Chapitre 7

Automates finis

Introduction

On cherche ici à résoudre le problème de décision suivant :

Un mot m étant donné, m appartient-il au langage L ?

Pour cela, on va construire des "machines abstraites" qui savent reconnaître l'appartenance ou la non-appartenance d'un mot à un langage donné. Ces machines sont appelées des **automates**.

A langage connu, des questions se posent :

- existe-t-il un automate qui reconnait le langage, c'est-à-dire qui permet de savoir si un mot appartient au langage ?
- dans la négative, quelle est la classe des langages reconnaissables par un automate ?

et à automate connu :

- quel est le langage reconnu par l'automate ?
- existe-t-il un automate plus simple effectuant le même travail ?

1 Automates finis déterministes

1.1 Définition et représentation

Un **automate** est une machine abstraite permettant de reconnaître un langage sur un alphabet. De manière formelle, on donne la définition suivante :

Définition.

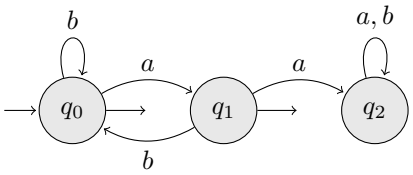
Un **automate fini déterministe** (DFA pour deterministic finite automaton) est défini par un quintuplet $A = (\Sigma, Q, q_0, F, \delta)$ où :

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

Représentations d'un automate.
Il est d'usage de représenter un automate par un graphe dont les noeuds sont les états et les arêtes les couples (q_i, q_j) appartenant à Q^2 étiquetés par a appartenant à Σ 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



L'état initial (ici q_0) est désigné par une flèche entrante, les états acceptants (ici q_0 et q_1) sont représentés par une flèche sortante (certains auteurs représentent les états acceptants avec un double cercle).

Il sera souvent pratique d'étendre la définition de δ aux mots :

Définition.

Soit $A = (\Sigma, Q, q_0, F, \delta)$ un automate fini déterministe. La fonction de transition δ^* étendue aux mots est la fonction partielle $\delta^* : Q \times \Sigma^* \to Q$ définie récursivement par :

- pour tout $q \in Q$, $\delta^*(q, \varepsilon) = q$,
- pour tous $q \in Q$, $u \in \Sigma^*$ et $a \in \Sigma$, $\delta^*(q, ua) = \delta(\delta^*(q, u), a)$.

1.2 Langage reconnu

Le fonctionnement d'un automate est intuitif : il lit un mot symbole par symbole, en partant de l'état initial et en se déplaçant suivant la fonction de transition. Si en fin de mot l'état courant est un état acceptant, le mot est accepté. Dans le cas contraire, il est rejeté. Plus formellement :

Définition.

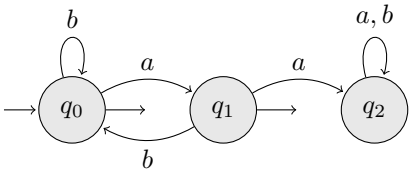
Soit $A = (\Sigma, Q, q_0, F, \delta)$ un automate fini déterministe.

- Une **transition** est un triplet (q_i, a, q_j) tel que $\delta(q_i, a) = q_j$. Elle sera représentée par $q_i \xrightarrow{a} q_j$.
- Un **chemin** dans A est une suite finie de transitions consécutives débutant par l'état initial q_0 :

$$q_0 \xrightarrow{a_1} q_1 \xrightarrow{a_2} \dots \xrightarrow{a_n} q_n$$

  Le mot $u = a_1 a_2 \dots a_n$ est alors appelé l'**étiquette du chemin** et on a : $\delta^*(q_0, u) = q_n$.
- Un mot $u \in \Sigma^*$ est **reconnu par** A si et seulement si $\delta^*(q_0, u) \in F$.

Exemple. Reprenons l'automate défini précédemment :



Les mots *aba* et *bbab* sont reconnus par l'automate. Le mot *abaab* ne l'est pas.

Définition.

- Le **langage reconnu** par un automate A, noté $\mathscr{L}(A)$, est l'ensemble des mots reconnus par A.
- Un langage L est dit **reconnaissable** s'il existe un automate fini déterministe A tel que $L = \mathscr{L}(A)$.
- On note $\mathrm{Rec}(\Sigma)$ l'ensemble des langages reconnaissables sur un alphabet Σ.

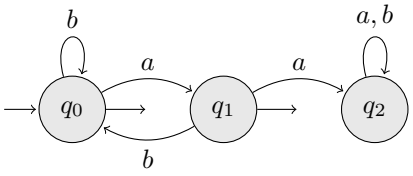
Remarque. Tous les langages ne sont pas reconnaissables, puisque l'ensemble des automates est dénombrable alors que l'ensemble des langages ne l'est pas. En effet :

- Σ^* est un ensemble infini, donc $\mathscr{P}(\Sigma^*)$, qui est l'ensemble des langages, est indénombrable.
- Les ensembles d'états possibles sont dénombrables (l'ensemble des $[\![0, n-1]\!]$, pour $n \in \mathbb{N}^*$) et une fois Q fixé, il n'y a qu'un nombre fini de façons de choisir q_0, F et δ.

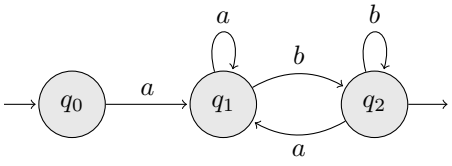
Le théorème de Kleene que nous verrons plus loin établit l'équivalence entre les expressions rationnelles et les automates finis dans le sens où ces deux notions définissent les mêmes langages.

Exercice.

1. Déterminer le langage reconnu par l'automate A_1 suivant :



2. Déterminer le langage reconnu par l'automate A_2 suivant :



1.3 Implémentation

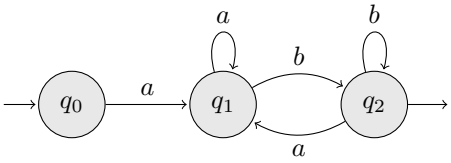
Pour modéliser un automate en OCaml, il est pratique d'utiliser le type `char` pour représenter l'alphabet Σ, le type `int` pour l'ensemble des états et un dictionnaire pour énumérer la liste des transitions possibles.

Pour simplifier, la liste des transitions possibles sera représentée par le type `((int * char) * int) list` et nous définirons la fonction `assoc` pour associer au couple (q, a) l'état $\delta(q, a)$.

Ceci conduit à définir le type :

```
type dfa = {start: int ; accept: int list ; delta: ((int * char) * int) list} ;;
```

Par exemple, l'automate



sera défini par :

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

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

2 Simplification des automates

2.1 Complétion

Définition.

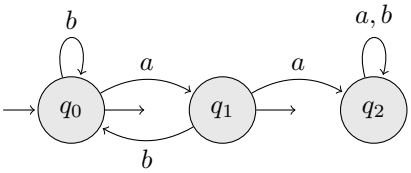
Soit $A = (\Sigma, Q, q_0, F, \delta)$ un automate fini déterministe.

- Un **blocage** de l'automate A est un couple $(q, a) \in Q \times \Sigma$ pour lequel la fonction de transition δ n'est pas définie.
- Un automate sans blocage est dit **complet**.

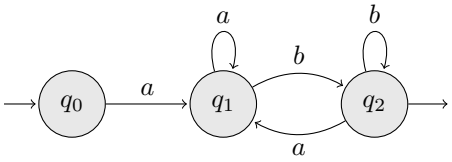
Remarque. Un automate est donc complet si et seulement si δ est définie sur tout l'ensemble $Q \times \Sigma$.

Exemples.

1. L'automate A_1 suivant est complet :



2. L'automate A_2 suivant n'est pas complet :



En effet, 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. Le couple (q_0, b) est donc un blocage de l'automate.

Propriété 1 (Complétion d'un automate)

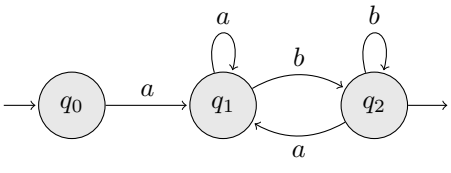
Tout langage reconnaissable est reconnu par un automate complet.

Preuve.

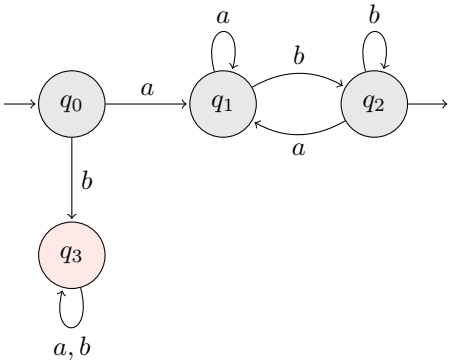
□

Méthode.
Pour compléter un automate, on ajoute un état **puits** et tous les blocages deviennent des transitions vers cet état.

Exemple. Avec cette méthode, on passe ainsi de l'automate incomplet



à l'automate complet

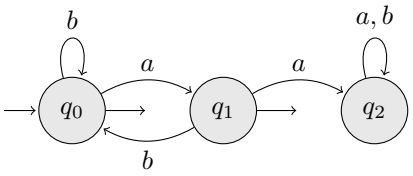


2.2 Émondage

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.

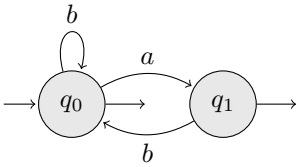
Or ceci peut s'avérer particulièrement inefficace : par exemple, dans le cas de l'automate



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 (q_2 est un puits). Puisqu'il ne s'agit pas d'un état acceptant, il peut être supprimé sans modifier le langage reconnu par l'automate :

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

δ	a	b
q_0	q_1	q_0
q_1	$-$	q_0



L'automate obtenu est dit émondé. Plus formellement :

Définition.

Soit $A = (\Sigma, Q, q_0, F, \delta)$ un automate fini déterministe. On dit d'un état $q \in Q$ 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,
- **utile** si et seulement s'il est accessible et co-accessible.

Si tous les états $q \in Q$ sont utiles, on dit que l'automate A est **émondé**.

Propriété 2 (Émondage d'un automate)

Tout langage reconnaissable non vide est reconnu par un automate émondé.

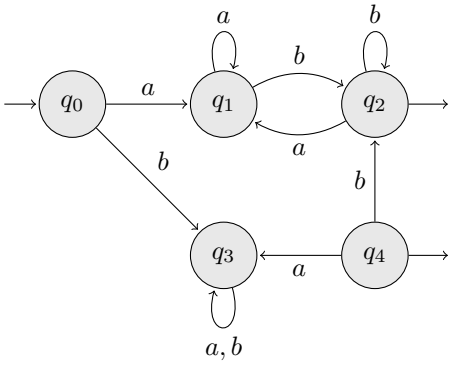
Preuve.

□

Méthode.

Pour émonder un automate, on efface les états qui ne sont pas à la fois accessibles et co-accessibles et toutes les transitions impliquant ces états.

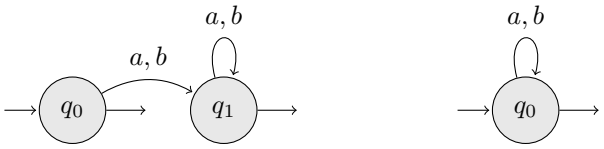
Exercice. Donner une version émondée de l'automate A ci-dessous puis le langage reconnu par A.



Automate A

Automate A émondé

Remarque. Le fait qu'un automate soit émondé ne signifie pas qu'il soit le plus petit possible (au sens du nombre d'états). Considérons par exemple les deux automates suivants :



Ces deux automates reconnaissent tous les deux le langage Σ^* pour $\Sigma = \{a, b\}$. On peut aussi facilement vérifier que ces automates sont complets et émondés. Cependant, le deuxième automate possède un état de moins que le premier.

2.3 Standardisation

Définition.

Un automate $A = (\Sigma, Q, q_0, F, \delta)$ est dit **standard** si et seulement s'il n'existe pas de lettre $x \in \Sigma$ et d'état $q \in Q$ tel que $\delta(q, x) = q_0$.

Autrement dit, il n'existe pas de transition aboutissant à l'état initial.

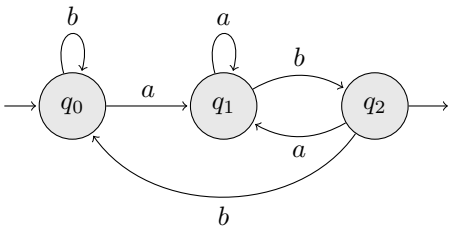
Propriété 3 (Standardisation d'un automate)

Tout langage reconnaissable est reconnu par un automate standard.

Preuve.

□

Exercice. Donner une version standardisée de l'automate A ci-dessous puis le langage reconnu par A.



Automate A

Automate A standardisé

Exercice 1 (★)
On considère l'alphabet $\Sigma = \{a, b\}$. Déterminer un automate reconnaissant le langage des mots sur Σ :

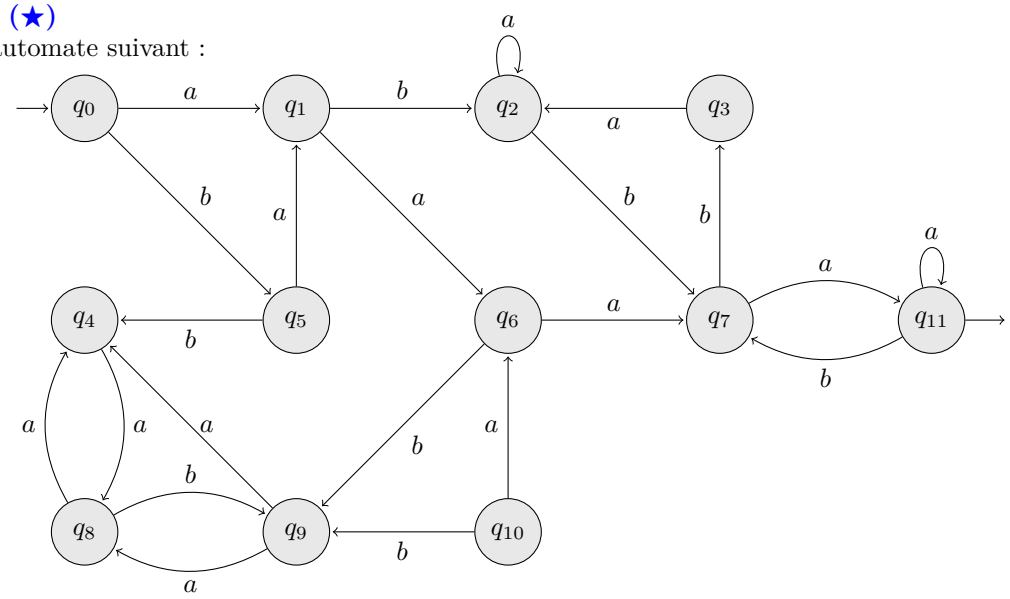
1. comptant au plus une occurrence de a ;
2. ne comportant pas le facteur aa ;
3. ayant un nombre pair de a ;
4. ayant un nombre pair de a et un nombre multiple de 3 de b ;
5. n'ayant pas plus de deux occurrences consécutives de la même lettre.

Exercice 2 (★★)
On considère l'alphabet $\Sigma = \{a, b\}$. Déterminer un automate reconnaissant le langage des mots sur Σ :

1. ayant aba pour préfixe ;
2. ayant aba pour suffixe ;
3. ayant aba pour sous-mot ;
4. ayant aba pour facteur.

Exercice 3 (★)
Montrer qu'il existe des langages reconnus par un automate déterministe qui ne peuvent être reconnus par un automate déterministe avec un seul état acceptant. On pourra considérer le langage $\{\varepsilon, a\}$.

Exercice 4 (★)
Émonder l'automate suivant :



Exercice 5 (★★)
Soit l'alphabet $\Sigma = \{0, 1\}$. Déterminer des automates reconnaissant :

1. Le langage des mots qui sont l'écriture binaire d'un multiple de 2.
2. Le langage des mots qui sont l'écriture binaire d'un multiple de 4.
3. Le langage des mots qui sont l'écriture binaire d'un multiple de 3.

Exercice 6 (★★)
Soit A un automate à n états et L le langage reconnu par cet automate.
Montrer que si L est non vide, alors il contient au moins un mot de longueur au plus $n - 1$.

---

3 Automates finis 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.

3.1 Définitions

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 de $Q \times \Sigma$ dans $\mathscr{P}(Q)$, appelée **fonction de transition**.

Remarque. Il n'y a plus ici de blocage au sens déjà étudié, mais $\delta(q, a)$ peut prendre la valeur $\emptyset$.

Attention.

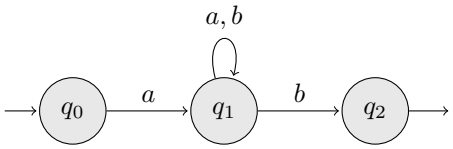
Il y a dans cette définition plusieurs défauts de déterminisme :

- il peut y avoir plusieurs états initiaux ;
- il peut y avoir plusieurs transitions possibles depuis un état pour un caractère donné.

Exemple. L'automate A suivant est non déterministe :

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

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



Définition.

Soit $A = (\Sigma, Q, I, F, \delta)$ un automate fini non déterministe.

- Une **transition** est un triplet (q_i, a, q_j) tel que q_j appartient à $\delta(q_i, a)$.
  Elle est représentée par $q_i \xrightarrow{a} q_j$.
- Un **chemin** dans l'automate A est une suite finie de transitions consécutives
  $$q_0 \xrightarrow{a_1} q_1 \xrightarrow{a_2} \dots \xrightarrow{a_n} q_n$$
  débutant par l'état initial q_0, le mot $a_1 a_2 \dots a_n$ est appelé l'**étiquette du chemin**.
- Un chemin est dit **acceptant** si et seulement si l'état d'arrivée de celui-ci est un état acceptant.
- Un mot de Σ^* est **reconnu par l'automate** A si et seulement s'il étiquette un chemin acceptant.
- Le **langage reconnu** par l'automate A, noté $\mathscr{L}(A)$, est l'ensemble des mots reconnus par A.

Remarque.

- A la différence des automates déterministe, il peut exister plusieurs chemins dans un automate non déterministe pour un mot donné.

  Par exemple, pour l'automate non déterministe A précédent et pour le mot abb, nous avons les deux chemins :
  $$q_0 \xrightarrow{a} q_1 \xrightarrow{b} q_1 \xrightarrow{b} q_1 \qquad \text{et} \qquad q_0 \xrightarrow{a} q_1 \xrightarrow{b} q_1 \xrightarrow{b} q_2.$$

- Un mot m est reconnu par un automate non déterministe si et seulement s'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.

  Par exemple, le mot abb est reconnu par l'automate non déterministe A précédent puisque le chemin
  $$q_0 \xrightarrow{a} q_1 \xrightarrow{b} q_1 \xrightarrow{b} q_2$$
  est acceptant.

Exercice. Déterminer le langage reconnu par l'automate non déterministe A précédent.

3.2 Déterminisation

Nous avons introduit deux modèles d'automates : déterministes et non déterministes. Il se trouve que ces deux modèles sont équivalents dans le sens où ils reconnaissent les mêmes langages. On pourra donc parler en général de **langage reconnaissable**, sans avoir besoin de préciser par quel type d'automate fini.

Pour prouver ce résultat, on commence par introduire la définition suivante :

Définition.

Soit $A_N = (\Sigma, Q_N, I_N, F_N, \delta_N)$ un automate non déterministe.

On appelle **automate des parties** associé à A_N, l'automate déterministe $A_D = (\Sigma, Q_D, I, F_D, \delta_D)$ construit comme suit :

- $Q_D = \mathscr{P}(Q_N)$,
- $F_D = \{P \in Q_D \mid P \cap F_N \neq \emptyset\}$,
- $\delta_N : \left\{ \begin{array}{ccc} Q_D \times \Sigma & \to & Q_D \\ (P, a) & \mapsto & \displaystyle\bigcup_{q \in P} \delta_N(q, a) \end{array} \right.$

On peut alors démontrer le résultat annoncé :

Théorème 4 (Déterminisation)

Soit $L \subset \Sigma^*$ un langage.

L est reconnu par un **automate fini déterministe** si et seulement s'il est reconnu par un **automate fini non déterministe**.

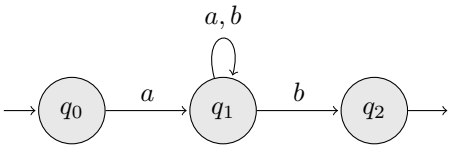
Méthode.

Pour déterminiser un automate, on peut procéder comme suit :

- on écrit un tableau dont les colonnes sont indexées par les lettres et la première ligne est indexée par l'état initial de l'automate des parties ;
- on indique dans cette première ligne, à quel(s) état(s) mène chaque lettre puis on ajoute des lignes correspondant aux ensembles d'états atteints non encore considérés ;
- on recommence jusqu'à ce que le procédé s'arrête.

On n'obtient ainsi que les états accessibles (mais pas forcément co-accessibles) de l'automate des parties.

Exercice. Considérons toujours l'automate non déterministe :



Construire l'automate déterministe associé, puis donner la version émondée.

Remarque. Le résultat précédent prouve que les automates non déterministes ne sont pas plus "riche" 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. Ils fournissent de plus 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.

Exercice. Considérons le langage L dénoté par $(a+b)^*a(a+b)^{n-1}$.

1. Donner un automate non déterministe à $n+1$ états qui reconnait L.

2. Montrer que tout automate déterministe qui reconnait L possède au moins 2^n états.

3.3 Les ε-transitions

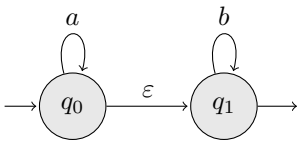
Nous allons maintenant présenter une variante des automates finis qui autorise l'utilisation de transitions qui ne lisent pas de lettre du mot.

Ces transitions sont appelées des **ε-transitions** (ou des **transitions spontanées**) et notées avec un ε (symbole du mot vide...) en guise d'étiquette :

$$q_i \xrightarrow{\varepsilon} q_j$$

Exercice.

1. Considérons l'automate :



   Donner le langage reconnu par cet automate.

2. On souhaite construire un automate qui reconnaît les mots commençant par *aba* ou se terminant par *bab*.
   (a) Construire un automate qui reconnaît les mots commençant par *aba*.

   (b) Construire un automate qui reconnaît les mots se terminant par *bab*.

   (c) En assemblant ces deux automates à l'aide d'ε-transitions, construire un automate qui reconnaît les mots commençant par *aba* ou se terminant par *bab*.

Remarques.

1. L'utilisation des ε-transitions permet parfois de décrire plus lisiblement un langage. Nous verrons plus loin qu'elles permettent aussi de simplifier la description de certaines opérations (l'union, la concaténation et la fermeture de Kleene).
2. L'existence d'ε-transitions rend généralement l'automate non déterministe, puisque nous avons toujours la possibilité d'emprunter une telle transition même quand il existe une transition ordinaire que nous pourrions emprunter.

Définition.

Un **automate fini non déterministe asynchrone** (c'est-à-dire contenant des ε-transitions) 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 de $Q \times (\Sigma \cup \{\varepsilon\})$ dans $\mathscr{P}(Q)$, appelée **fonction de transition**.

Le théorème suivant montre que les automates avec ε-transitions décrivent la même classe de langages que les automates sans ε-transition. On pourra donc encore parler de **langage reconnaissable**, sans préciser si l'automate est ou non asynchrone.

Théorème 5 (Suppression des ε-transitions)

Soit $L \subset \Sigma^*$ un langage.

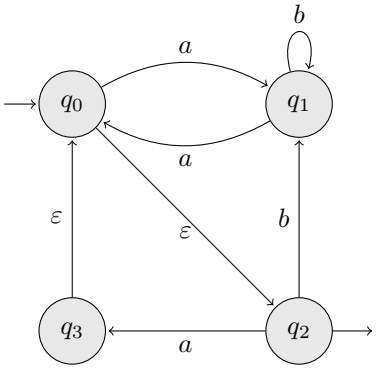
L est reconnu par un **automate fini non déterministe asynchrone** si et seulement s'il est reconnu par un **automate fini non déterministe**.

Nous allons proposer deux algorithmes d'élimination des ε-transitions qui transforment l'automate initial asynchrone en un automate équivalent sans ε-transition.

Clôture transitive.

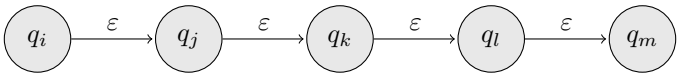
Les deux algorithmes commencent par une **clôture transitive** des ε-transitions de l'automate asynchrone. En notant $G = (Q, A)$ le graphe associé à l'automate, nous considérons le graphe partiel $G' = (Q, A')$ de G où A' se limite aux arcs correspondant aux ε-transitions. Nous ajoutons à A les couples de sommets (s, t) appartenant à Q^2 reliés par un chemin dans A', qui deviennent de nouvelles ε-transitions dans l'automate.

Exercice. Effectuer la clôture transitive des ε-transitions de l'automate :



Automate A Clôture transitive de l'automate A

Remarque. Nous ne sommes pas en train d'éliminer les ε-transitions... Par exemple dans le cas :



la clôture transitive de l'automate est :

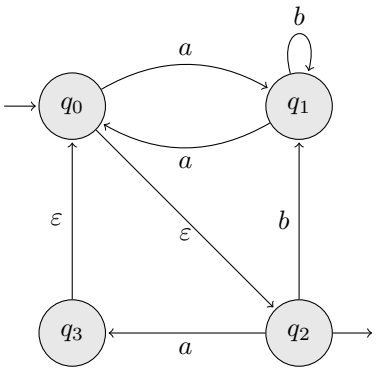
Nous avons donc ajouté six ε-transitions !

Suppression des ε-transitions par fermeture avant.

Dans un premier temps, pour tout état t pour lequel il existe une ε-transition à un état u et pour toute transition d'un état s à l'état t étiquetée par la lettre x (différente donc de ε), nous ajoutons une transition de l'état s à l'état u étiqueté par la lettre x, puis une fois fini, nous supprimons l'ε-transition de l'état t à l'état u.

Dans un deuxième temps, pour tout état t pour lequel il existe une ε-transition d'un état initital i à l'état t, nous ajoutons t à l'ensemble des états initiaux et nous supprimons l'ε-transition de l'état i à l'état t.

Exercice. Éliminer les ε-transitions dans l'automate par fermeture avant.



Automate A

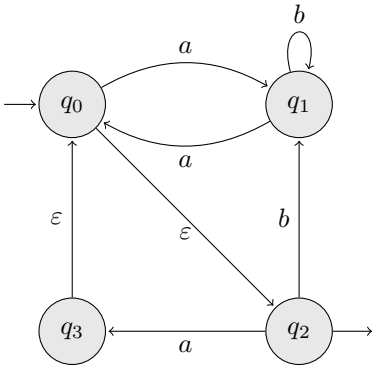
Fermeture avant de l'automate A

Suppression des ε-transitions par fermeture arrière.

Dans un premier temps, pour tout état t pour lequel il existe une ε-transition d'un état s à l'état t et pour toute transition de l'état t à un état u étiquetée par la lettre x (différente donc de ε), nous ajoutons une transition de l'état s à l'état u étiqueté par la lettre x, puis une fois fini, nous supprimons l'ε-transition de l'état s à l'état t.

Dans un deuxième temps, pour tout état t pour lequel il existe une ε-transition de l'état t à un état acceptant f, nous ajoutons t à l'ensemble des états acceptants et nous supprimons l'ε-transition de l'état t à l'état f.

Exercice. Éliminer les ε-transitions dans l'automate par fermeture arrière.



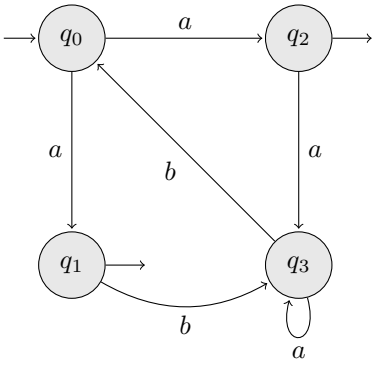
Automate A

Fermeture arrière de l'automate A

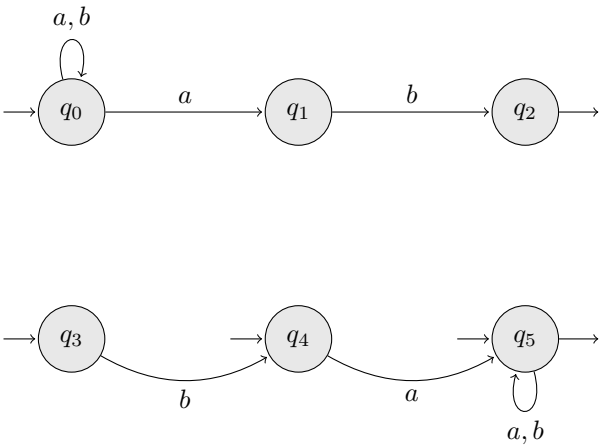
Remarque. Il est possible d'éviter la clôture transitive des ε-transitions de l'automate en définissant la notion de ε-chemin entre deux états et de rechercher, sans faire de clôture, les états reliés par un ε-chemin.

Exercice 7 (★)

Déterminiser les automates suivants :



Automate A_1



Automate A_2

Exercice 8 (★)
Soit $\Sigma = \{a, b\}$. Déterminer un automate non déterministe et un déterministe reconnaissant le langage des mots commençant et terminant par a.

Exercice 9 (★★)
Soit $\Sigma = \{a, b\}$. Trouver un automate non déterministe qui reconnaît le langage des suffixes de $baba$ puis le déterminiser.

Exercice 10 (★★)
Soit Σ l'ensemble des lettres minuscules de l'alphabet français.

1. Définir un automate fini non déterministe acceptant les mots contenant le facteur "cpge".
2. Déterminiser cet automates.

Exercice 11 (★★ - Algorithme KMP)
1. Soit u un mot fixé et $\text{Pref}(u)$ le langage des préfixes de u.
   (a) Déterminer le langage reconnu par l'automate $(\text{Pref}(u), \varepsilon, \{u\}, \delta)$ avec, pour tout $x \in \text{Pref}(u)$ et tout $a \in \Sigma$, $\delta(x, a)$ est le plus long suffixe de xa qui appartient à $\text{Pref}(u)$.
   (b) Adapter l'automate précédent pour qu'il reconnaisse les mots dont u est facteur.
2. On considère dans cette question le cas où l'alphabet est $\Sigma = \{a, b\}$ et le mot u est aba.
   (a) Donner un automate non déterministe qui reconnait le langage $\Sigma^* aba \Sigma^*$ et sa déterminisation.
   (b) Donner l'automate déterministe qui reconnait le langage $\Sigma^* aba \Sigma^*$ en utilisant l'algorithme KMP décrit à la première question.

---

4 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 :

Théorème 6 (de Kleene)

Soit Σ un alphabet et L un langage sur Σ.

L est un langage rationnel si et seulement s'il existe un automate fini A tel que $L = \mathscr{L}(A)$.

Remarque. Pour tout alphabet Σ, nous avons donc l'égalité $\mathrm{Rat}(\Sigma) = \mathrm{Rec}(\Sigma)$.

1. Nous allons montrer que $\mathrm{Rat}(\Sigma) \subset \mathrm{Rec}(\Sigma)$ en décrivant un algorithme construisant explicitement un automate (l'automate de Glushkov ou l'automate de Thomson) associé à une expression rationnelle.
2. Moins utile en pratique, l'inclusion $\mathrm{Rec}(\Sigma) \subset \mathrm{Rat}(\Sigma)$ nous servira essentiellement à justifier l'équivalence dans le théorème de Kleene.
3. Quelques propriétés des langages reconnaissables (en particulier de clôture) seront établies pour être étendues aux langages rationnels.

4.1 L'algorithme de Berry-Sethi

Rappel. Un langage L est local si et seulement s'il existe deux parties P et S de Σ et une partie N de Σ^2 (l'ensemble des mots de longueur 2) telles que :

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

Nous savons que, si de telles parties existent, on a :

- $P = P(L) = \{a \in \Sigma \mid a\Sigma^* \cap L \neq \emptyset\}$ est l'ensemble des premières lettres des mots de L,
- $S = S(L) = \{a \in \Sigma \mid \Sigma^* a \cap L \neq \emptyset\}$ est l'ensemble des dernières lettres des mots de L,
- $N = N(L) = \Sigma^2 \setminus F(L)$ où $F(L) = \{u \in \Sigma^2 \mid \Sigma^* u \Sigma^* \cap L \neq \emptyset\}$ est l'ensemble des facteurs de longueur 2 des mots de L.

Définition.

Un automate déterministe $A = (\Sigma, Q, q_0, F, \delta)$ est **local** si et seulement si, pour toute lettre $x \in \Sigma$, il existe un état $q \in Q$ tel que pour tout $q' \in Q$, (q', x) est un blocage ou $\delta(q', x) = q$.

Autrement dit, pour toute lettre x, toutes les transitions étiquetées par x arrivent dans un même état.

Théorème 7 (Reconnaissance des langages locaux par les automates locaux)

Tout langage local est reconnaissable par un automate local.

Preuve.

□

Rappelons que toute expression rationnelle linéaire dénote un langage local. On déduit donc du théorème précédent la :

Propriété 8

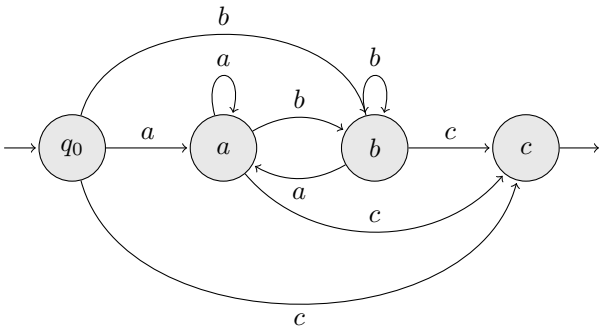
Tout langage dénoté par une expression rationnelle linéaire est local donc reconnaissable par un automate local.

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 et appliqués à cette expression, nous obtenons :

$$P = \{a, b, c\}\ ,\ S = \{c\}\ ,\ F = \{aa, ab, ba, bb, ac, bc\}\ .$$

L'automate local reconnaissant le langage dénoté par cette expression est donc :



4.2 Construction de l'automate de Glushkov

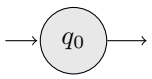
Considérons maintenant une expression rationnelle quelconque e.

Nous avons montré que :

- si le langage dénoté par e est non vide, alors il existe une expression rationnelle équivalente e' ne contenant pas le symbole $\emptyset$ (**Propriété 10** du **Chapitre 6**).
- si le langage dénoté par e est non vide, alors il existe une expression rationnelle e'' ne contenant ni le symbole $\emptyset$, ni le symbole ε, telle que e soit équivalente à ε, e'' ou $\varepsilon + e''$ (**Propriété 11** du **Chapitre 6**).

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 sait également construire des automates reconnaissant les langages $\emptyset$ et $\{\varepsilon\}$:

Automate reconnaissant $\emptyset$



Automate reconnaissant $\{\varepsilon\}$

Nous supposons donc désormais que e est une expression rationnelle ne comportant ni le symbole $\emptyset$, ni le symbole ε. 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.

Nous obtenons alors une nouvelle expression rationnelle sur le nouvel alphabet $\Sigma' = \{c_1, c_2, \ldots, c_n\}$, expression rationnelle qui est linéaire appelée la **linéarisation de l'expression** e.

Exemple. A l'expression

$$e = (a + b)(a^* + ba^* + b^*)^*$$

va être associée l'expression rationnelle linéaire

$$e_l = (c_1 + c_2)(c_3^* + c_4c_5^* + c_6^*)^*.$$

Pour retrouver l'expression rationnelle initiale à partir de l'expression linéarisée, il suffit de connaître la fonction de marquage $\mu : \Sigma' \to \Sigma$ précisant par quel caractère de Σ doit être remplacé le caractère c_k.

Exemple. Dans le cas de l'exemple ci-dessus :

$$\mu(c_1) = \mu(c_3) = \mu(c_5) = a\,,\ \mu(c_2) = \mu(c_4) = \mu(c_6) = b\,.$$

Théorème 9 (Une implication du théorème de Kleene)

Tout langage dénoté par une expression rationnelle sans symbole $\emptyset$ ni ε est reconnaissable par un automate fini.

Preuve.

□

Remarques.

1. Compte tenu de la remarque faite au préalable, ceci prouve le sens direct du théorème de Kleene.
2. 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**.
3. L'automate obtenu par l'algorithme de Berry-Sethi 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.

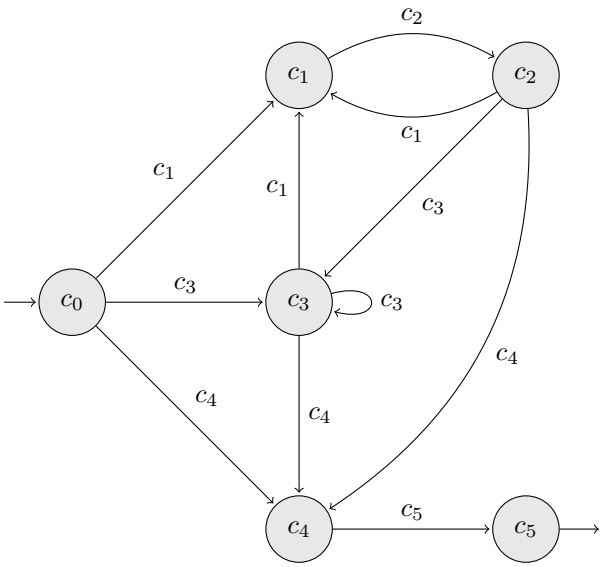
Exemple. Considérons l'expression rationnelle $e = (ab + b)^*ba$. Sa linéarisation est :

$$e' = (c_1c_2 + c_3)^*c_4c_5.$$

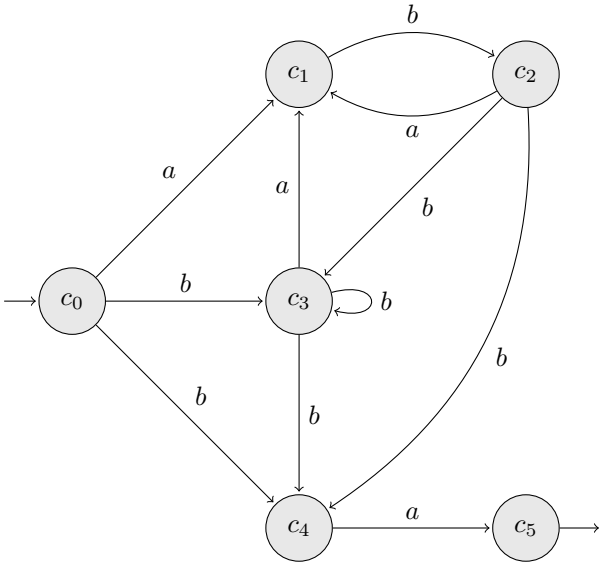
Nous calculons :

$$P = \{c_1, c_3, c_4\}, \quad S = \{c_5\}, \quad F = \{c_1c_2, c_2c_1, c_2c_3, c_2c_4, c_3c_1, c_3c_3, c_3c_4, c_4c_5\}.$$

Donc l'automate local standard associé à e' est :

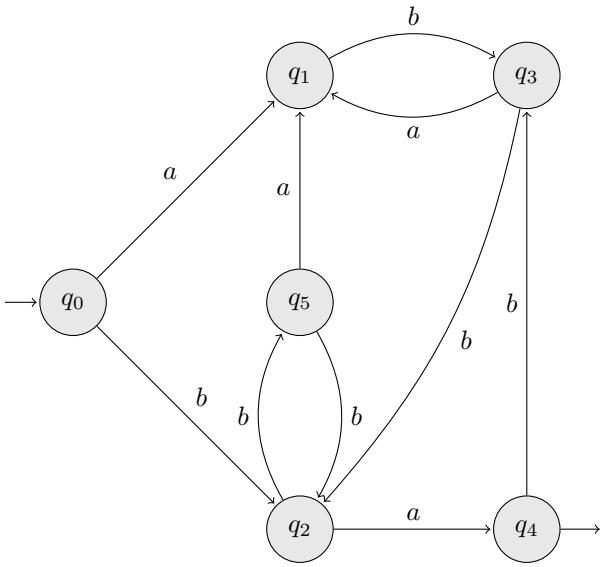


Il nous reste à supprimer le marquage pour obtenir l'automate de Glushkov associé à e :



Nous pouvons finalement le déterminiser :

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



4.3 Construction de l'automate de Thomson

Rappel. Les expressions rationnelles sont définies inductivement de la manière suivante :

- Cas de base :
  - $\emptyset$ et ε sont des expressions rationnelles,
  - pour tout a appartenant à Σ, a est une expression rationnelle.
- Étape d'induction :
  - pour toutes expressions rationnelles e_1 et e_2, la réunion $(e_1 + e_2)$ de e_1 et e_2 est une expression rationnelle,
  - pour toutes expressions rationnelles e_1 et e_2, la concaténation $(e_1.e_2)$ de e_1 et e_2 est une expression rationnelle,
  - pour toute expression rationnelle e, l'itération (aussi appelée l'étoile) (e^*) de e est une expression rationnelle.

Considérons une expression rationnelle e et cherchons à construire un automate $\mathscr{A}(e)$ qui reconnaît le langage dénoté par l'expression rationnelle e. L'algorithme consiste à construire un automate $\mathscr{A}(e)$ par induction structurelle en suivant la construction de l'expression rationnelle e :

- Cas de base :
  - un automate $\mathscr{A}(\varepsilon)$ est :
  - soit a appartenant à Σ, alors un automate $\mathscr{A}(a)$ est :

- Étape d'induction :
  - pour toutes expressions rationnelles e_1 et e_2, un automate $\mathscr{A}(e_1 + e_2)$ est :

  - pour toutes expressions rationnelles e_1 et e_2, un automate $\mathscr{A}(e_1.e_2)$ est :

  - pour toute expression rationnelle e, un automate $\mathscr{A}(e^*)$ est :

Comme nous pouvons le voir, ces constructions font apparaître des ε-transitions qui peuvent être ensuite éliminées et l'automate non déterministe obtenu peut être ensuite déterminisé.

Exemple. Construire l'automate de Thomson associé à l'expression rationnelle $e = (ab + b)^*ba$.

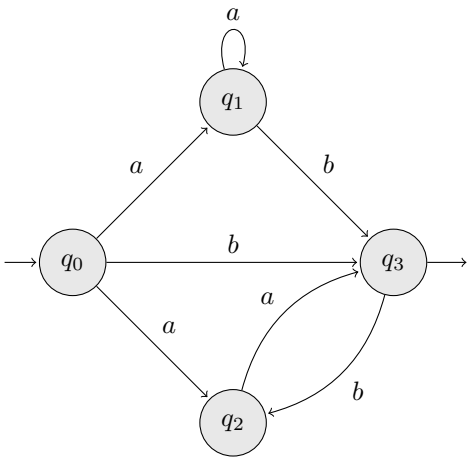
4.4 Des automates aux expressions rationnelles

Théorème 10 (Réciproque du théorème de Kleene)

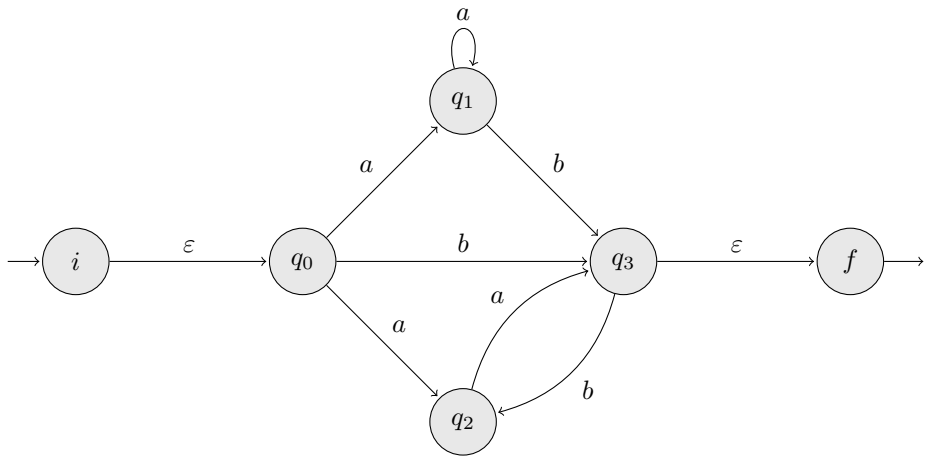
Tout langage reconnu par un automate est rationnel.

Nous allons illustrer la preuve par un exemple en suivant l'algorithme de Brzozowski et McCluskey appelé aussi algorithme d'élimination des états. Il consiste à 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.

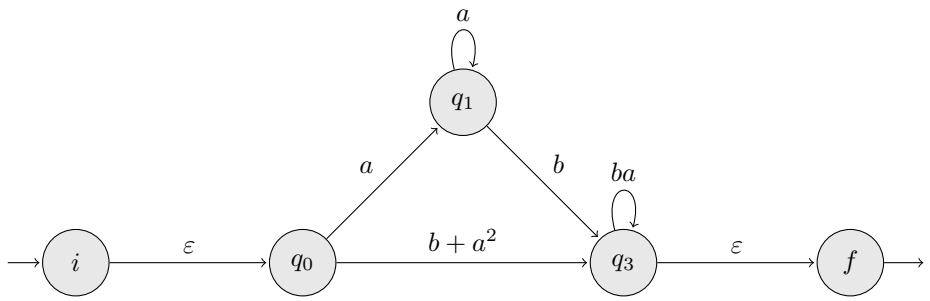
Considérons l'automate suivant :



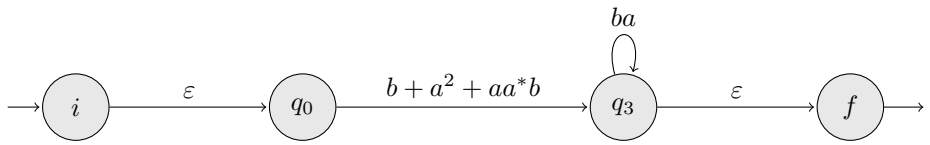
Nous commençons par ajouter un état initial et un état final :



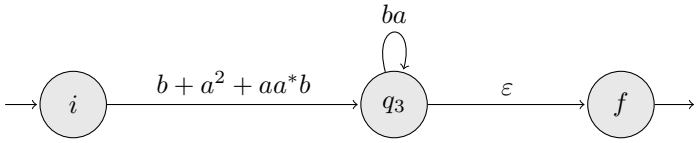
Nous éliminons l'état q_2 :



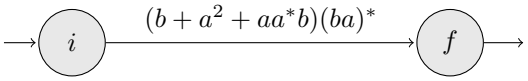
Nous éliminons l'état q_1 :



Nous éliminons l'état q_0 :



Nous éliminons l'état q_3 :



Nous avons ainsi 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.

4.5 Propriétés des langages rationnels

Propriété 11 (Stabilité des langages reconnaissables)

Soit Σ un alphabet fini.

(1) Soit L un langage reconnaissable par un automate $A = (\Sigma, Q, q_0, F, \delta)$.
Alors $\overline{L} = \Sigma^* \setminus L$ est un langage reconnaissable par l'automate $A' = (\Sigma, Q, q_0, Q \setminus F, \delta)$.

(2) Soient L_1 et L_2 deux langages reconnaissables respectivement par un automate $A_1 = (\Sigma, Q_1, q_{1,0}, F_1, \delta_1)$ et un automate $A_2 = (\Sigma, Q_2, q_{2,0}, F_2, \delta_2)$.
Alors le langage $L_1 \cap L_2$ est reconnaissable par l'automate $A = (\Sigma, Q, q_0, F, \delta)$ où :

$$Q = Q_1 \times Q_2, \quad q_0 = (q_{1,0}, q_{2,0}), \quad F = F_1 \times F_2,$$

$$\forall q = (q_1, q_2) \in Q, \forall x \in \Sigma, \quad \delta((q_1, q_2), x) = (\delta_1(q_1, x), \delta_2(q_2, x)).$$

(3) Soit L un langage reconnaissable par l'automate non déterministe $A = (\Sigma, Q, I, F, \delta)$.
Alors le miroir de L (c'est-à-dire le langage constitué des miroirs des mots de L) est un langage reconnaissable par l'automate $A' = (\Sigma, Q, F, I, \delta')$ où :

$$\forall (q, q') \in Q^2, \forall x \in \Sigma, \quad \delta'(q, x) = q' \Leftrightarrow \delta(q', x) = q\,.$$

On déduit alors directement de la propriété précédente et du théorème de Kleene la :

Propriété 12 (Stabilité des langages rationnels)

Les langages rationnels sont stables par passage au complémentaire, par intersection et par passage au langage miroir.

Remarques.

1. Rappelons que, par définition, la réunion ou la concaténation de deux langages rationnels est rationnel et que l'étoile de Kleene d'un langage rationnel est rationnel.
   Ainsi, $\mathrm{Rat}(\Sigma)$ est stable par réunion, intersection, passage au complémentaire, concaténation, étoile de Kleene et passage au langage miroir.
2. $\mathrm{Rat}(\Sigma)$ est également stable par différence ensembliste et différence symétrique puisque :
$$L_1 \setminus L_2 = L_1 \cap \overline{L_2} \quad \text{et} \quad L_1 \Delta L_2 = (L_1 \cup L_2) \setminus (L_1 \cap L_2).$$

4.6 Lemme de l'étoile

Nous disposons désormais de deux manières de prouver qu'un langage est rationnel :

- en exhibant une expression rationnelle qui le dénote ;
- en exhibant un automate qui le reconnait.

En revanche, pour montrer qu'un langage n'est pas rationnel, nous utiliserons en général le résultat suivant :

Propriété 13 (Lemme de l'étoile)

Soit L un langage rationnel.
Il existe un entier naturel k tel que tout mot m appartenant à L de longueur supérieure ou égale à k se factorise sous la forme $m = uvw$ avec :

$$|v| \geq 1, \quad |uv| \leq k, \quad \forall n \in \mathbb{N}, uv^n w \in L.$$

Remarques.

1. 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.
2. 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.
3. 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.

□

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

Exercice. Montrer que le langage $\{a^k b^k \mid k \in \mathbb{N}\}$ n'est pas rationnel.

Exercice 12 (★)
Soit L le langage sur l'alphabet $\Sigma = \{a, b\}$ formé des mots où toute occurrence de a est suivie d'une occurrence de b. Le langage L est-il reconnaissable ? Si oui, donner un automate reconnaissant L.

Exercice 13 (★★)
1. Déterminer l'automate de Glushkov associé à l'expression rationnelle $a(a+b)^*a$.
2. Même question avec l'expression rationnelle $(a+c)^*abb + (a+c)^*$.

Exercice 14 (★★)
Soit l'alphabet $\Sigma = \{a, b\}$ et L le langage des mots sur Σ tels que le nombre de a dans les mots de L soit un multiple de 3.

1. Construire un automate reconnaissant L.
2. En déduire une expression rationnelle dénotant L.

Exercice 15 (★)
Soit L le langage sur l'alphabet $\Sigma = \{a, b\}$ formé des mots u n'admettant pas le facteur a^2 et tels que $|u|_a = 0[2]$. Montrer que L est rationnel.

Exercice 16 (★★)
Montrer qu'il existe des langages reconnus par un automate déterministe à plusieurs états finaux qui ne peuvent être reconnus par un automate déterministe à un seul état final.

On pourra considérer le langage a^*b^*.

Exercice 17 (★★)
Un mot sur un alphabet Σ est double s'il contient au moins deux occurrences de chaque lettre présente dans son écriture.

1. Montrer qu'un mot de longueur $2^{|\Sigma|}$ contient un facteur double.
2. Montrer que cette borne est optimale, c'est-à-dire qu'il existe un mot de longueur $2^{|\Sigma|} - 1$ sans facteur double.
3. Le langage des mots doubles est-il rationnel ?

Exercice 18 (★)
On considère l'alphabet $\Sigma = \{a, b\}$. Montrer que les langages suivants ne sont pas rationnels :

- $L_1 = \{u \in \Sigma^* \mid |u|_a = |u|_b\}$;
- $L_2 = \{a^i b^j \mid i < j\}$;
- $L_3 = \{uu \mid u \in \Sigma^*\}$.

Exercice 19 (★★)
Soit A un automate déterministe reconnaissant $\mathscr{L}(A)$. On cherche à construire A_m un automate déterministe reconnaissant $\mathscr{L}(A)$ et ayant un nombre minimal d'états. Pour cela, on identifie dans l'automate A des états équivalents : q_1 et q_2 sont dits équivalents si pour tout mot u, les états q_1 et q_2 sont tous les deux bloquants ou, si $\delta(q_1, u) = q_1'$ et $\delta(q_2, u) = q_2'$, alors q_1' et q_2' sont tous les deux finals ou non finals. Plutôt que de chercher ces états, on détermine si deux états q_1 et q_2 ne sont pas équivalents, à l'aide de deux règles :

1. l'un est final et l'autre pas ;
2. si les deux états sont non équivalents, et si $\sigma \in \Sigma$ est tel que $\delta(q_1', \sigma) = q_1$ et $\delta(q_2', \sigma) = q_2$ alors q_1' et q_2' ne sont pas équivalents.

On construit donc une partition en classes d'équivalences d'états, initialisée avec la première règle, et affinée avec la deuxième jusqu'à stabilité.

Appliquer cet algorithme (dit algorithme de Hopcroft) à l'automate suivant :