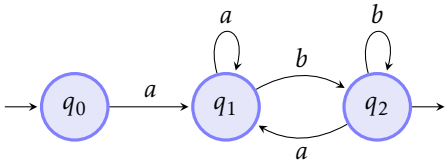
Chapitre 4

Automates

1. Automates finis déterministes

1.1 Introduction

De manière informelle, un automate est une machine abstraite qui peut prendre un nombre fini d'*états*, qui reçoit en entrée un mot écrit sur un alphabet Σ, et qui change d'état à la lecture des lettres de ce dernier.
Certains états sont désignés au préalable comme des états *acceptants* ; à la fin de la lecture du mot passé en entrée l'automate aura changé d'état ; on dira que le mot est *accepté* si l'état final est un état acceptant, et *rejeté* dans le cas contraire.
Considérons à titre d'exemple l'automate fini suivant : les états sont représentés par les nœuds du graphe et les transitions par les arêtes. Ces dernières sont étiquetées par une lettre de l'alphabet $\Sigma = \{a, b\}$ et représentent le changement d'état lié à la lecture du mot en entrée.



L'état initial (ici q_0) est désigné par une flèche entrante ; les états acceptants (ici q_2) sont représentés par une flèche sortante [1].
On se convaincra aisément que pour qu'un mot soit lu et aboutisse à un état acceptant il faut et il suffit qu'il débute par un a et se termine par un b ; on dira que cet automate *reconnait* le langage des mots de $a\Sigma^*b$.

On peut observer qu'un mot débutant par la lettre b ne peut être lu par cet automate, faute d'une transition partant de q_0 et étiquetée par b. On dira que le couple (q_0, b) est un *blocage* de l'automate. Un automate sans blocage est dit *complet* ; il est toujours possible de rendre un automate complet en ajoutant un état inutile qu'on appelle un *puit* (voir figure 1).

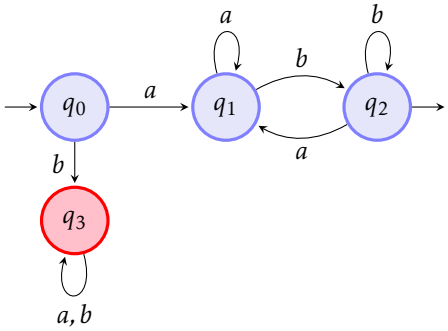
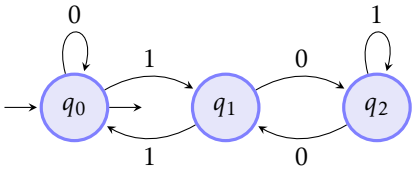


FIGURE 1 – Un automate complet qui reconnait le langage $a\Sigma^*b$; l'état q_3 est un puit.

Le second exemple que nous allons donner est un peu plus complexe. L'alphabet utilisé est l'alphabet binaire $\Sigma = \{0, 1\}$ et le mot lu par l'automate ci-dessous correspond à l'écriture binaire d'un entier n :



1. Certains auteurs représentent les états acceptants avec un double cercle.

Nous allons montrer que cet automate reconnait les entiers n qui sont divisibles par 3. Plus précisément, nous allons montrer par récurrence sur n que l'état final atteint par la lecture de l'entier n est q_0 si $n \equiv 0 \bmod 3$, q_1 si $n \equiv 1 \bmod 3$ et q_2 si $n \equiv 2 \bmod 3$.

- C'est vrai si $n = 0$;
- si $n \geqslant 1$, supposons le résultat acquis jusqu'au rang $n-1$, et posons $n = 2p + r$ avec $r \in \{0, 1\}$.
  Par hypothèse de récurrence appliquée à p, avant la lecture de r l'automate se trouve dans l'état q_i avec $p \equiv i \bmod 3$.
  - Si $i = 0$ alors $n \equiv r \bmod 3$ et si $r = 0$ l'automate reste à l'état q_0, si $r = 1$ l'automate passe à l'état q_1 ;
  - si $i = 1$ alors $n \equiv 2 + r \bmod 3$ et si $r = 0$ l'automate passe à l'état q_2, si $r = 1$ l'automate passe à l'état q_0 ;
  - si $i = 2$ alors $n \equiv 1 + r \bmod 3$ et si $r = 0$ l'automate passe à l'état q_1, si $r = 1$ l'automate reste à l'état q_2.

  Dans les trois cas le résultat est acquis au rang n.

1.2 Définition formelle d'un automate

Définition. — *Un* automate fini [2] déterministe *(DFA, pour* deterministic finite automaton*) est défini par un quintuplet* $A = (\Sigma, Q, q_0, F, \delta)$ *où :*

- Σ *est un alphabet (fini) ;*
- Q *est un ensemble fini dont les éléments sont appelés les* états *de* A ;
- $q_0 \in Q$ *est l'état* initial ;
- $F \subset Q$ *est l'ensemble des états* acceptants *(ou finaux) ;*
- δ *est une application d'une partie de* $Q \times \Sigma$ *dans* Q, *appelée* fonction de transition.

Lorsque δ *est définie sur* $Q \times \Sigma$ *tout entier, l'automate* A *est dit* complet.

Le qualificatif de *déterministe* sera justifié lors de l'introduction des automates *non-déterministes* plus loin dans ce chapitre.

Comme nous l'avons vu en introduction, il est d'usage de représenter un automate par un graphe dont les nœuds sont les états et les arêtes les couples $(q_i, q_j) \in Q^2$ étiquetés par $a \in \Sigma$ tel que $\delta(q_i, a) = q_j$.

- $\Sigma = \{a, b\}$;
- $Q = \{q_0, q_1, q_2\}$;
- $F = \{q_0, q_1\}$;

δ	a	b
q_0	q_1	q_0
q_1	q_2	q_0
q_2	q_2	q_2

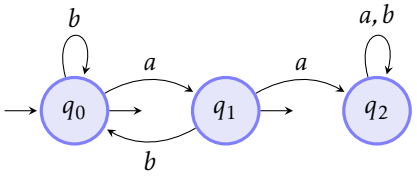


Figure 2 – Un automate fini déterministe complet et sa représentation graphique.

Une transition $\delta(q_i, a) = q_j$ sera représentée par $q_i \xrightarrow{a} q_j$. Un *chemin* dans l'automate A est une suite finie de transitions consécutives $q_0 \xrightarrow{a_1} q_1 \xrightarrow{a_2} \cdots \xrightarrow{a_n} q_n$ débutant par l'état initial q_0. Le mot $a_1 a_2 \cdots a_n$ est appelé l'*étiquette* du chemin.

Définition. — *Un chemin est dit* acceptant *lorsque l'état d'arrivée de celui-ci est un état acceptant ; un mot de* Σ^* *est* reconnu *par l'automate* A *lorsqu'il est l'étiquette d'un chemin acceptant. Le langage* L(A) reconnu *par l'automate* A *est l'ensemble des mots reconnus par* A.

Par exemple, le langage reconnu par l'automate représenté figure 2 est le langage des mots sur $\{a, b\}$ qui ne comportent pas deux a consécutifs. Notons qu'il s'agit d'un langage rationnel dénoté par $(b + ab)^*(\varepsilon + a)$.
En effet, Notons L_i l'ensemble des étiquettes des chemins débutant par l'état q_i et finissant par un état acceptant.
On dispose des relations :

$$L_0 = \varepsilon + bL_0 + aL_1, \qquad L_1 = \varepsilon + bL_0 \quad \text{et} \quad L_2 = \emptyset.$$

On en déduit que $L_0 = \varepsilon + bL_0 + a(\varepsilon + bL_0) = (b + ab)L_0 + (\varepsilon + a)$ et d'après le lemme d'Arden [3], $L_0 = (b + ab)^*(\varepsilon + a)$.

2. Plus exactement *à états finis*.
3. Voir l'exercice 11 du chapitre 3.

Remarque. Le théorème de Kleene qui sera prouvé plus loin établit l'équivalence entre les expressions rationnelles et les automates finis dans le sens où ces deux notions définissent les mêmes langages. Ce résultat est remarquable car il relie deux notions de natures différentes, combinatoire pour les expressions rationnelles et opérationnelles pour les automates.

1.3 Émondage

Nous avons mis en évidence la fonction principale d'un automate : reconnaitre des mots en parcourant un par un ses caractères. Dans le cas d'un automate complet, la complexité de l'algorithme qui en résulte est proportionnelle à la longueur du mot puisqu'aucune transition n'est bloquante. Or ceci peut s'avérer particulièrement inefficace : dans l'exemple de l'automate présenté figure 2, une fois arrivé dans l'état q_2 il est impossible de le quitter et il n'est donc pas nécessaire de poursuivre la lecture du mot (on dit que q_2 est un *puit*). Puisqu'il ne s'agit pas d'un état acceptant, il peut être supprimé sans modifier le langage reconnu par l'automate. On dira que l'automate représenté figure 3 est un automate *équivalent* à celui de la figure 2 car il reconnait le même langage, et le couple (q_1, a) sera appelé un *blocage* car $\delta(q_1, a)$ n'est pas défini.

- $\Sigma = \{a, b\}$;
- $Q = \{q_0, q_1\}$;
- $F = \{q_0, q_1\}$;

δ	a	b
q_0	q_1	q_0
q_1	–	q_0

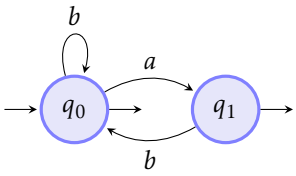


Figure 3 – Un automate incomplet équivalent à celui de la figure 2.

L'état q_2 de l'automate présenté figure 2 a pu être supprimé car il n'existe pas de chemin menant de q_2 à un état acceptant. Plus généralement, on dit d'un état q qu'il est :

- *accessible* lorsqu'il existe un chemin menant de l'état initial q_0 à q ;
- *co-accessible* lorsqu'il existe un chemin menant de q à un état acceptant.

Un état accessible et co-accessible est dit *utile* ; sachant que le chemin décrit par un mot reconnu ne passe que par des états utiles, on ne change pas le langage reconnu en supprimant tous les états qui ne le sont pas ainsi que les transitions impliquant ces états. L'automate obtenu, qui ne possède plus que des états utiles, est dit *émondé*.

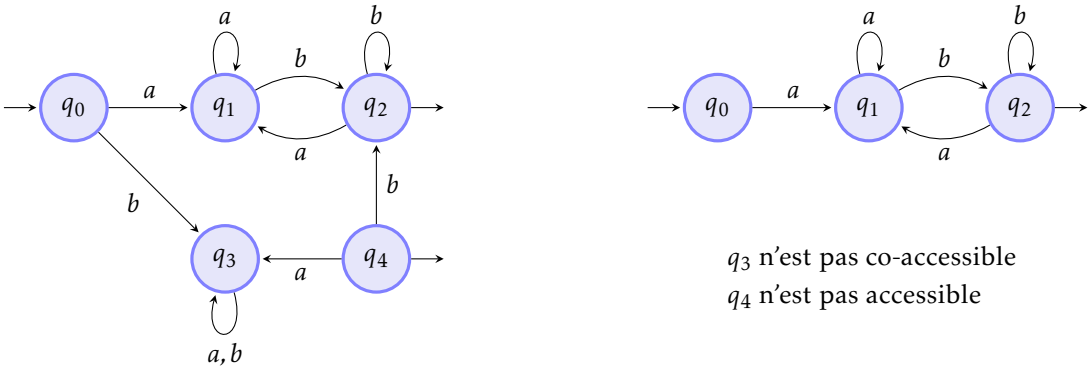


Figure 4 – Un automate complet et son émondage. Tous deux reconnaissent le langage dénoté par $a(a+b)^*b$.

1.4 Mise en œuvre pratique

Pour modéliser un automate en Caml il est pratique d'utiliser le type *char* pour représenter l'alphabet Σ, le type *int* pour l'ensemble des états et d'utiliser un dictionnaire pour énumérer la liste des transitions possibles. Pour simplifier, la liste des transitions possibles sera représentée par le type *(int * char) * int list* et on utilisera la fonction **assoc** pour associer au couple (q, a) l'état $\delta(q, a)$ [4].

4. la fonction **assoc** est de type *'a –> ('a * 'b) list –> 'b* ; appliquée à un élément a et une liste ℓ cette fonction renvoie la première valeur b telle que $(a, b) \in \ell$ et déclenche l'exception **Not_found** s'il n'en existe pas.

Ceci conduit à définir le type :

```
type dfa = {Start: int ;
            Accept: int list ;
            Delta: ((int * char) * int) list} ;;
```

Par exemple, l'automate émondé représenté figure 4 sera défini par :

```
let a = {Start = 0; Accept = [2]; Delta = [ (0, 'a'), 1; (1, 'a'), 1; (1, 'b'), 2;
                                            (2, 'a'), 1; (2, 'b'), 2 ]} ;;
```

La fonction suivante détermine l'état final de l'automate après parcours du chemin étiqueté par un mot passé en paramètre (et déclenche l'exception **Not_found** en cas de blocage) :

```
let chemin a m =
  let rec aux q = function
    | k when k = string_length m -> q
    | k                          -> aux (assoc (q, m.[k]) a.Delta) (k+1)
  in aux a.Start 0 ;;
```

Enfin, la fonction qui suit détermine si un mot est reconnu ou pas :

```
let reconnu a m =
  try mem (chemin a m) a.Accept
  with Not_found -> false ;;
```

2. Automates non-déterministes

Dans cette section nous allons généraliser la notion d'automate fini en utilisant plusieurs états initiaux et en autorisant deux transitions sortantes d'un même état q à porter la même étiquette. Nous constaterons que cette généralisation n'augmente pas l'expressivité du modèle : les langages reconnus sont les mêmes que pour les automates déterministes ; cependant le nombre d'états d'un automate non-déterministe peut être notablement inférieur à l'automate déterministe équivalent.

2.1 Définition

Définition. — *Un* automate fini non-déterministe *(NFA,* nondeterministic finite automaton*) est défini par un quintuplet* $A = (\Sigma, Q, I, F, \delta)$ *où :*

- Σ *est un alphabet (fini) ;*
- Q *est un ensemble fini dont les éléments sont appelés les* états *de* A *;*
- $I \subset Q$ *est l'ensemble des états* initiaux *;*
- $F \subset Q$ *est l'ensemble des états* acceptants *(ou finaux) ;*
- δ *est une application d'une partie de* $Q \times \Sigma$ *dans* $\mathscr{P}(Q)$*, appelée* fonction de transition.

On notera que la notion de blocage n'a plus vraiment de sens puisque $\delta(q, a)$ peut prendre la valeur $\emptyset$.

Une *transition* est un triplet (q_i, a, q_j) tel que $q_j \in \delta(q_i, a)$; celle-ci est représentée par $q_i \xrightarrow{a} q_j$. Un *chemin* est une suite finie de transitions consécutives $q_0 \xrightarrow{a_1} q_1 \xrightarrow{a_2} \cdots \xrightarrow{a_n} q_n$.

Définition. — *Un mot de* Σ^* *est* reconnu *par l'automate* A *s'il étiquette un chemin menant d'un état initial à un état acceptant. Le langage des mots reconnus par l'automate* A *est noté* $L(A)$.

Autrement dit, pour qu'un mot m soit reconnu il suffit qu'il existe *au moins* un chemin étiqueté par m menant d'un état initial à un état acceptant, ce qui nécessite de tester tous les chemins partant d'un état initial et étiquetés par m avant de pouvoir affirmer que ce dernier n'est pas reconnaissable.

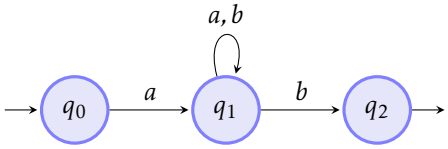


FIGURE 5 – Un automate non-déterministe qui reconnait le langage dénoté par $a(a+b)^*b$.

2.2 Déterminisation

La propriété essentielle que nous allons établir est que tout automate non-déterministe est équivalent (dans le sens où il reconnait le même langage) à un automate déterministe. La preuve que nous allons établir est couramment appelée *construction par sous-ensembles*.

Considérons donc un automate non-déterministe $A = (\Sigma, Q, I, F, \delta)$ et posons :

$$A' = (\Sigma, \mathscr{P}(Q), I, F', \delta') \quad \text{avec} \quad F' = \left\{P \in \mathscr{P}(Q) \;\middle|\; P \cap F \neq \emptyset\right\} \quad \text{et} \quad \forall (P,a) \in \mathscr{P}(Q) \times \Sigma,\ \delta'(P,a) = \bigcup_{q \in P} \delta(q,a).$$

Si A est un automate non-déterministe à n états, l'automate A′ est un automate déterministe à 2^n états. Pour bien comprendre cette construction, calculons A′ lorsque A est l'automate représenté figure 5. A′ possède $2^3 = 8$ états : $\emptyset$, $\{q_0\}$, $\{q_1\}$, $\{q_2\}$, $\{q_0,q_1\}$, $\{q_1,q_2\}$, $\{q_0,q_2\}$, $\{q_0,q_1,q_2\}$; son état initial est $\{q_0\}$ et ses états acceptants tous ceux qui contiennent q_2, à savoir $\{q_2\}$, $\{q_1,q_2\}$, $\{q_0,q_1,q_2\}$. Enfin, le tableau des transitions de A′ est le suivant :

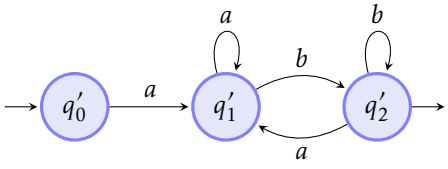
δ'	a	b
$\emptyset$	$\emptyset$	$\emptyset$
$\{q_0\}$	$\{q_1\}$	$\emptyset$
$\{q_1\}$	$\{q_1\}$	$\{q_1,q_2\}$
$\{q_2\}$	$\emptyset$	$\emptyset$

δ'	a	b
$\{q_0,q_1\}$	$\{q_1\}$	$\{q_1,q_2\}$
$\{q_1,q_2\}$	$\{q_1\}$	$\{q_1,q_2\}$
$\{q_0,q_2\}$	$\{q_1\}$	$\emptyset$
$\{q_0,q_1,q_2\}$	$\{q_1\}$	$\{q_1,q_2\}$

Pour éviter d'avoir à représenter un automate trop volumineux on ne représente que les états accessibles : pour ce faire on commence par remplir le tableau des transitions avec l'état initial et on n'ajoute que les états qui apparaissent comme résultats des transitions précédentes. On ne représente pas non plus l'état $\emptyset$ qui ne peut être co-accessible (en revanche d'autres états non co-accessibles peuvent subsister). Dans le cas qui nous intéresse cela conduit au tableau simplifié suivant :

δ'	a	b
$\{q_0\}$	$\{q_1\}$	–
$\{q_1\}$	$\{q_1\}$	$\{q_1,q_2\}$
$\{q_1,q_2\}$	$\{q_1\}$	$\{q_1,q_2\}$

En posant $q'_0 = \{q_0\}$, $q'_1 = \{q_1\}$ et $q'_2 = \{q_1,q_2\}$ l'automate déterminisé (et partiellement émondé) est donc :



THÉORÈME. — *Les automates* A *et* A′ *reconnaissent le même langage.*

Preuve. On montre par récurrence sur $|u|$ que pour tout mot $u \in \Sigma^*$, il existe dans A un chemin étiqueté par u menant à un état q si et seulement s'il existe dans A′ un chemin étiqueté par u menant à un état P contenant q.

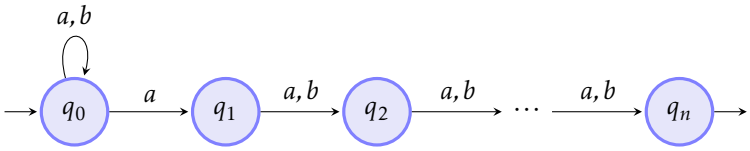
- Dans A tout chemin étiqueté par ε mène à un élément de I ; dans A′ tout chemin étiqueté par ε mène à l'état I. Le résultat énoncé est donc vrai pour le mot ε.
- Si $u \neq \varepsilon$, supposons le résultat acquis pour tout mot de longueur strictement inférieure, et posons $u = a_1 a_2 \cdots a_n$.

  Considérons un chemin $q_0 \xrightarrow{a_1} q_1 \xrightarrow{a_2} \cdots \xrightarrow{a_n} q_n$ dans A. Par hypothèse de récurrence il existe dans A′ un chemin $I \xrightarrow{a_1} P_1 \xrightarrow{a_2} \cdots \xrightarrow{a_{n-1}} P_{n-1}$ tel que $q_{n-1} \in P_{n-1}$. Or $q_n \in \delta(q_{n-1}, a_n)$ donc $q_n \in \delta'(P_{n-1}, a_n)$. En posant $P_n = \delta'(P_{n-1}, a_n)$ on établit un chemin $I \xrightarrow{a_1} P_1 \xrightarrow{a_2} \cdots \xrightarrow{a_n} P_n$ tel que $q_n \in P_n$.

Réciproquement, considérons un chemin $I \xrightarrow{a_1} P_1 \xrightarrow{a_2} \cdots \xrightarrow{a_n} P_n$ dans A, et considérons un élément $q_n \in P_n$. Il existe donc un état $q_{n-1} \in P_{n-1}$ tel que $q_n \in \delta(q_{n-1}, a_n)$. Or par hypothèse de récurrence il existe un chemin $q_0 \xrightarrow{a_1} q_1 \xrightarrow{a_2} \cdots \xrightarrow{a_{n-1}} q_{n-1}$ dans A, ce qui prouve l'existence d'un chemin $q_0 \xrightarrow{a_1} q_1 \xrightarrow{a_2} \cdots \xrightarrow{a_n} q_n$.

Sachant que $F' = \left\{P \in \mathscr{P}(Q) \;\middle|\; P \cap F \neq \emptyset\right\}$ ceci prouve qu'un mot est reconnu par A si et seulement s'il est reconnu par A'. □

Remarque. Le résultat précédent prouve que les automates non-déterministes ne sont pas plus « riches » que les automates déterministes puisqu'ils reconnaissent exactement les mêmes langages. En outre ils ne sont pas très efficaces pour le traitement effectif des données : pour un mot donné, il faut explorer tous les chemins étiquetés par celui-ci, à partir de tous les états initiaux, avant de pouvoir affirmer que ce mot n'est pas reconnu alors que pour un automate déterministe il y a au plus un chemin étiqueté pour un mot donné.
Cependant ils sont parfois plus simples à construire à partir d'une caractérisation donnée d'un langage et seront pour cette raison précieux dans certaines preuves à venir. En outre, ils fournissent des automates ayant souvent un nombre plus réduit d'états. La démarche que nous suivrons plus loin consistera à trouver un automate non-déterministe reconnaissant un langage donné, à le déterminiser puis à émonder ce dernier avec pour objectif d'obtenir un automate déterministe ayant un nombre le plus réduit possible d'états.
Il faut cependant noter qu'il sera parfois tout bonnement impossible d'obtenir un automate déterministe ayant un nombre d'états du même ordre de grandeur que pour l'automate non déterministe équivalent. Considérons en effet le langage dénoté par $(a+b)^*a(a+b)^{n-1}$. Il est facile d'obtenir un automate non-déterministe à $n+1$ états qui le reconnait :



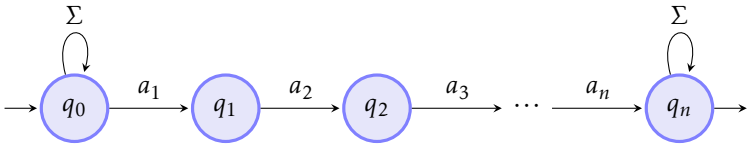
En revanche nous allons montrer que tout automate déterministe qui reconnait L possède au moins 2^n états.

Preuve. Considérons un tel automate A et notons q_0 son état initial. À tout mot u de n lettres on peut associer l'état $q(u)$ auquel aboutit le chemin étiqueté par u. On définit ainsi une application $q : \{a, b\}^n \to Q$ et nous allons montrer que cette application est injective.
Pour cela on considère deux mots distincts u et v tels que $q(u) = q(v)$ ainsi que le plus long suffixe w commun à u et à v. Sans perte de généralité on peut poser $u = u'aw$ et $v = v'bw$. Le mot w est de longueur inférieure ou égale à $n-1$; complétons-le pour former un mot ww' de longueur $n-1$. Dans ces conditions le mot $uw' = u'aww'$ est reconnu par A mais pas $vw' = v'bww'$. Cependant puisque A est déterministe et $q(u) = q(v)$ les chemins étiquetés par uw' et vw' doivent mener au même état (ou être tous deux bloquants) ce qui est absurde.
q est donc bien injectif et A possède au moins 2^n états. □

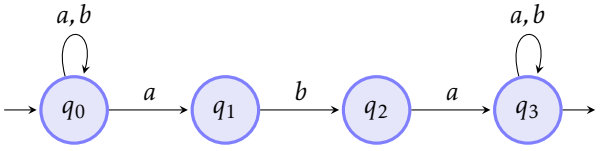
2.3 Recherche d'un mot dans un texte

Nous allons maintenant illustrer l'intérêt que peut présenter la formalisation d'un problème par des automates en abordant le problème de la recherche d'un mot dans un texte.

Si $u \in \Sigma^*$ est un mot donné, on souhaite obtenir un automate qui caractérise la présence de u dans un texte, autrement dit qui reconnait le langage $\Sigma^* u \Sigma^*$. Ce problème est aisément résolu à l'aide d'un automate non-déterministe : si $u = a_1 a_2 \cdots a_n$ il suffit de considérer l'automate :



Cependant, si on veut effectivement utiliser ce formalisme pour traiter un texte, il est préférable de disposer d'un automate déterministe ; on calcule donc l'automate déterminisé de l'automate ci-dessus. La figure 6 présente l'automate non-déterministe associé à la recherche du mot *aba* sur l'alphabet $\{a, b\}$ ainsi que sa déterminisation.



δ'	a	b
$\{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$
$\{q_0, q_1\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$
$\{q_0, q_2\}$	$\{q_0, q_1, q_3\}$	$\{q_0\}$
$\{q_0, q_1, q_3\}$	$\{q_0, q_1, q_3\}$	$\{q_0, q_2, q_3\}$
$\{q_0, q_2, q_3\}$	$\{q_0, q_1, q_3\}$	$\{q_0, q_3\}$
$\{q_0, q_3\}$	$\{q_0, q_1, q_3\}$	$\{q_0, q_3\}$

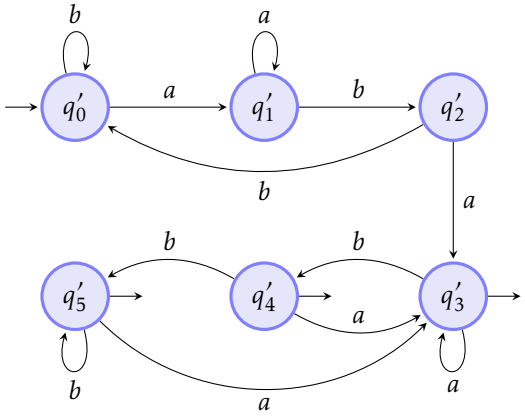
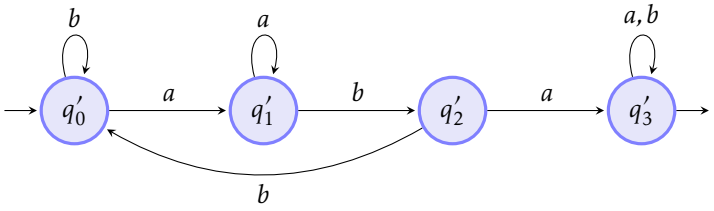


FIGURE 6 – L'automate non-déterministe qui reconnait le langage $\Sigma^* aba \Sigma^*$ et sa déterminisation.

L'automate déterministe obtenu est émondé et possède 6 états mais il n'est pas optimal : à l'évidence les états q'_4 et q'_5 ne sont pas nécessaires et il peut être réduit à l'automate à quatre états suivant :



• Algorithme KMP [5]

Il existe un moyen d'obtenir mécaniquement un automate déterministe ayant un nombre plus réduit d'états en considérant l'ensemble $\mathrm{P}(u)$ des préfixes du mot u et si v est un mot en notant $s(v)$ le plus long suffixe de v qui soit dans $\mathrm{P}(u)$.
Nous allons considérer l'automate déterministe $\mathrm{A} = (\Sigma, \mathrm{P}(u), \{\varepsilon\}, \{u\}, \delta)$ où la fonction de transition est définie par :

$$\forall p \in \mathrm{P}(u), \quad \forall x \in \Sigma, \qquad \delta(p, x) = s(px)$$

La figure 7 représente l'automate ainsi construit à partir du mot $u = aba$.

δ	a	b
ε	a	ε
a	a	ab
ab	aba	ε
aba	a	ab

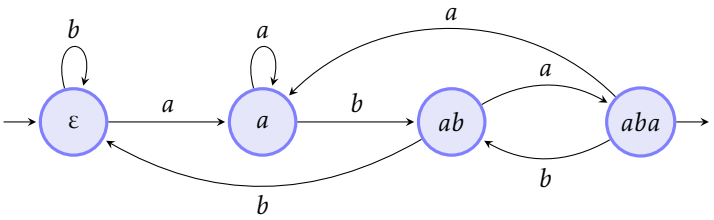


FIGURE 7 – Un automate déterministe qui reconnait le langage $\Sigma^* aba$.

THÉORÈME. — *L'automate* A *reconnait le langage* $\Sigma^* u$.

5. D'après le nom de leurs inventeurs, KNUTH, MORRIS et PRATT.

Preuve. Si $p \in \mathrm{P}(u)$ et $v \in \Sigma^*$ nous allons montrer par récurrence sur $|v|$ que le chemin qui part de l'état p et qui est étiqueté par v mène à l'état $s(pv)$.

- Si $v = \varepsilon$ on a $s(p) = p$ puisque $p \in \mathrm{P}(u)$.
- Si v n'est pas le mot vide, supposons le résultat acquis pour tout mot de longueur inférieure et notons $v = v'a$. Par hypothèse de récurrence le chemin qui part de p étiqueté par v' mène à l'état $s(pv')$. Celui étiqueté par v mène donc à l'état $\delta(s(pv'), a) = s(s(pv')a)$. Il s'agit donc de prouver que $s(s(pv')a) = s(pv)$.

  Posons $w = s(s(pv')a)$ et $w' = s(pv')$. w est suffixe de $w'a$ et w' suffixe de pv' donc w est suffixe de $pv'a = pv$. De plus $w \in \mathrm{P}(u)$ donc w est suffixe de $s(pv)$ (plus long suffixe de pv qui soit dans $\mathrm{P}(u)$).
  Si $s(pv) = \varepsilon$ alors $w = \varepsilon$. Si $s(pv) \neq \varepsilon$ on peut poser $s(pv) = xa$ puisque $v = v'a$. Alors $x \in \mathrm{P}(u)$ et x est suffixe de pv' donc x est suffixe de $s(pv') = w'$ et xa suffixe de $w'a$. Puisque $xa \in \mathrm{P}(u)$ alors xa est suffixe de $s(w'a) = w$. Dans les deux cas nous avons prouvé que $s(pv)$ est suffixe de w donc en définitive $w = s(pv)$.

Ainsi le chemin qui part de l'état ε et étiqueté par un mot v mène à l'état $s(v)$, et $s(v) = u \iff v \in \Sigma^* u$. □

Corollaire. — *Pour obtenir un automate qui reconnait le langage $\Sigma^* u \Sigma^*$ il suffit de considérer l'automate* A *et de transformer l'état u en puit.*

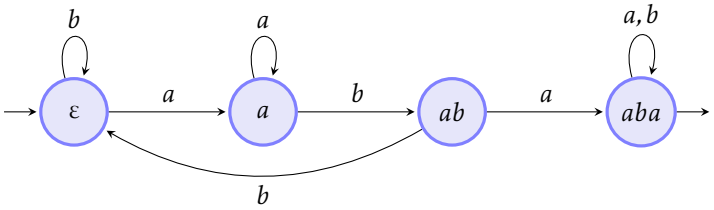


Figure 8 – Un automate déterministe qui reconnait le langage $\Sigma^* aba \Sigma^*$.

3. Automates finis et langages rationnels

Il est maintenant temps d'aborder le théorème essentiel de ce chapitre, qui affirme l'équivalence entre langages rationnels et langages reconnaissables par un automate. Il s'agit du :

Théorème (Kleene). — *Un langage* L *sur un alphabet Σ est rationnel si et seulement s'il existe un automate fini* A *tel que* $\mathrm{L}(\mathrm{A}) = \mathrm{L}$.

Dans un premier temps nous montrerons que tout langage rationnel est reconnaissable en décrivant un algorithme construisant explicitement un automate (l'automate de Glushkov) associé à une expression rationnelle. Moins utile en pratique, la réciproque, à savoir la détermination d'une expression rationnelle à partir d'un automate, nous servira essentiellement à justifier l'équivalence dans le théorème de Kleene. Enfin, quelques propriétés des langages reconnaissables (en particulier de clôture) seront établies pour être étendues aux langages rationnels.

3.1 L'algorithme de Berry-Sethi

Définition. — *Un automate déterministe* $\mathrm{A} = (\Sigma, \mathrm{Q}, q_0, \mathrm{F}, \delta)$ *est dit* local *lorsque pour toute lettre x il existe un état $q \in \mathrm{Q}$ tel que pour tout $q' \in \mathrm{Q}$, $\delta(q', x)$ est un blocage ou $\delta(q', x) = q$.*
Il est dit standard *lorsqu'il n'existe pas de lettre x et d'état q tel que $\delta(q, x) = q_0$.*

Autrement dit, un automate est local lorsque pour chaque lettre x toutes les transitions étiquetées par x arrivent dans un même état, et il est standard lorsqu'il n'existe pas de transition aboutissant à l'état initial.

Théorème. — *Si* L *est un langage local, le langage* $\mathrm{L} \setminus \{\varepsilon\}$ *est reconnaissable par un automate local standard.*

Preuve. Notons $\mathrm{P} = \mathrm{P}(\mathrm{L})$, $\mathrm{S} = \mathrm{S}(\mathrm{L})$, $\mathrm{F} = \mathrm{F}(\mathrm{L})$ [6] et considérons l'automate $\mathrm{A} = (\Sigma, \Sigma \cup \{\varepsilon\}, \varepsilon, \mathrm{S}, \delta)$ avec :

$$\forall x \in \mathrm{P},\ \delta(\varepsilon, x) = x \qquad \text{et} \qquad \forall xy \in \mathrm{F},\ \delta(x, y) = y.$$

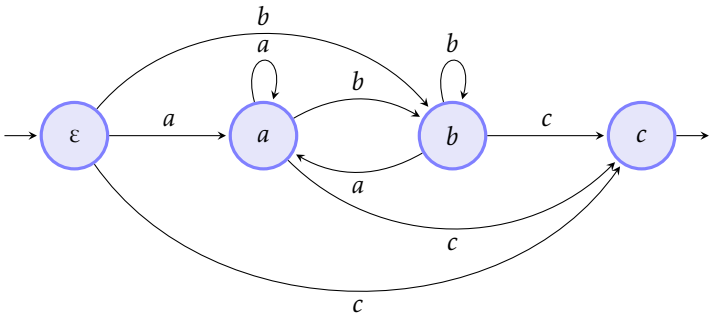
6. Ces notations sont celles du chapitre précédent.

Par construction, le mot $u = a_1 a_2 \cdots a_n$ est reconnu par A si et seulement si $a_1 \in P$, $a_i a_{i+1} \in F$ pour $i \in [\![1, n-1]\!]$ et $a_n \in S$. □

Remarque. Si le langage local L contient le mot vide il suffit d'ajouter l'état ε à l'ensemble des états acceptant pour obtenir un automate qui reconnait L.

Corollaire. — *Tout langage dénoté par une expression rationnelle linéaire est reconnaissable.*

À titre l'exemple, considérons l'expression rationnelle linéaire $(a+b)^*c$. Nous avons vu dans le chapitre précédent les algorithmes de calcul des ensembles P, S et F ; appliqués à cette expression on obtient $P = \{a, b, c\}$, $S = \{c\}$ et $F = \{aa, ab, ba, bb, ac, bc\}$. L'automate local reconnaissant le langage dénoté par cette expression est donc :



• Construction de l'automate de Glushkov

Considérons maintenant une expression rationnelle quelconque e. Nous avons vu au chapitre précédent que si le langage dénoté par e est non vide il existe une expression rationnelle équivalente e' ne contenant pas le symbole ∅. Nous avons aussi démontré l'existence d'une expression rationnelle e'' ne contant pas le symbole ε telle que e'' soit équivalente à ε, e'' ou $\varepsilon + e''$. Il est facile de construire un automate reconnaissant le langage dénoté par $\varepsilon + e''$ à partir d'un automate reconnaissant le langage dénoté par e'' : il suffit d'ajouter q_0 aux états acceptants. *On suppose donc désormais que e est une expression rationnelle ne comportant ni le symbole ∅, ni le symbole ε.*

Linéarisation d'une expression rationnelle par marquage
Notons n le nombre de lettres (*non nécessairement distinctes*) de e et ordonnons ces dernières par ordre croissant d'apparition dans e. Remplaçons dans e chaque lettre par le caractère c_k où k désigne son rang d'apparition. On obtient une nouvelle expression rationnelle sur l'alphabet $\Sigma' = \{c_1, c_2, \ldots, c_n\}$, expression qui est linéaire.
Par exemple, à l'expression $e = (a+b)(a^* + ba^* + b^*)^*$ va être associée l'expression rationnelle linéaire $e' = (c_1 + c_2)(c_3^* + c_4 c_5^* + c_6^*)^*$.
Pour retrouver l'expression rationnelle initiale à partir de l'expression linéarisée il suffit de connaître la fonction de marquage $\mu : \{c_1, c_2, \ldots, c_n\} \to \Sigma$ précisant par quel caractère de Σ doit être remplacé le caractère c_k.
Dans le cas de l'exemple ci-dessus on a $\mu(c_1) = \mu(c_3) = \mu(c_5) = a$ et $\mu(c_2) = \mu(c_4) = \mu(c_6) = b$.

Théorème. — *Tout langage dénoté par une expression rationnelle sans symbole ∅ ni ε est reconnaissable par un automate fini.*

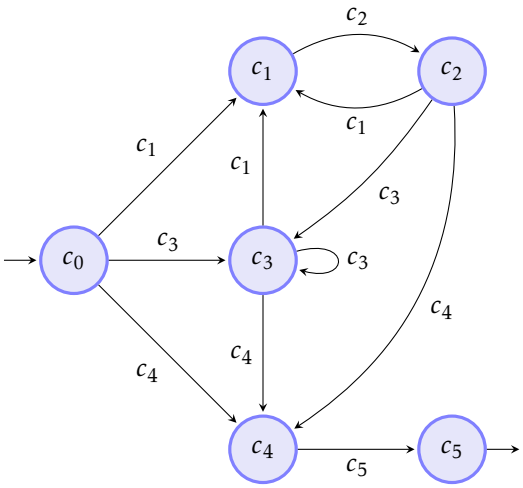
Preuve. Soit e une expression rationnelle sans ∅ ni ε, et $A = (\{c_1, \ldots, c_n\}, Q, q_0, F, \delta)$ l'automate standard local associé à la linéarisation de e. On remplace dans A toutes les transitions étiquetées par le caractère c_i par une transition étiquetée par $\mu(c_i)$; autrement dit on considère l'automate $A' = (\Sigma, Q, q_0, S, \delta')$ obtenu en posant $\delta(q_i, c) = \delta(q_i \mu(c))$. Alors A′ est un automate standard (en général non déterministe) qui reconnait le langage dénoté par e. □

Remarque. Compte tenu de la remarque faite au préalable, ceci prouve le sens direct du théorème de Kleene. La procédure que nous venons de décrire :

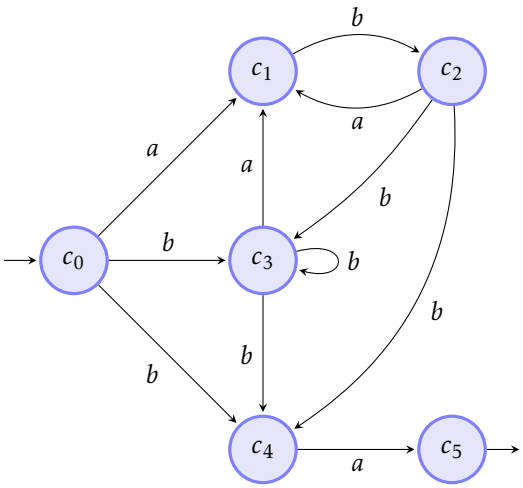
- linéarisation de l'expression ;
- calcul des ensembles P, S et F définissant le langage local associé ;
- construction de l'automate local ;
- suppression des marques utilisées pour la linéarisation

porte le nom d' *algorithme de* Berry-Sethi et l'automate obtenu s'appelle l'*automate de* Glushkov de l'expression rationnelle. Ce dernier est en général non déterministe et possède $|e|+1$ états ; il peut être rendu déterministe par déterminisation.

Exemple. Considérons l'expression rationnelle $e = (ab+b)^*ba$. Sa linéarisation est $e' = (c_1c_2+c_3)^*c_4c_5$. On calcule $P = \{c_1, c_3, c_4\}$, $S = \{c_5\}$ et $F = \{c_1c_2, c_2c_1, c_2c_3, c_2c_4, c_3c_1, c_3c_3, c_3c_4, c_4c_5\}$ donc l'automate local standard associé à e' est :

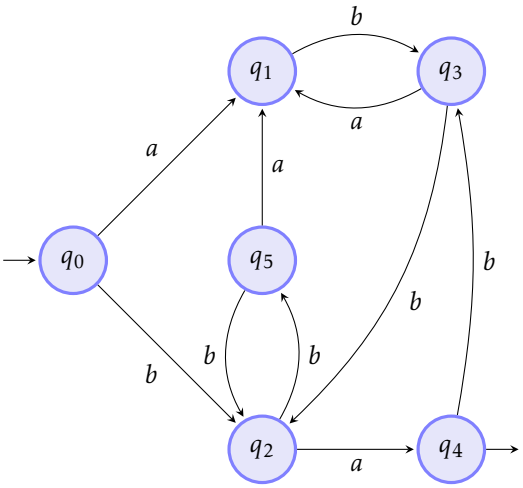


Il reste à supprimer le marquage pour obtenir l'automate de Glushkov associé à e :



On peut finalement le déterminiser :

δ'	a	b
$\{c_0\}$	$\{c_1\}$	$\{c_3, c_4\}$
$\{c_1\}$	–	$\{c_2\}$
$\{c_3, c_4\}$	$\{c_1, c_5\}$	$\{c_3\}$
$\{c_2\}$	$\{c_1\}$	$\{c_3, c_4\}$
$\{c_1, c_5\}$	–	$\{c_2\}$
$\{c_3\}$	$\{c_1\}$	$\{c_3, c_4\}$



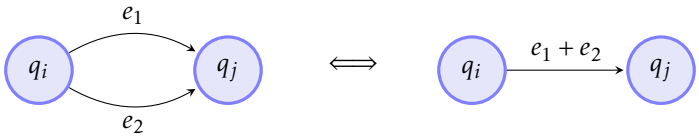
3.2 Des automates aux expressions rationnelles

Pour montrer que tout langage reconnu peut être dénoté par une expression rationnelle nous allons généraliser la définition des automates en s'autorisant l'étiquetage d'une transition par une expression rationnelle. Plus précisément :

Définition. — *On appelle* automate généralisé *un automate* $A = (\mathrm{Rat}(\Sigma), Q, I, F, \delta)$ *dont les états et les transitions sont en nombres finis et dont les transitions sont étiquetées par des expressions rationnelles.*

Avec un automate généralisé, un mot u est reconnu s'il existe un chemin $q_0 \xrightarrow{e_1} q_1 \xrightarrow{e_2} \cdots \xrightarrow{e_n} q_n$ menant d'un état initial à un état acceptant tel que u appartienne au langage dénoté par $e_1 e_2 \cdots e_n$.

Remarque. La possibilité d'étiqueter une transition par ε permet de supposer qu'un automate généralisé ne possède qu'un seul état initial et un seul état acceptant : si tel n'est pas le cas, il suffit d'ajouter à Q deux états i et f ainsi que les transitions : $\delta(i, \varepsilon) = q$ pour tout $q \in I$ et $\delta(q, \varepsilon) = f$ pour tout $q \in F$. L'automate $(\Sigma, Q \cup \{i, f\}, \{i\}, \{f\}, \delta)$ est alors équivalent à A. Remarquons en outre qu'avec cette construction il n'existe pas de transition vers i ni de transition à partir de f.
Notons aussi qu'on peut supposer qu'entre deux états il n'existe qu'au plus une transition. En effet, deux transitions $\delta(q_i, e_1) = q_j$ et $\delta(q_i, e_2) = q_j$ peuvent être remplacées par une seule : $\delta(q_i, e_1 + e_2) = q_j$.

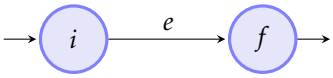


Le résultat suivant montre que le pouvoir expressif des automates n'est pas augmenté par cette généralisation :

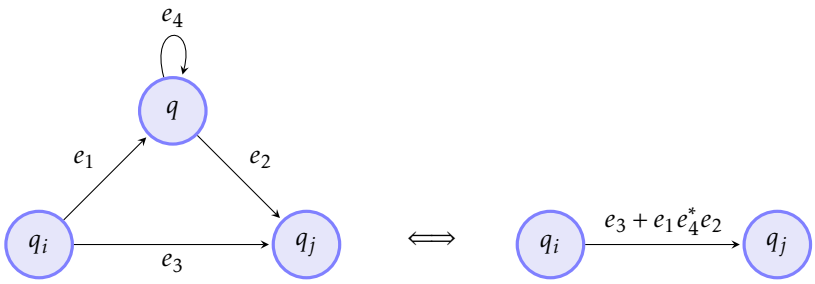
Théorème. — *Tout langage reconnu par un automate généralisé est rationnel.*

Preuve. Considérons un automate généralisé A possédant un unique état initial i, un unique état final f et au plus une transition par couple d'états, et raisonnons par récurrence sur le nombre d'états |Q|.

- Si |Q| = 2 l'automate A est de la forme suivante :



  et le langage reconnu par A est dénoté par e donc est rationnel.
- Si |Q| > 2, supposons le résultat acquis au rang précédent et considérons un état $q \in Q \setminus \{i, f\}$. Pour chaque couple d'états (q_i, q_j) tel que $q_i \neq q$ et $q_j \neq q$ on effectue la transformation suivante :

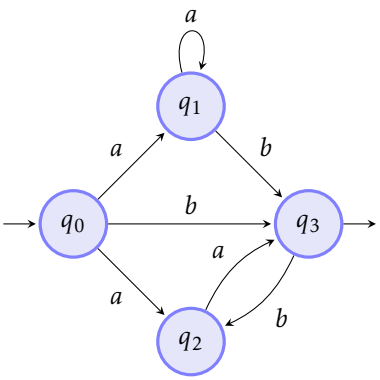


  (avec le cas échéant e_1, e_2, e_3, e_4 égal à $\emptyset$ si la transition correspondante n'existe pas).
  Ceci permet d'éliminer totalement l'état q et donc d'obtenir un automate généralisé équivalent ayant un état de moins, à qui on peut appliquer l'hypothèse de récurrence pour conclure.

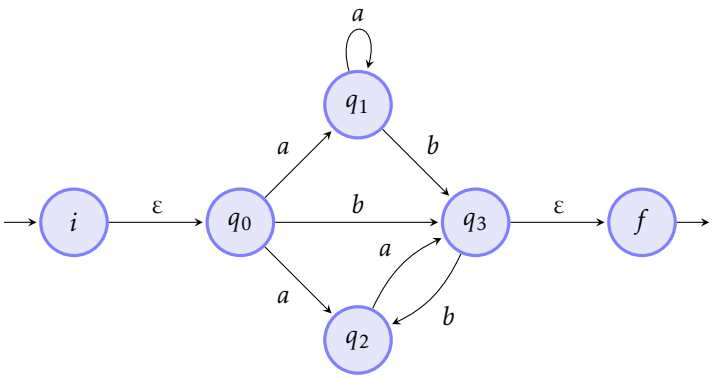
□

Algorithme de Brzozowski et McCluskey
Appelé aussi algorithme d'*élimination des états*, il consiste à appliquer la démarche décrite dans la preuve ci-dessus pour éliminer un par un les différents états d'un automate jusqu'à ne plus obtenir qu'un automate généralisé à deux états qui fournira une expression rationnelle dénotant le langage reconnu par l'automate initial.

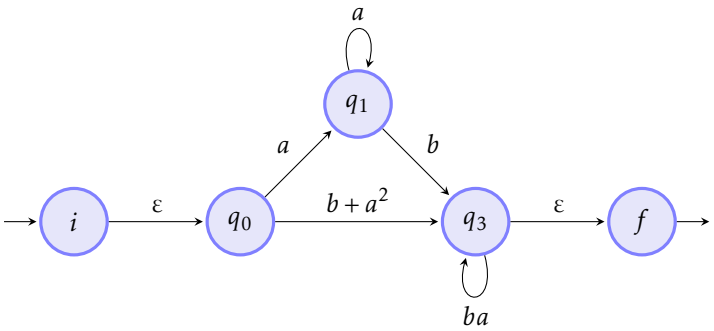
Exemple. Considérons l'automate suivant :



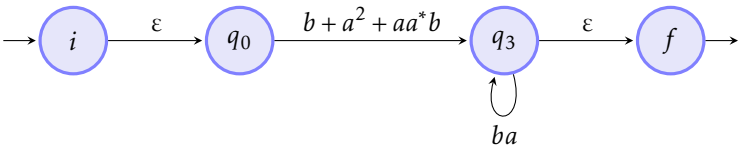
On commence par ajouter un état initial et un état final :



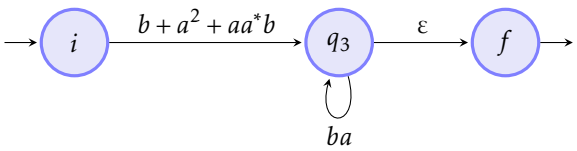
On élimine l'état q_2 :



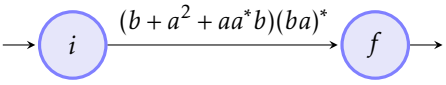
On élimine l'état q_1 :



On élimine l'état q_0 :



On élimine enfin l'état q_3 :



Nous avons montré que l'automate reconnait le langage dénoté par $(b+a^2+aa^*b)(ba)^*$. Évidemment, l'expression obtenue dépend de l'ordre dans lequel les éliminations ont été réalisées.

3.3 Propriétés des langages reconnaissables

Nous allons maintenant nous attacher à prouver des propriétés de stabilité des langages reconnus par un automate : stabilité par complémentation et par intersection.

Théorème. — *Les langages reconnaissables sont clos par complémentation.*

Preuve. Soit L un langage reconnu par un automate déterministe *complet* $A = (\Sigma, Q, q_0, F, \delta)$, et $\overline{L} = \Sigma^* \setminus L$. Posons $A' = (\Sigma, Q, q_0, Q \setminus F, \delta)$; alors l'automate A' reconnait $\overline{L}$. □

Remarque. Il est important que l'automate A soit complet car un blocage de A serait aussi un blocage de A'.

Théorème. — *Les langages reconnaissables sont clos par intersection.*

Preuve. Soient L_1 et L_2 deux langages reconnus respectivement par les automates déterministes complets $A_1 = (\Sigma, Q_1, q_{1,0}, F_1, \delta_1)$ et $A_2 = (\Sigma, Q_2, q_{2,0}, F_2, \delta_2)$. On pose $Q = Q_1 \times Q_2$, $q_0 = (q_{1,0}, q_{2,0})$, $F = F_1 \times F_2$ et $\delta((q_1, q_2), a) = (\delta_1(q_1, a), \delta(q_2, a))$. Alors $A = (\Sigma, Q, q_0, F, \delta)$ reconnait le langage $L_1 \cap L_2$.
En effet, considérons $u \in \Sigma^*$ et (q_1, q_2) l'état auquel aboutit le chemin étiqueté par u (il ne peut y avoir de blocage car les automates sont complets).

- Si $u \notin L_1$ alors $q_1 \notin F_1$ donc $(q_1, q_2) \notin S$;
- si $u \notin L_2$ alors $q_2 \notin F_2$ donc $(q_1, q_2) \notin S$;
- si $u \in L_1 \cap L_2$ alors $q_1 \in F_1$ et $q_2 \in F_2$ donc $(q_1, q_2) \in F$.

□

Sachant que le théorème de Kleene à prouvé l'équivalence entre langages rationnels et langages reconnaissables, une conséquence importante de ces deux derniers résultats est que :

Les langages rationnels sont clos par complémentation et par intersection.

On notera que ce résultat n'est pas du tout évident si on s'en tient à la définition des expressions rationnelles donnée dans le chapitre précédent.

À l'inverse les propriétés de stabilité par union, concaténation et passage à l'étoile de Kleene des langages rationnels permettent d'en déduire que les langages reconnaissables sont clos par union, concaténation et passage à l'étoile (cela dit, il n'est pas très compliqué de prouver ces propriétés en construisant les automates associés à ces constructions ensemblistes).

Terminons cette section avec un dernier résultat de clôture :

Théorème. — *Les langages reconnaissables sont clos par passage au miroir.*

Preuve. Considérons un automate *non déterministe* $A = (\Sigma, Q, I, F, \delta)$ et définissons l'automate $A' = (\Sigma, Q, F, I, \delta')$ avec :

$$\delta'(q, x) = q' \iff \delta(q', x) = q$$

(concrètement on inverse le sens des arcs de A).
Alors le langage reconnu par A' est l'image miroir du langage reconnu par A. □

Une conséquence du théorème de Kleene est que les langages rationnels sont eux aussi stables par passage au langage miroir.

3.4 Lemme de l'étoile [7]

On dispose désormais de deux manières de prouver qu'un langage est rationnel : en exhibant une expression rationnelle qui le dénote ou un automate qui le reconnait. En revanche, pour montrer qu'un langage n'est pas rationnel il peut être utile de disposer du résultat suivant :

Lemme (de l'étoile). — *Si* L *est un langage rationnel il existe un entier* k *tel que tout mot* $m \in L$ *de longueur supérieure ou égale à* k *se factorise sous la forme* $m = uvw$ *avec :*

$$(i) \quad |v| \geqslant 1 \qquad (ii) \quad |uv| \leqslant k \qquad (iii) \quad \forall n \in \mathbb{N},\ uv^n w \in L.$$

7. Attention, ce résultat n'est pas explicitement au programme.

Remarque. Ce résultat est aussi connu sous le nom de *lemme de pompage*, dans le sens où le facteur v du mot m peut être « pompé » un nombre quelconque de fois et produire ainsi des mots de L.

Rappelons que tout langage fini est rationnel et notons que pour ces derniers ce résultat est évident dès lors qu'on considère un entier k strictement supérieur au plus long des mots de L. En revanche, lorsque L est de cardinal infini il possède nécessairement des mots de longueur arbitrairement grande. Pour ces langages ce résultat affirme que passé une certaine taille les mots de L sont toujours construits par répétition d'un facteur v s'insérant au sein d'un mot uw de L.

Preuve. Considérons un automate fini déterministe $A = (\Sigma, Q, q_0, F, \delta)$ et $k = |Q|$. Soit m un mot de L tel que $|m| \geqslant k$. Le chemin $q_0 \xrightarrow{a_1} q_1 \xrightarrow{a_2} q_2 \cdots \xrightarrow{a_p} q_p$ reconnaissant m implique $p+1$ états donc passe nécessairement deux fois par le même état $q_i = q_j$ avec $0 \leqslant i < j \leqslant k$:

$$q_0 \xrightarrow{a_1} \cdots \xrightarrow{a_i} q_i \cdots\cdots \xrightarrow{a_j} q_j \longrightarrow \cdots \xrightarrow{a_p} q_p$$

Posons $u = a_1 \cdots a_i$, $v = a_{i+1} \cdots a_j$ et $w = a_{j+1} \cdots a_p$. Alors pour tout $n \in \mathbb{N}$ le chemin étiqueté par $uv^n w$ conduit à l'état acceptant q_p donc $uv^n w \in L$. □

Ce résultat est le principal utilisé pour prouver qu'un langage n'est pas rationnel mais attention, *il n'exprime pas une équivalence* : il existe des langages non rationnels qui vérifient les conclusions du lemme de l'étoile.

Il existe plusieurs variantes du lemme de l'étoile ; dans la première version un facteur arbitraire est itéré ; la version suivante itère un facteur qui se trouve dans une zone prédéterminée du mot, ce qui permet plus de souplesse dans l'application de ce résultat :

Lemme (de l'étoile). — *Si* L *est un langage rationnel il existe un entier k tel que tout mot $m = uvw$ tel que $|v| \geqslant k$ se factorise sous la forme $m = u(v_1 v_2 v_3)w$ avec :*

$$(i) \quad |v_2| \geqslant 1 \qquad (ii) \quad |v_1 v_2| \leqslant k \qquad (iii) \quad \forall n \in \mathbb{N},\ u(v_1 v_2^n v_3)w \in L.$$

Preuve. On applique la méthode décrite dans la preuve précédente uniquement après avoir parcouru le chemin étiqueté par u. □

Exemple. Le langage $L = \left\{a^n b^n \;\middle|\; n \in \mathbb{N}\right\}$ n'est pas rationnel. Supposons en effet les conclusions du lemme de l'étoile vérifiées et posons $u = a^k$, $v = b^k$ et $w = \varepsilon$. D'après le lemme précédent v se factorise en $v = b^{k_1} b^{k_2} b^{k_3}$ avec $k_2 \geqslant 1$ et on doit avoir : $\forall n \in \mathbb{N}$, $a^k b^{k+nk_2} \in L$, ce qui est absurde.

Exemple. Si Σ possède au moins deux lettres le langage $L = \left\{m^2 \;\middle|\; m \in \Sigma^*\right\}$ des carrés parfaits n'est pas rationnel. Supposons en effet les conclusions du lemme de l'étoile vérifiées et posons $m = ab^k$. Le mot m^2 se factorise en $m^2 = uvw$ avec $u = a$, $v = b^k$ et $w = ab^k$. D'après le lemme ci-dessus il existe $j \geqslant 1$ tel que pour tout $n \in \mathbb{N}$, $ab^{k+nj}ab^k \in L$, ce qui est absurde.

4. Exercices

4.1 Automates finis déterministes

Exercice 1 Dans chacun des cas déterminer un automate reconnaissant sur l'alphabet $\{a, b\}$ les langages suivants :

1. L est le langage des mots contenant au moins une fois la lettre a ;
2. L est le langage des mots contenant au plus une fois la lettre a ;
3. L est le langage des mots contenant un nombre pair de fois la lettre a ;
4. L est le langage des mots admettant aba pour facteur ;
5. L est le langage des mots admettant aba pour sous-mot.

Exercice 2 Dessiner un automate fini déterministe sur l'alphabet $\{0, 1\}$ qui reconnait les représentations binaires des entiers divisibles par 5.
Dessiner un automate fini déterministe sur l'alphabet $\{0, 1\}$ qui reconnait les représentations binaires des entiers divisibles par 5, lorsque ceux-ci sont lus à partir du bit de poids le plus faible.

Exercice 3 Sur l'alphabet $\Sigma = \{a, b\}$ déterminer un automate qui reconnait le langage des mots comptant au moins deux occurrences de leur dernier caractère.

Exercice 4 On considère l'alphabet $\Sigma = \{0, 1\}$ et le langage $L = 1(0+1)^*$ des mots qui commencent par 1. Chaque mot u de L peut être considéré comme l'écriture binaire d'un entier $n > 0$. On note alors succ(u) l'écriture binaire de $n+1$.

1. On note $L' = \left\{u \in L \;\middle|\; |\text{succ}(u)| = |u|\right\}$. Définir un automate fini déterministe qui reconnait le langage L'.

On considère le morphisme h qui double chaque lettre d'un mot, c'est-à-dire celui défini par $h(0) = 00$ et $h(1) = 11$, et $E = \left\{h(u) \;\middle|\; u \in L\right\}$.

2. Définir un automate fini déterministe qui reconnait le langage E.

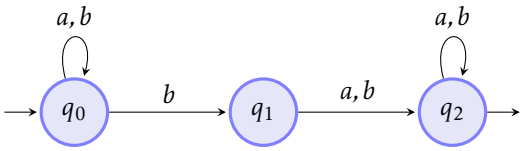
Pour tout mot $u \in L'$ on note $s(u)$ le mot qui entrelace u et succ(u) : si $u = a_1 a_2 \cdots a_n$ et succ$(u) = b_1 b_2 \cdots b_n$ alors $s(u) = a_1 b_1 a_2 b_2 \cdots a_n b_n$. On pose $S = \left\{s(u) \;\middle|\; u \in L'\right\}$.

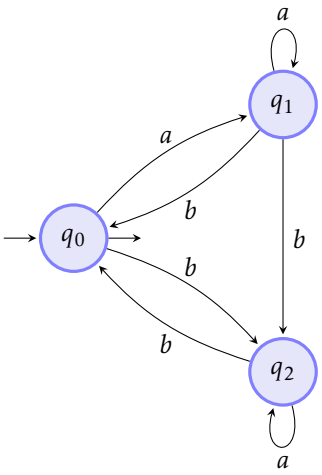
3. Trouver deux langages L_1 et L_2 tels que $S = EL_1L_2$ et en déduire un automate fini déterministe qui reconnait le langage S.

Exercice 5 Rédiger une fonction Caml qui calcule un automate émondé équivalent à un automate passé en paramètre.

4.2 Automates non déterministes

Exercice 6 Déterminiser les automates suivants :





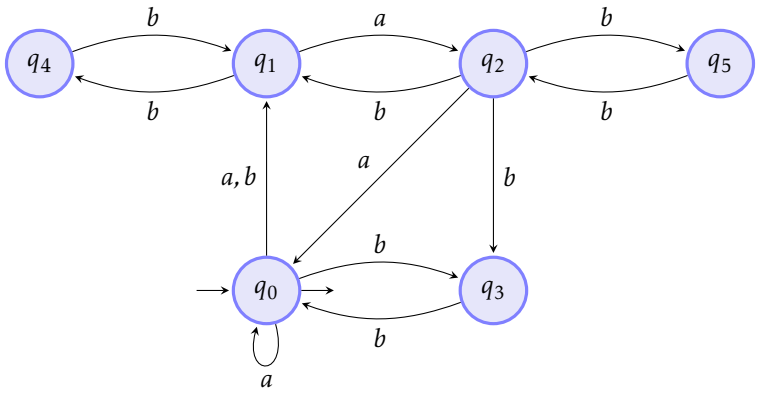
Exercice 7 Un barman aveugle joue au jeu suivant avec un client : il a devant lui un plateau sur lequel sont disposés quatre verres formant un carré. Chacun de ces verres peut être retourné ou non, sans que le barman ne le sache. Le but de ce dernier est de s'arranger pour que tous les verres soient tournés dans le même sens. Pour ce faire, il peut à chaque tour choisir l'une des trois actions suivantes :

- tourner l'un des verres ;
- tourner deux verres voisins ;
- tourner deux verres opposés ;

mais pour corser la difficulté, le client peut tourner le plateau d'un nombre quelconque de quart de tours entre chacune des actions du barman. Le jeu s'arrête dès qu'une des deux positions gagnantes est atteinte.

1. Montrer qu'on peut restreindre à quatre le nombre de configurations différentes, puis représenter les actions possibles du jeu par un automate non déterministe.
2. Déterminiser cet automate et en déduire une stratégie gagnante pour le barman.

Exercice 8 On considère l'automate non déterministe A défini par la figure suivante :



Le *transposé* de A, note $\mathcal{T}(A)$, est l'automate dans lequel toutes les flèches de A ont été inversées (y compris les états initiaux et acceptants).
Le déterminisé de A, noté $\mathcal{D}(A)$ est l'automate déterministe obtenu par l'algorithme de déterminisation du cours.
Construisez $A' = \mathcal{D}(\mathcal{T}(A))$ puis $A'' = \mathcal{D}(\mathcal{T}(A'))$ et justifier que A et A'' reconnaissent le même langage.

Remarque. Cette méthode, appelée *minimisation de* Brzozowski, permet d'obtenir un automate déterministe équivalent à A et de taille minimale.

Exercice 9 Donner un automate non déterministe reconnaissant les mots possédant le facteur *abba*, et le déterminiser. Appliquer ensuite l'algorithme KMP pour obtenir un automate déterministe reconnaissant ce même langage.

4.3 Théorème de Kleene

Exercice 10 Appliquer l'algorithme de Berry-Sethi pour déterminer l'automate de Glushkov associé à l'expression rationnelle $(a+c)^*(abb+\varepsilon)$.

Exercice 11 Construire un automate reconnaissant le langage des mots m sur $\Sigma = \{a, b\}$ tels que $|m|_a$ soit un multiple de 3, et en déduire une expression rationnelle dénotant ce langage.

Exercice 12 On considère le langage L sur l'alphabet $\Sigma = \{a, b\}$ dénoté par l'expression $(ab+abb+aa)^*$. Trouver un automate à quatre états qui reconnait L, puis construire un automate qui reconnait le complémentaire $\overline{L}$.

Exercice 13 L'exercice 4 montre qu'il est possible de décider si le langage reconnu par un automate est vide : il suffit de l'émonder puis de vérifier si les états acceptants font partie des états utiles. En déduire qu'il est possible de décider si deux expressions rationnelles dénotent le même langage.

Exercice 14 On note pref(L) l'ensemble des préfixes des mots de L, suff(L) l'ensemble des suffixes, fact(L) l'ensemble des facteurs. Montrer que si L est rationnel il en est de même de ces trois langages.

Exercice 15 On rappelle que si L est un langage sur un alphabet Σ alors $\sqrt{L} = \left\{u \in \Sigma^* \;\middle|\; u^2 \in L\right\}$.
Montrer que si L est rationnel il en est de même de $\sqrt{L}$.

Exercice 16 On rappelle que le quotient gauche de deux langages K et L est défini par :

$$K^{-1}L = \left\{v \in \Sigma^* \;\middle|\; \exists u \in K \text{ tel que } uv \in L\right\}.$$

Montrer que si L est un langage rationnel il en est de même de $K^{-1}L$ (on notera qu'on ne fait pas d'hypothèses particulières sur K).

4.4 Lemme de l'étoile

Exercice 17 Montrer que les langages suivants ne sont pas rationnels :

$$L_1 = \left\{ a^i b^j \;\middle|\; i < j \right\} \qquad L_2 = \left\{ u \in \{a,b\}^* \;\middle|\; |u|_a = |u|_b \right\} \qquad L_3 = \left\{ a^p \;\middle|\; p \text{ est premier} \right\}$$

Exercice 18 Montrer que le langage de Dick n'est pas rationnel. Caml est-il un langage rationnel ?

Exercice 19 Soit Σ un alphabet ayant au moins deux lettres, et L l'ensemble des palindromes [8] de Σ^*. Montrer que L n'est pas un langage rationnel.

Exercice 20 On note L l'ensemble des mots sur l'alphabet $\Sigma = \{0,1\}$ qui correspondent à l'écriture binaire d'un nombre premier. On rappelle qu'un nombre de Mersenne est un nombre premier de la forme $M_n = 2^n - 1$. On conjecture que ceux-ci sont en nombre infini, mais ceci n'a pas encore été prouvé.
Montrer que si cette conjecture est vraie alors L n'est pas un langage rationnel.

8. Un *palindrome* est un mot égal à son image miroir.