant à x_i ($\delta(q_0, x_i) = q_i$ *si* $x_i \in Premier$).

- une transition de l'état q_i vers l'état q_j pour tout x_j appartenant à Suivant(x_i) et étiquetée par la lettre correspondant à x_j ($\delta(q_i, x_j) = q_j$, *pour tout* $x_j \in Suivant(x_i)$).
- Puis remplacer les symboles x_i par les lettres d'origine de l'expression régulière.

	suivant
x_1	x_2
x_2	x_3
x_3	x_1 , x_4
x_4	x_1 , x_4
x_5	x_5

Remarque : L'automate obtenu est souvent non déterministe.

Exemple :

Construction un AFN qui reconnait le langage : (aba + b)* + a*

Linéarisation de l'expression régulière : $\left(x_1x_2x_3 + x_4\right)^* + x_5^*$

Premier = $\{x_1, x_4, x_5\}$

Dernier = $\{x_3, x_4, x_5\}$

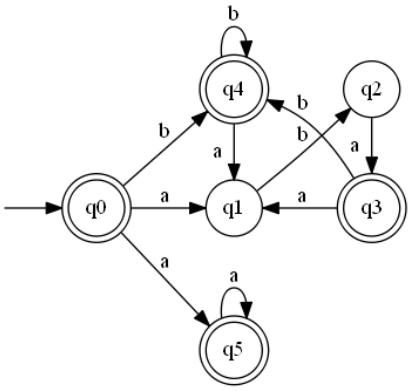
Ensemble des états de l'automate fini

$Q = \{q_0, q_1, q_2, q_3, q_4, q_5\}$

Etat initial : q_0

Etats finals : $\{q_0, q_3, q_4, q_5\}$

(ε appartient au langage ⇒ l'état initial est un état final).



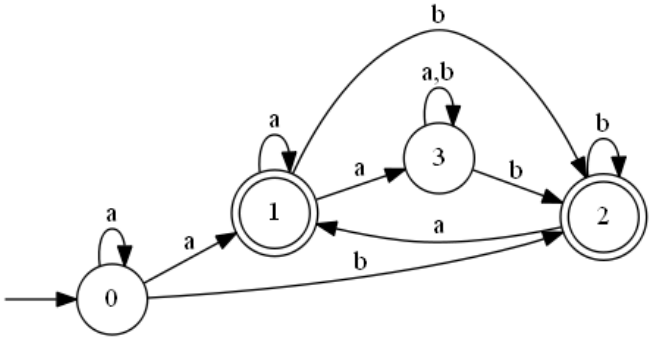
Déterminisation d'un automate

Théorème : pour tout automate fini non déterministe, il existe un automate fini déterministe équivalent (« équivalent » signifie qu'il reconnaît le même langage).

Procédure de déterminisation :

- **Etape 1** : calculer l'ε-fermeture (clôture, étoile) de chaque état.
  L'ε-fermeture d'un état q est l'ensemble des états qui peuvent être atteints à partir de q en ne faisant que des ε-transitions, auquel on ajoute q lui-même.
- **Etape 2** : Créer la table de transition du nouvel automate comme suit
  1. Commencer par l'ε-fermeture de l'état initial (ou par l'union des ε-fermetures des états initiaux en cas de plusieurs états initiaux). Ce sera l'état initial du nouvel automate.
  2. Pour chaque lettre, ajouter dans la table les états produits, avec leurs ε-fermetures. Chacun de ces ensembles d'états devient un état dans le nouvel automate.
  3. Recommencer 2 à partir de chaque nouvel état produit, jusqu'à ce qu'on ne crée plus de nouvel état.
  4. Tous les états contenant au moins un état final deviennent finaux.
  5. Pour plus de clarté, renommer éventuellement les états.

Exemple : Déterminiser l'AFN suivant :



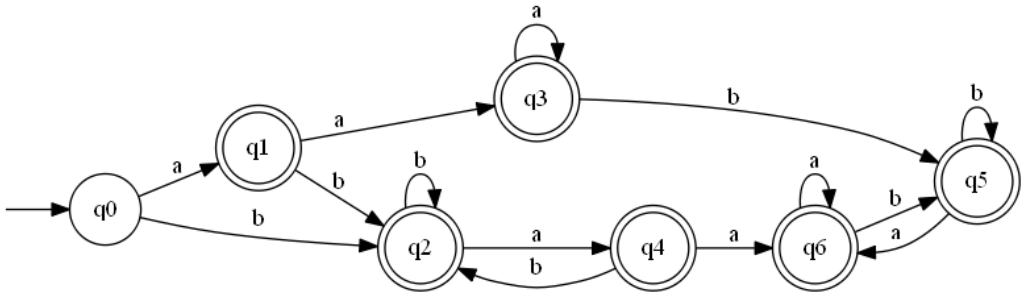
Dans cet automate, on n'a pas de ε-transitions, alors l'ε-fermeture de chaque est l'état lui-même.

La table de transition de l'automate fini déterministe résultant est :

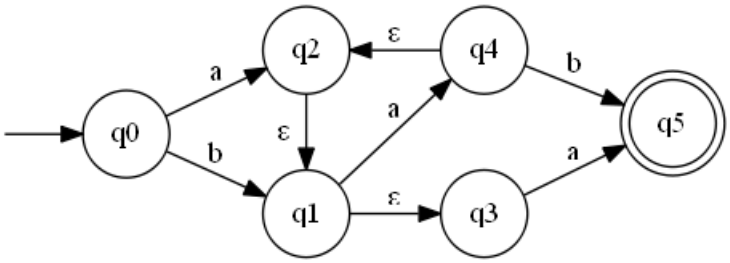
	a	b
→ 0	{0, 1}	{2}
*{0, 1}	{0,1,3}	{2}
*{2}	{1}	{2}
*{0,1,3}	{0,1,3}	{2,3}
*{1}	{1,3}	{2}
*{2,3}	{1,3}	{2,3}
*{1,3}	{1,3}	{2,3}

Après renommage des états, on obtient :

	a	b
→ q0	q1	q2
* q1	q3	q2
* q2	q4	q2
* q3	q3	q5
*q4	q6	q2
* q5	q6	q5
*q6	q6	q5



Exemple : Construire un AFD équivalent à l'AFN suivant.



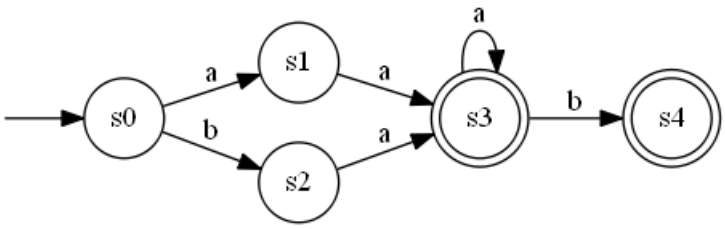
Calcul de l'ε-fermeture (clôture, étoile) de chaque état

état	ε-fermeture
q_0	$\{q_0\}$
q_1	$\{q_1, q_3\}$
q_2	$\{q_1, q_2, q_3\}$
q_3	$\{q_3\}$
q_4	$\{q_1, q_2, q_3, q_4\}$
q_5	$\{q_5\}$

Table de transition de l'automate fini déterministe

état	a	b
→ q_0	$\{q_1, q_2, q_3\}$	$\{q_1, q_3\}$
$\{q_1, q_2, q_3\}$	$\{q_1, q_2, q_3, q_4, q_5\}$	∅
$\{q_1, q_3\}$	$\{q_1, q_2, q_3, q_4, q_5\}$	∅
* $\{q_1, q_2, q_3, q_4, q_5\}$	$\{q_1, q_2, q_3, q_4, q_5\}$	$\{q_5\}$
* q_5	∅	∅

	a	b
→ s_0	s_1	s_2
s_1	s_3	∅
s_2	s_3	∅
* s_3	s_3	s_4
* s_4	∅	∅



Minimisation d'un AFD

La minimisation d'un automate fini déterministe (AFD) vise à transformer un AFD donné en un AFD équivalent comportant le nombre minimal d'états, tout en reconnaissant le même langage régulier.

Théorème de Myhill-Nérode : Soit L un langage régulier. Parmi tous les AFD reconnaissant L, il en existe un et un seul qui a un nombre minimal d'états.

En partant de n'importe quel AFD complet, une façon élémentaire pour obtenir l'automate minimal équivalent est :

- de supprimer les états inaccessibles à partir de l'état initial (et de compléter l'automate).
- d'identifier les états équivalents (indistinguables).
- Construire l'automate minimal en regroupant ensemble les états dans chaque classe d'équivalence.

Définition. Deux états q_1 et q_2 sont équivalents (indistinguables) si aucun mot ne les sépare : $q_1 \sim q_2$ ssi pour tout mot u, on a : $\delta(q_1, u) \in F$ implique $\delta(q_2, u) \in F$ et réciproquement.

Procédure de minimisation

Etape 1 : Supprimer les états inaccessibles à partir de l'état initial (et de compléter l'automate).

Etape 2 : Identifier les états équivalents (indistinguables).

- On commence avec un tableau de taille N × N (N est le nombre d'états de notre automate). Chaque case dans ce tableau correspond à un couple d'états (p,q). Pour ne pas avoir deux fois le même couple (p,q) et (q,p) ni les couples inutiles (p,p), on supprime alors ces cases (on obtient un tableau de taille (N-1) × (N-1)). Notre but final est de marquer toutes les cases (p,q) pour des états p et q non-équivalents ou distinguables.
- Au départ, les paires $(p \in F, q \notin F)$, i.e. (p état final, q état non final), sont marquées par D.
- Pour chaque case (p,q) et lettre a, on calcule $\delta(p,a) = p'$ et $\delta(q,a) = q'$. On marque (p,q) si et seulement si la case (p',q') est déjà marquée.
- On itère jusqu'à ce que le tableau ne se modifie pas.

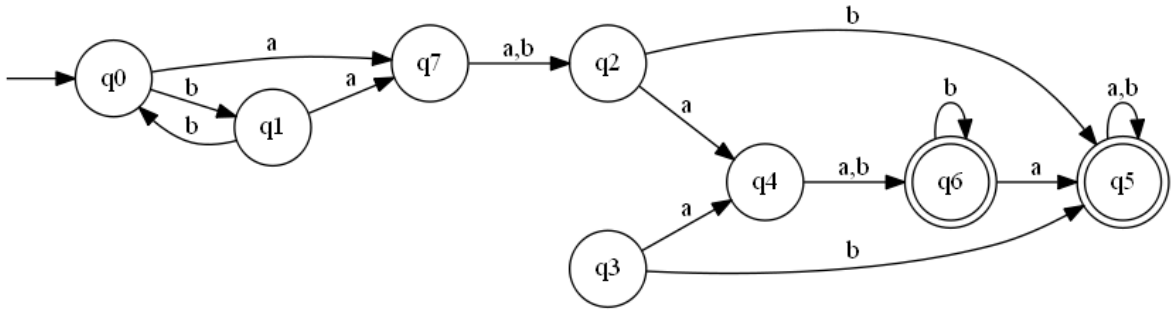
Etape 3 : Construire l'automate fini minimal (réduit)

- On détermine d'abord les classes d'équivalence. Pour chaque état q, la classe d'équivalence de q se compose de tous les états p pour lesquels la paire (p,q) n'est

pas marquée à l'étape 2 (dans le tableau). Les états de l'automate minimal sont les classes d'équivalence. L'état initial est la classe d'équivalence qui contient q_0. Les états finaux de l'automate minimal sont les classes d'équivalence qui se composent des états finaux de l'automate à minimiser. La fonction de transition est définie comme suit : Pour déterminer $\delta'(X,a)$, pour une classe d'équivalence X, choisissez n'importe quel $q \in X$ et déterminer $\delta'(X,a) = Y$, où Y est la classe d'équivalence qui contient $\delta(q,a)$.

- Supprimer tous les états non accessibles et tous les états non co-accessibles. Toutes les transitions vers ces états depuis d'autres états deviennent indéfinies.

Exemple : Minimiser l'AFD suivant



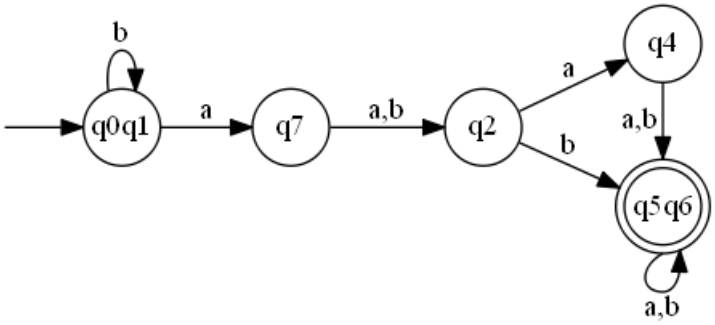
En premier lieu, on supprime l'état q3 parce que cet état n'est pas accessible à partir de l'état initial. Ensuite, on construit la table d'équivalence (sans q3)

.

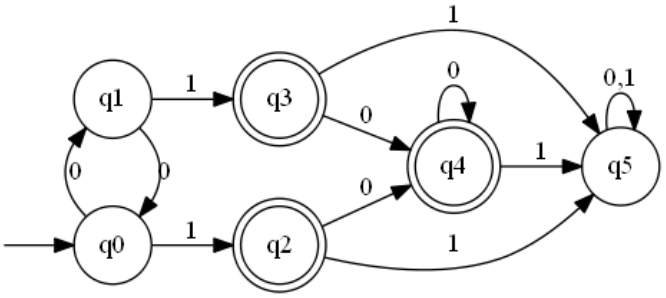
q1						
q2	D	D				
q4	D	D	D			
q5	D	D	D	D		
q6	D	D	D	D		
q7	D	D	D	D	D	D
	q0	q1	q2	q4	q5	q6

On regroupe les états dans des classes d'équivalence. A partir du tableau, q0 et q1 sont équivalents, et q5 et q6 sont équivalents. Alors, les classes d'équivalences sont : {q0, q1}, {q2}, {q4}, {q5, q6}, et la table de transitions de l'automate minimale est la suivante :

	a	b
→ q_0q_1	q_7	q_0q_1
q_2	q_4	q_5q_6
q_4	q_5q_6	q_5q_6
* q_5q_6	q_5q_6	q_5q_6
q_7	q_2	q_2



Exemple : Minimiser l'AFD suivant



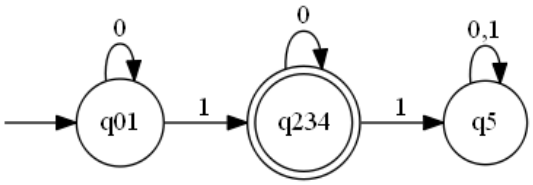
En premier lieu, Il n'y a pas d'état inaccessible. On construit alors la table d'équivalence.

q1					
q2	D	D			
q3	D	D			
q4	D	D			
q5	D	D	D	D	D
	q0	q1	q2	q3	q4

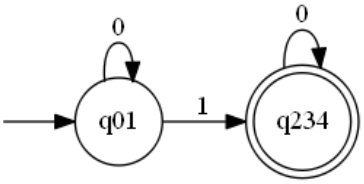
On regroupe les états dans des classes d'équivalence. A partir du tableau, q0 et q1 sont équivalents, et q2 , q3 et q4 sont équivalents. Alors, les classes d'équivalences sont : q_{01}={q0, q1}, q_{234}={q2, q3, q4} et {q5}. La table de transitions de l'automate résultant est la suivante :

	0	1
→ q_{01}	q_{01}	q_{234}
q_{234}	q_{234}	q_5
q_5	q_5	q_5

Le graphe de l'automate obtenu est :



On remarque que l'état q5 n'est pas co-accessible, alors on supprime cet état, et l'automate minimal devient :

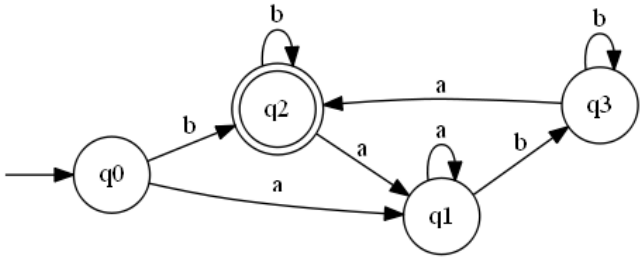


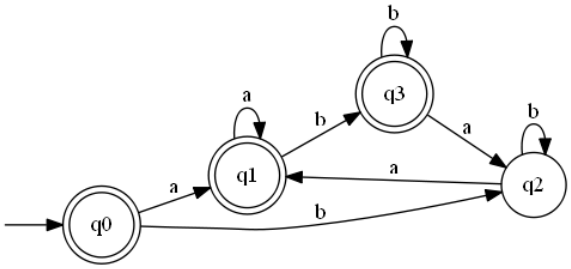
Opérations sur les automates (les langages)
On considère des automates finis déterministes.

Complément
Soit un automate déterministe A reconnaissant le langage L. L'automate complémentaire reconnaissant le langage inverse (c'est-à-dire L_C ou Σ^*-L) peut être obtenu comme suit :

- Compléter l'automate A s'il n'est pas complet.
- L'automate complémentaire peut être obtenu en transformant les états non finaux à des états finaux et vice-versa.

Exemple :





Concaténation

Soit les deux automates déterministes $A_1 = (\Sigma_1, Q_1, \delta_1, q_{10}, F_1)$ et $A_2 = (\Sigma_2, Q_2, \delta_2, q_{20}, F_2)$ reconnaissant les langages L_1 et L_2. Pour construire l'automate A qui reconnaît le langage $L = L_1.L_2$, on peut procéder de la manière suivante:

- L'ensemble des états du nouveau automate A est $Q = Q_1 \cup Q_2$.
- L'état initial de l'automate A est l'état initial du premier automate A_1.
- Les états finaux de l'automate A sont les états finaux du deuxième automate A_2.
- Chaque transition des deux automates A_1 et A_2 est une transition dans l'automate A.
- On ajoute de nouvelles ε-transitions entre chaque état final de A_1 (qui n'est plus final dans le nouvel automate A) et l'état initial de A_2.

Union (Somme des automates)

Soit les deux automates déterministes $A_1 = (\Sigma_1, Q_1, \delta_1, q_{10}, F_1)$ et $A_2 = (\Sigma_2, Q_2, \delta_2, q_{20}, F_2)$ reconnaissant les langages L_1 et L_2. L'union de deux langages permet de construire un langage dont les mots sont issus soit d'un mot du premier langage soit d'un mot du second langage. L'automate fini reconnaissant le langage $L = L_1 \cup L_2$ peut être construit comme suit :

- L'état de départ est l'état composé du couple des deux états initiaux (l'état initial du nouvel automate).
- Pour chaque état créé (p,q) et pour chaque lettre a de l'alphabet $\Sigma = \Sigma_1 \cup \Sigma_2$, ajouter dans la table de transition les états produits calculés de la manière suivante :

$$\delta((p,q),a) = (p',q'), \text{ avec}$$

$$p' = \begin{cases} \delta_1(p,a), & \text{si } \delta_1(p,a) \text{ est définie} \\ \varnothing, & \text{sinon} \end{cases}$$

$$q' = \begin{cases} \delta_2(q,a), & \text{si } \delta_2(p,a) \text{ est définie} \\ \varnothing, & \text{sinon} \end{cases}$$

- Chacun des nouveaux états produits devient un état dans le nouvel automate.
- Recommencer les deux étapes précédentes à partir de chaque nouvel état produit, jusqu'à ce qu'on ne crée plus de nouvel état.
- Tous les états contenant au moins un état final de A_1 ou de A_2 deviennent finaux.
- Pour plus de clarté, renommer éventuellement les états.

Intersection (Produit des automates)

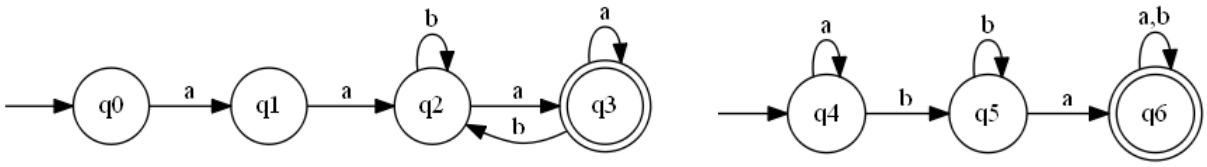
Soit les deux automates déterministes $A_1 = (\Sigma_1, Q_1, \delta_1, q_{10}, F_1)$ et $A_2 = (\Sigma_2, Q_2, \delta_2, q_{20}, F_2)$ reconnaissant les langages L_1 et L_2. L'automate fini reconnaissant le langage $L = L_1 \cap L_2$ peut être construit comme suit :

- L'état de départ est l'état composé du couple des deux états initiaux (l'état initial du nouvel automate).
- Pour chaque état créé (p,q) et pour chaque lettre a de l'alphabet $\Sigma = \Sigma_1 \cap \Sigma_2$, ajouter dans la table de transition les états produits calculés de la manière suivante :

$$\delta\big((p,q),a\big) = (p',q'), \text{ si } \delta_1(p,a) = p' \text{ et } \delta_2(q,a) = q'$$

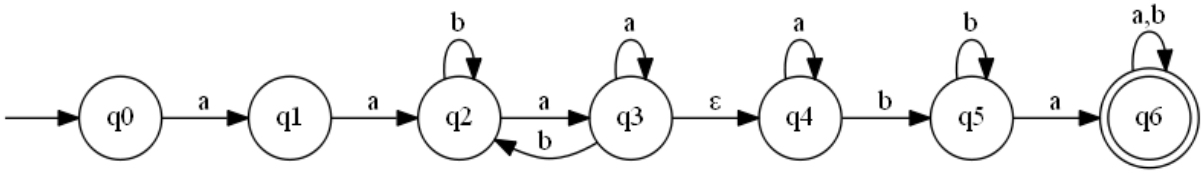
- Chacun des nouveaux états produits devient un état dans le nouvel automate.
- Recommencer les deux étapes précédentes à partir de chaque nouvel état produit, jusqu'à ce qu'on ne crée plus de nouvel état.
- Chaque nouvel état composé d'un état final de A_1 et d'un autre état final de A_2 devient état final dans le nouvel automate.
- Pour plus de clarté, renommer éventuellement les états.

Exemple : Soit A_1 l'automate de gauche et A_2 l'automate de droite. Soient L_1 et L_2 leurs langages respectifs



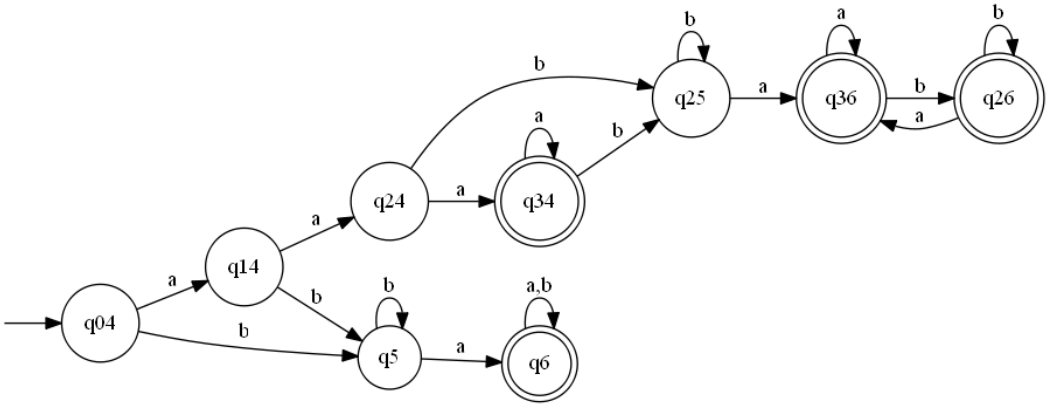
- Construire l'automate reconnaissant le langage $L_1.L_2$.
- Construire l'automate reconnaissant le langage $L_1 \cup L_2$.
- Construire l'automate reconnaissant le langage $L_1 \cap L_2$.

o Automate pour le langage $L_1.L_2$



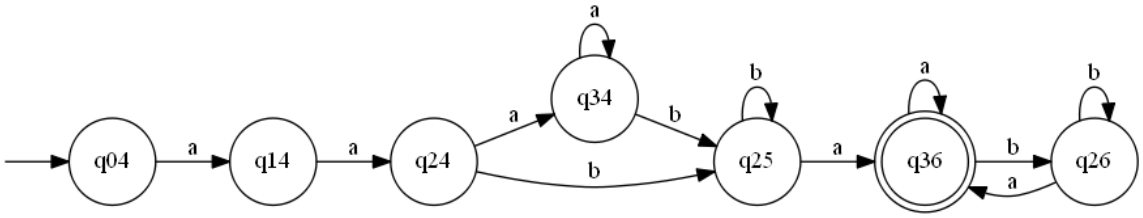
o Automate reconnaissant le langage $L_1 + L_2$

		a	b
→ q_{04}	→ (q_0,q_4)	(q_1,q_4)	q_5
q_{14}	(q_1,q_4)	(q_2,q_4)	q_5
q_5	q_5	q_6	q_5
q_{24}	(q_2,q_4)	(q_3,q_4)	(q_2,q_5)
*q_6	*q_6	q_6	q_6
*q_{34}	*(q_3,q_4)	(q_3,q_4)	(q_2,q_5)
q_{25}	(q_2,q_5)	(q_3,q_6)	(q_2,q_5)
*q_{36}	*(q_3,q_6)	(q_3,q_6)	(q_2,q_6)
*q_{26}	*(q_2,q_6)	(q_3,q_6)	(q_2,q_6)



- Automate reconnaissant le langage $L_1 \cap L_2$

		a	b
$\rightarrow q_{04}$	$\rightarrow (q_0,q_4)$	(q_1,q_4)	-
q_{14}	(q_1,q_4)	(q_2,q_4)	-
q_{24}	(q_2,q_4)	(q_3,q_4)	(q_2,q_5)
q_{34}	(q_3,q_4)	(q_3,q_4)	(q_2,q_5)
q_{25}	(q_2,q_5)	(q_3,q_6)	(q_2,q_5)
$*q_{36}$	$*(q_3,q_6)$	(q_3,q_6)	(q_2,q_6)
q_{26}	(q_2,q_6)	(q_3,q_6)	(q_2,q_6)



Composition d'automates

L'opération de composition des automates permet l'obtention du modèle global d'un système à partir des modèles de ses sous-systèmes. On peut distinguer deux sortes de compositions :

- *Composition synchrone*, qui est effectuée quand les alphabets associés aux automates considérés ont au moins un événement en commun.
- *Composition asynchrone*, qui est réalisée quand les alphabets associés aux automates considérés n'ont aucun événement en commun.

Soient deux automates $A_1 = (\Sigma_1, Q_1, \delta_1, q_{10}, F_1)$ et $A_2 = (\Sigma_2, Q_2, \delta_2, q_{20}, F_2)$. La composition entre A_1 et A_2 est l'automate $A = A_1 \,||\, A_2 = (\Sigma, Q, \delta, q_0, F)$, où :

- $Q = Q_1 \times Q_2$ est l'ensemble d'états,
- $\Sigma = \Sigma_1 \cup \Sigma_2$ est l'alphabet,
- $q_0 = (q_{10}, q_{20})$ est l'état initial,
- $F = F_1 \times F_2$ est l'ensemble des états finaux,
- δ est une relation de $Q \times \Sigma$ qui définit la fonction de transition d'état. Elle associe à chaque état (p,q) et à tout événement $\sigma \in \Sigma$ un état calculé de la façon suivante :
  - Pour $\sigma \notin \Sigma_1 \cap \Sigma_2$

$$\delta\big((p,q),\sigma\big)=\big(p',q\big) \text{ si } \sigma\in\Sigma_1 \text{ et } \delta_1\big(p,\sigma\big)=p',$$
$$\delta\big((p,q),\sigma\big)=\big(p,q'\big) \text{ si } \sigma\in\Sigma_2 \text{ et } \delta_2\big(q,\sigma\big)=q',$$

- Pour $\sigma\in\Sigma_1\cap\Sigma_2$:

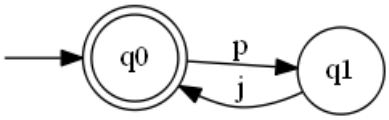
$$\delta\big((p,q),\sigma\big)=\big(p',q'\big) \text{ si } \delta_1\big(p,\sigma\big)=p' \text{ et } \delta_2\big(q,\sigma\big)=q'$$

Remarque : Si un événement $\sigma\in\Sigma_1\cap\Sigma_2$, il doit se produire de manière synchrone dans les deux automates. Par contre, s'il n'appartient qu'à un ensemble, il se produit de manière asynchrone.

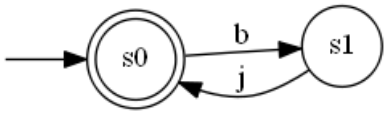
Exemple : Considérons deux automates pour modéliser le fonctionnement d'un distributeur de jetons. On a $\Sigma_1=\{p,j\}$ et $\Sigma_2=\{b,j\}$

Pour A_1, on suppose que le symbole p modélise l'événement « introduction d'une pièce » dans la machine et que j modélise l'événement « libération d'un jeton ». Pour A_2, le symbole b modélise l'événement « pression d'un bouton » et le symbole j a la même signification que dans A_1.

A_1 :



A_2 :



	p	b	j
→ *(q_0,s_0)	(q_1,s_0)	(q_0,s_1)	-
(q_1,s_0)	-	(q_1,s_1)	-
(q_0,s_1)	(q_1,s_1)	-	-
(q_1,s_1)	-	-	(q_0,s_0)