Automates

Quentin Fortier

January 26, 2023

Automate (non déterministe)

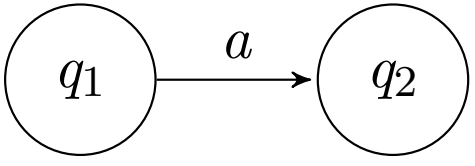
Définition

Un **automate (non déterministe)** est un 5-uplet (Σ, Q, I, F, E) où :

- Σ est un alphabet
- Q est un ensemble fini d'**états**
- $I \in Q$ est un ensemble d'**états initiaux**
- $F \subseteq Q$ est un ensemble d'**états acceptants** (ou **états finaux**)
- $E \subseteq Q \times \Sigma \times Q$ est un ensemble de **transitions**

On peut voir un automate comme un graphe orienté, où :

- Q est l'ensemble des sommets
- une transition $(q_1, a, q_2) \in E$ est un arc étiqueté par une lettre a, représenté par :

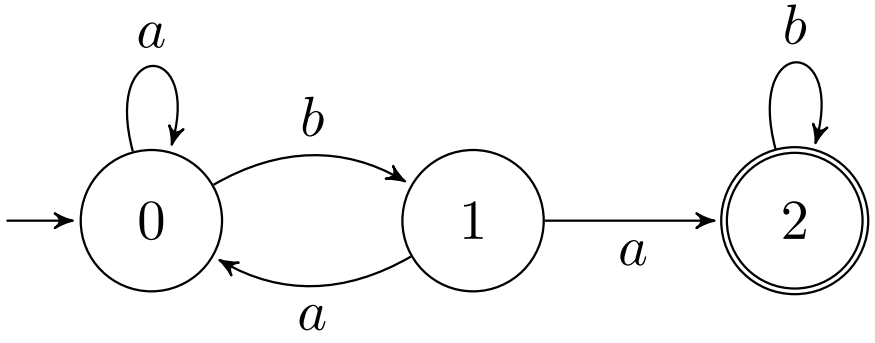


Automate (non déterministe)

Soit $A = (\Sigma, Q, I, F, E)$ où :

- $\Sigma = \{a, b\}$
- $Q = \{0, 1, 2\}$
- $I = \{0\}$
- $F = \{2\}$
- $E = \{(0, a, 0), (0, b, 1), (1, a, 0), (1, a, 2), (2, b, 2)\}$

A est représenté par :



Automate (non déterministe)

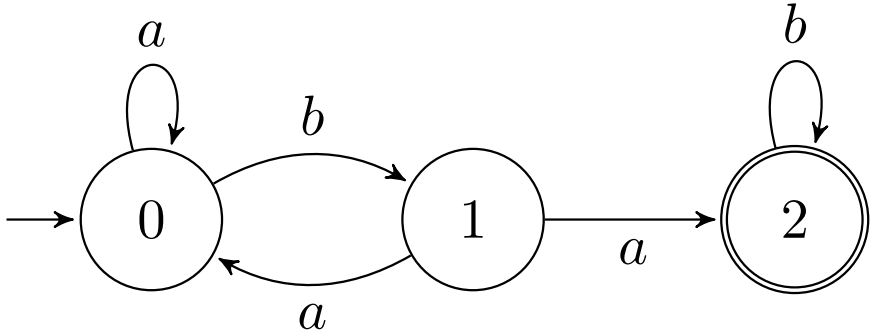
Définition

On peut remplacer l'ensemble E de transitions par une **fonction de transition** $\delta : Q \times \Sigma \longrightarrow \mathcal{P}(Q)$ telle que :

$$\delta(q, a) = \{q' \in Q \mid (q, a, q') \in E\}$$

Question

Donner la fonction de transition pour l'automate suivant.



Implémentation

Pour implémenter la fonction de transition δ, on peut utiliser un dictionnaire.
Le module **Hashtbl** permet d'utiliser un dictionnaire implémenté par table de hachage :

```
let d = Hashtbl.create 42;;
(* 42 est la taille initiale du tableau de la table de hachage *)
Hashtbl.add d 1 "un";; (* ajoute la clé 1 avec la valeur "un" *)
Hashtbl.add d 3 "trois";;
Hashtbl.find d 1;; (* renvoie la valeur associée à la clé 1 *)
Hashtbl.find d 2;; (* exception *)
Hashtbl.mem d 1;; (* renvoie true *)
Hashtbl.mem d 2;; (* renvoie false *)
```

Implémentation

On suppose que les états sont numérotés à partir de 0 et que les lettres sont des `char`.

Question

Proposer un type d'automate en OCaml.

On peut représenter un automate par le type suivant :

```
type automate = {
    initiaux : int list;
    finaux : int list;
    delta : (int*char, int list) Hashtbl.t
}
```

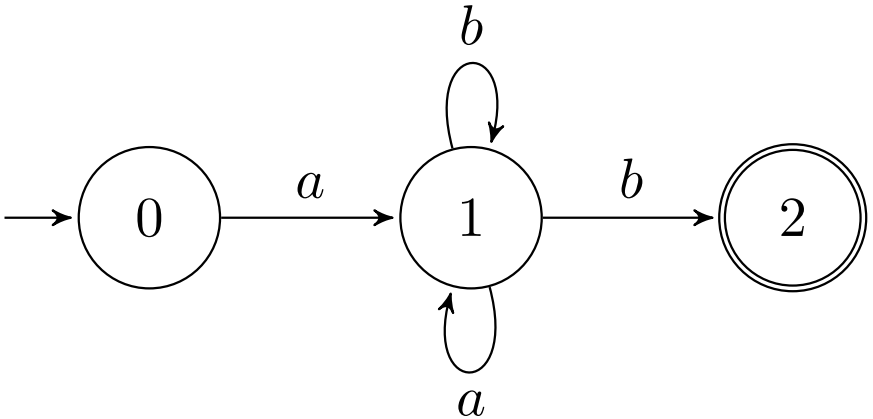
`delta` est donc un dictionnaire qui à chaque clé (q, a) associe la liste des états accessibles depuis l'état q en lisant la lettre a.

Implémentation

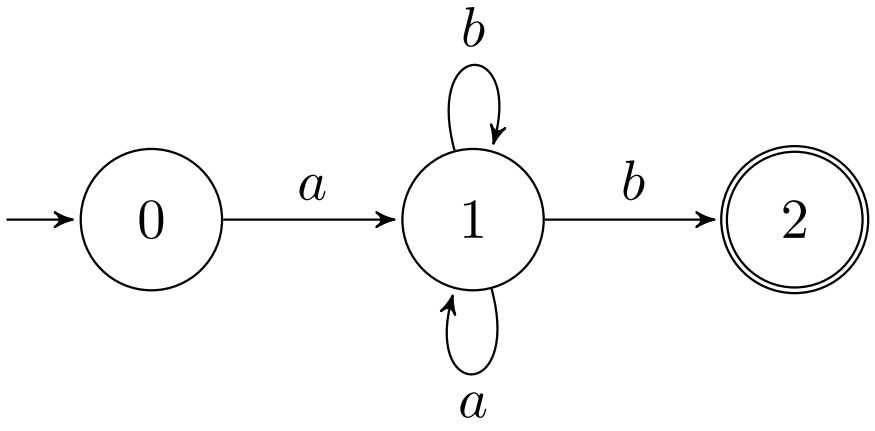
```
type automate = {
    initiaux : int list;
    finaux : int list;
    delta : (int*char, int list) Hashtbl.t
}
```

Question

Définir l'automate ci-dessous avec ce type.



Implémentation




```
let d = Hashtbl.create 42;;
Hashtbl.add d (0, 'a') [1];;
Hashtbl.add d (1, 'a') [1];;
Hashtbl.add d (1, 'b') [1; 2];;

let a = {
  initiaux = [0];
  finaux = [2];
  delta = d
}
```

Langage reconnaissable

Définition

Soit A un automate.

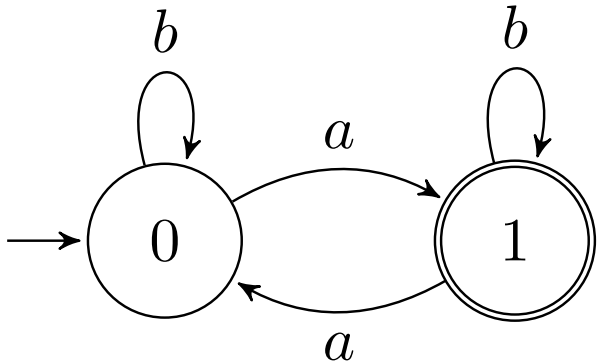
Un **chemin** dans A est une suite de transitions consécutives de la forme :

$$q_0 \xrightarrow{a_1} q_1 \xrightarrow{a_2} q_2 \xrightarrow{a_3} \dots \xrightarrow{a_n} q_n$$

L'**étiquette** de ce chemin est le mot $a_1 a_2 \dots a_n$.

Ce chemin est **acceptant** si $q_0 \in I$ et $q_n \in F$.

Exemple :



$0 \xrightarrow{b} 0 \xrightarrow{b} 0 \xrightarrow{a} 1 \xrightarrow{b} 1$ est un chemin acceptant, d'étiquette $bbab$.

Langage reconnaissable

Définition

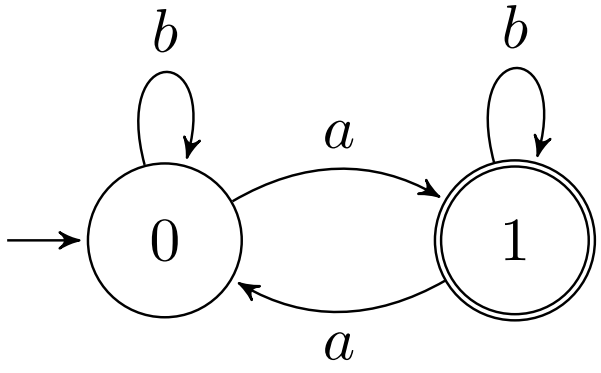
Soit A un automate.

1. Un mot est **accepté** par A s'il est l'étiquette d'un chemin acceptant.
2. Le **langage** $L(A)$ **accepté** (ou **reconnu**) par A est l'ensemble des mots acceptés par A.

Langage reconnaissable

Exercice

Quel est le langage reconnu par l'automate A ci-dessous ?



Un chemin est acceptant ssi il passe par un nombre impair de a, donc $L(A) = \{$ mot avec un nombre impair de a $\} = L(b^*a(b^*ab^*ab^*)^*)$.

Langage reconnaissable

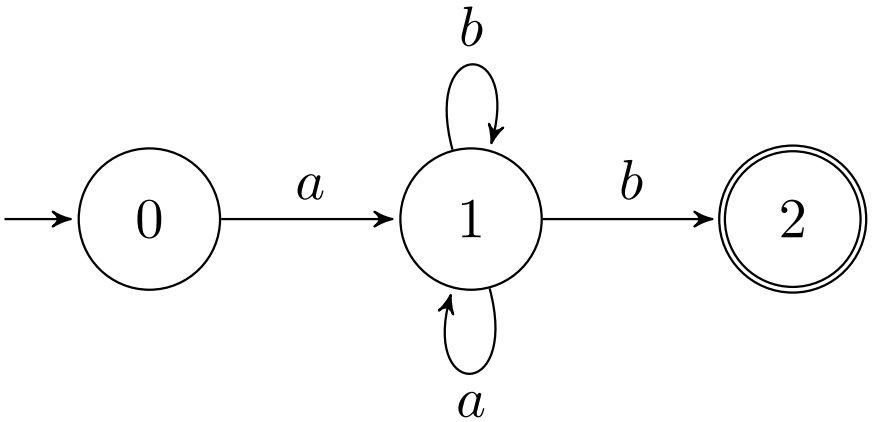
Définition

Un langage L est **reconnaissable** s'il existe un automate A tel que $L = L(A)$.

Remarque : On verra plus tard qu'un langage est reconnaissable ssi il est rationnel.

Question

Montrer que le langage $a(a+b)^*b$ est reconnaissable.



Langage reconnaissable

Exercice

1. Montrer que $ab \mid abc \mid c$ est reconnaissable.
2. Montrer que l'ensemble des mots de longueur paire sur $\Sigma = \{a, b\}$ est reconnaissable.
3. Montrer que $(b \mid ab \mid ba \mid aba)^*$ est reconnnaissable.

Test d'appartenance au langage d'un automate

Question

Comment déterminer informatiquement si un automate A accepte un mot $m = m_1 ... m_n$?

On part de l'ensemble I des états initiaux.
On calcule l'ensemble Q_1 des états accessibles à partir d'un état de I en lisant la lettre m_1.
On calcule l'ensemble Q_2 des états accessibles à partir d'un état de Q_1 en lisant la lettre m_2.
...
On calcule l'ensemble Q_n des états accessibles à partir d'un état de Q_{n-1} en lisant la lettre m_n.
Si Q_n contient un état final (c'est à dire $Q_n \cap F \neq \emptyset$), alors m est accepté par A.

Test d'appartenance au langage d'un automate

Question

Écrire une fonction `etape a etats lettre` qui renvoie la liste des états accessibles depuis la liste `etats` en lisant `lettre`, dans l'automate `a`.

```
let rec etape a etats lettre = match etats with
    | [] -> []
    | e::q -> let t = (e, lettre) in
      if Hashtbl.mem a.delta t
      then (Hashtbl.find a.delta t)@(etape a q lettre)
      else etape a q lettre
```

Remarque : `@` peut introduire des doublons... On pourrait utiliser une structure d'ensemble (`Hashtbl` ou `Set`) à la place de `list`.

Test d'appartenance au langage d'un automate

Question

Écrire une fonction `accepte a (mot : string)` qui détermine si le mot `mot` est accepté par l'automate `a`.

```
let accepte a mot =
  let etats = ref a.initiaux in
  for i = 0 to String.length mot - 1 do
    etats := etape a !etats mot.[i]
  done;
  List.exists (fun e -> List.mem e a.finaux) !etats
```

Automate complet

Définition

Deux automates sont **équivalents** s'ils ont le même langage.

Étant donné un automate, il est intéressant de trouver un automate équivalent plus simple (pour simplifier les démonstrations ou la programmation).

Automate complet

Définition

Un automate (Σ, Q, I, F, E) est **complet** si :

$$\forall q \in Q,\ \forall a \in \Sigma,\ \exists (q, a, q') \in E$$

Autrement dit : depuis tout état q et pour chaque lettre $a \in \Sigma$, il existe au moins une transition étiquetée par a (pas de blocage).

Automate complet

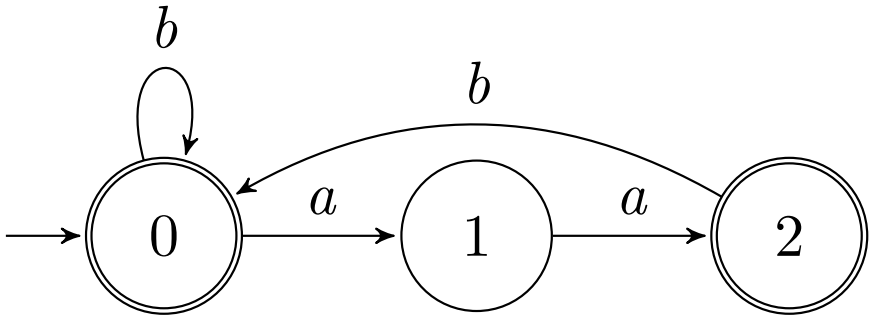
Théorème

Tout automate (Σ, Q, I, F, E) est équivalent à un automate complet.

Preuve :

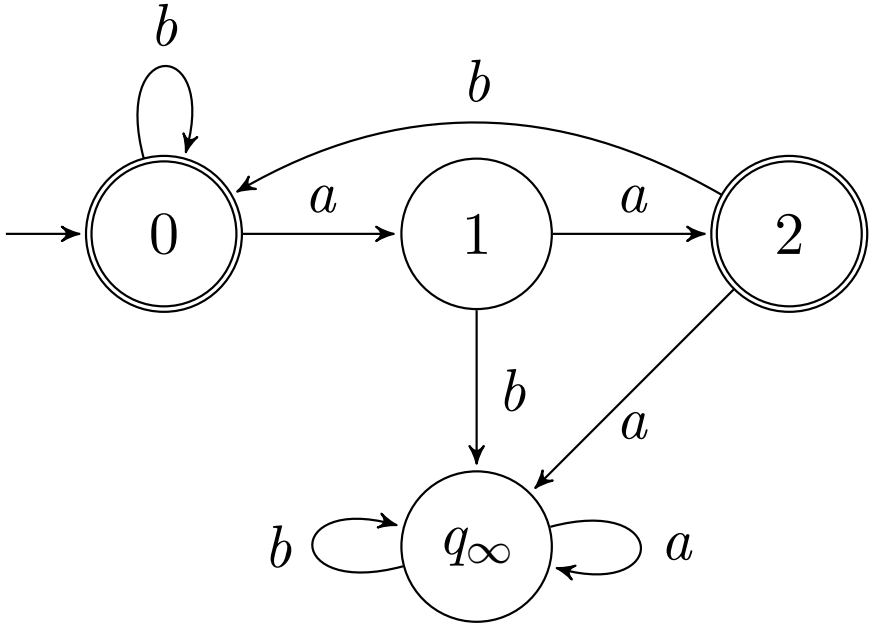
1. On ajoute un état « poubelle » q_∞ à Q
2. Pour toute lettre $a \in \Sigma$, on ajoute une transition $q_\infty \xrightarrow{a} q_\infty$
3. S'il n'y a pas de transition étiquetée par a depuis un état q, on ajoute une transition $q \xrightarrow{a} q_\infty$

Automate complet



Question

Donner un automate complet équivalent à l'automate ci-dessus ($\Sigma = \{a, b\}$).



Automate déterministe

Définition

Un automate (Σ, Q, q_i, F, E) est **déterministe** si :

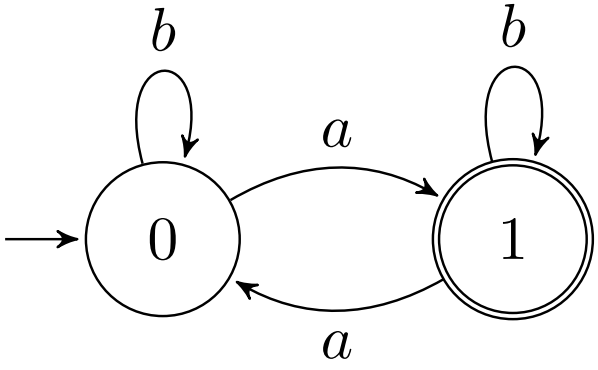
1. Il n'y a qu'un seul état initial q_i.
2. $(q, a, q_1) \in E \wedge (q, a, q_2) \in E \implies q_1 = q_2$: il y a au plus une transition possible en lisant une lettre depuis un état

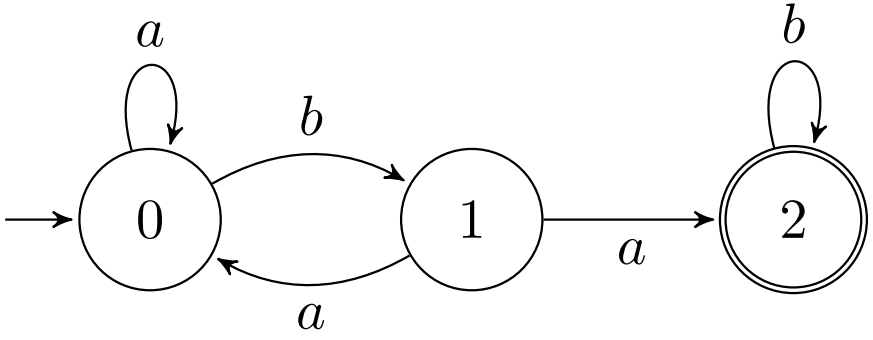
Remarque : si un automate est déterministe et complet alors il existe une unique transition possible depuis un état en lisant une lettre.
La fonction de transition est alors de la forme $\delta : Q \times \Sigma \longrightarrow Q$.

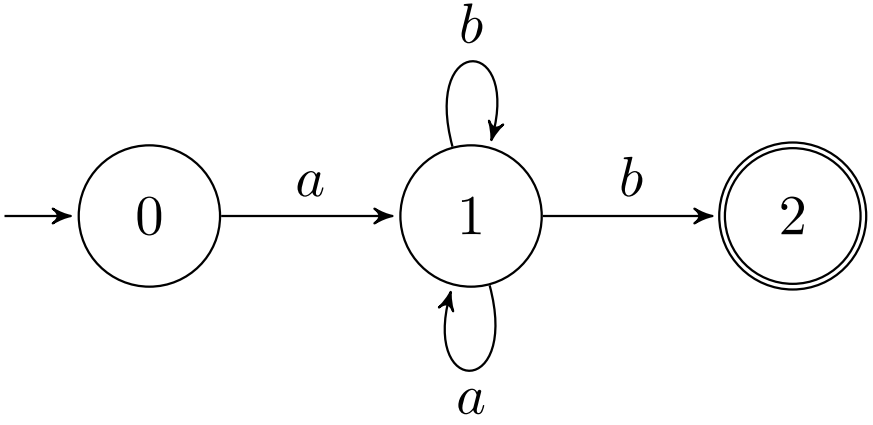
Automate déterministe

Question

Les automates suivants sont-ils déterministes ?







Automate déterministe

Définition

Soit $A = (\Sigma, Q, q_i, F, \delta)$ un automate déterministe complet.
On peut étendre $\delta : Q \times \Sigma \longrightarrow Q$ en une fonction de transition sur les mots $\delta^* : Q \times \Sigma^* \longrightarrow Q$ définie par :

- $\delta^*(q, \varepsilon) = q$
- Si $u = av$, $\delta^*(q, av) = \delta^*(\delta(q, a), v)$

$\delta^*(q, u)$ est l'état auquel on arrive en lisant le mot u depuis l'état q.

On a alors :

$$\delta^*(q_i, u) \in F \iff u \in L(A)$$

Automate déterministe

Les automates déterministes modélisent les **machines à calculer** : un automate prend une entrée (un mot), effectue une suite de transitions en lisant le mot lettre par lettre et renvoie vrai ou faux, suivant que le mot est accepté ou non.

C'est un premier pas vers les **machines de Turing** (constituées d'un automate et d'une mémoire), qui est la définition mathématique d'un ordinateur.

On peut alors prouver que certains problèmes ne peuvent pas être résolus par un algorithme.

Automate déterministe

Question

Donner un type OCaml pour représenter un automate déterministe complet.

```
type afdc = {
    initial : int;
    finaux : int list;
    delta : (int*char, int) Hashtbl.t
}
```

Automate déterministe

Question

Réécrire la fonction `accepte : afdc -> string -> bool` pour un automate déterministe complet.

```
let accepte a mot =
  let etat = ref a.initial in
  for i = 0 to String.length mot - 1 do
    etat := Hashtbl.find a.delta (!etat, mot.[i])
  done;
  List.mem !etat a.finaux
```

Complexité : $O(n)$, où n est la longueur du mot.

Automate déterministe

Théorème

Tout automate $A = (\Sigma, Q, I, F, \delta)$ est équivalent à un automate déterministe complet.

Preuve :
Soit $A' = (\Sigma, \mathcal{P}(Q), I, F', \delta')$ (**l'automate des parties**) tel que :

1. les états de A' sont les sous-ensembles de Q.
2. le seul état initial de A' est $I \subseteq Q$.
3. $F' = \{X \subseteq Q \mid X \cap F \neq \emptyset\}$: les états finaux de A' sont ceux contenant au moins un état final de A.
4. $\forall X \subseteq Q, \forall a \in \Sigma, \delta'(X, a) = \bigcup_{q \in X} \delta(q, a)$: il y a une transition $X \xrightarrow{a} X'$, où X' est l'ensemble des états accessibles depuis un état de X en lisant a.

A' est clairement déterministe et complet.

Automate déterministe

Montrons par récurrence sur la longueur $|m|$ d'un mot m la propriété :

Dans A, il existe un chemin étiqueté par m d'un état de I vers un état q

$\Longleftrightarrow$

Dans A', il existe un chemin étiqueté par m de I vers un état contenant q

La propriété est clairement vraie si $|m| = 0$.

Supposons la propriété vraie pour des mots de longueur $n-1$.
Soit $m = m_1...m_n$ un mot de longueur n.

$\Longrightarrow$ Soit $q_1 \in I \xrightarrow{m_1} q_2 \xrightarrow{m_2} ... \xrightarrow{m_{n-1}} q_n \xrightarrow{m_n} q \in F$ un chemin dans A.
Alors il y a un chemin de q_1 vers q_n d'étiquette $m_1...m_{n-1}$
Par hypothèse de récurrence, $m_1...m_{n-1}$ est l'étiquette d'un chemin de I vers X contenant q_n et $\delta(X, m_n)$ contient q.
Donc $m_1...m_n$ est l'étiquette d'un chemin de I vers un état contenant q.

Automate déterministe

Montrons par récurrence sur la longueur $|m|$ d'un mot m la propriété :

Dans A, il existe un chemin étiqueté par m d'un état de I vers un état q

$$\Longleftrightarrow$$

Dans A', il existe un chemin étiqueté par m de I vers un état contenant q

La propriété est clairement vraie si $|m| = 0$.

Supposons la propriété vraie pour des mots de longueur $n-1$.
Soit $m = m_1 ... m_n$ un mot de longueur n.

$\Longleftarrow$: preuve similaire.

On a donc montré que $\boxed{L(A) = L(A')}$.

Automate déterministe

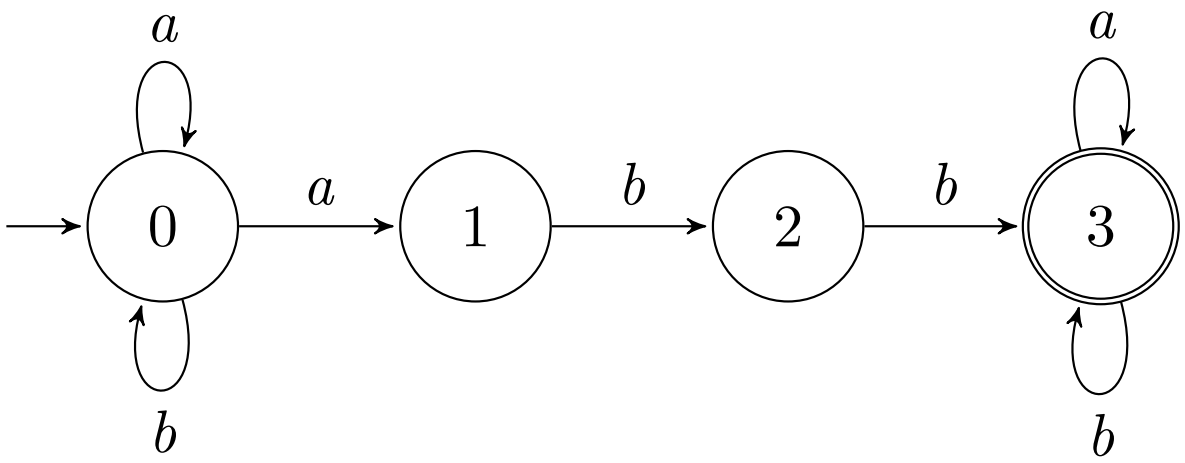
On peut construire le déterminisé de proche en proche, par parcours de l'automate des parties :

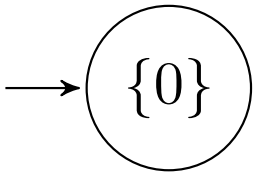
Algorithme de déterminisation

Entrée : Automate $A = (\Sigma, Q, I, F, \delta)$
Sortie : Automate déterministe $A' = (\Sigma, \mathcal{P}(Q), I, F', \delta')$

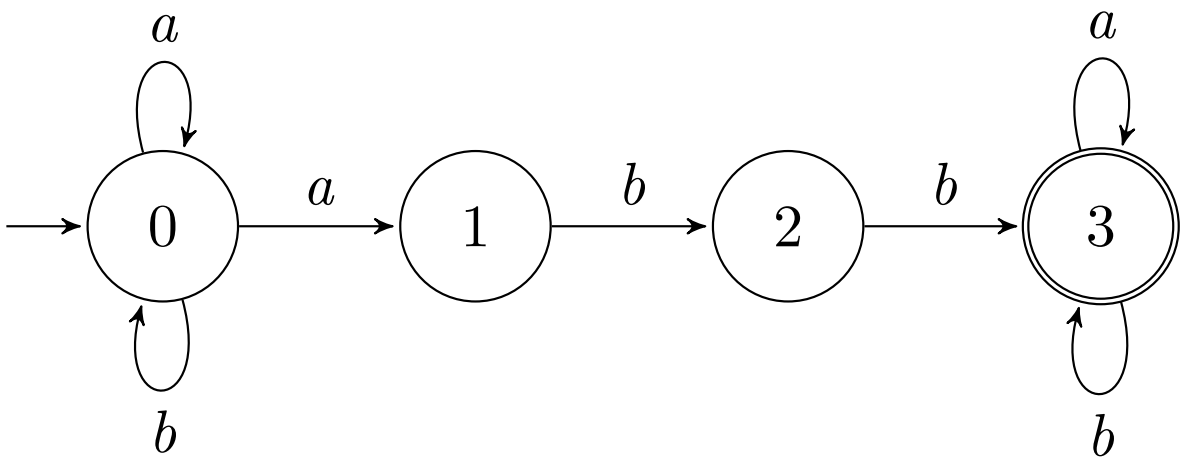
next $\leftarrow \{I\}$
Tant que next $\neq \emptyset$ **:**
  Extraire un élément X de `next`
  Pour $a \in \Sigma$ **:**
    $X' \leftarrow$ ensemble des états accessibles depuis un état de X en lisant a
    Si X' n'a pas déjà été visité **:**
      Ajouter une transition $X \xrightarrow{a} X'$ à A'
      Ajouter X' à `next`

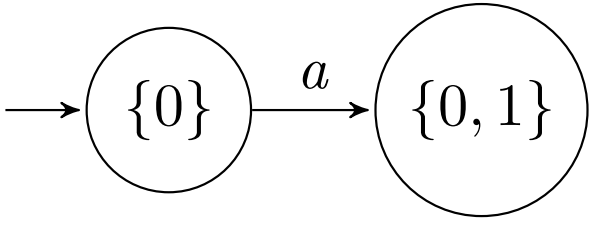
Automate déterministe



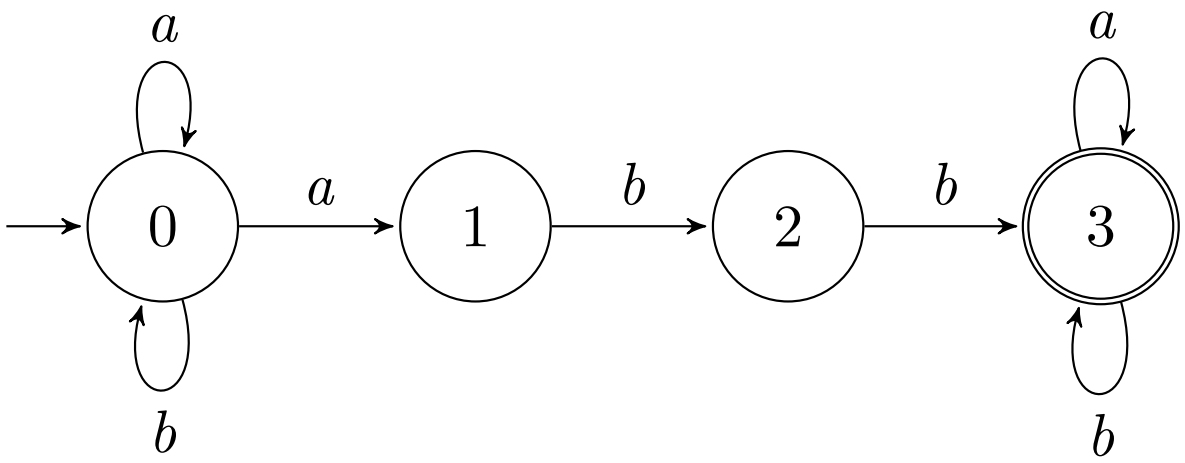


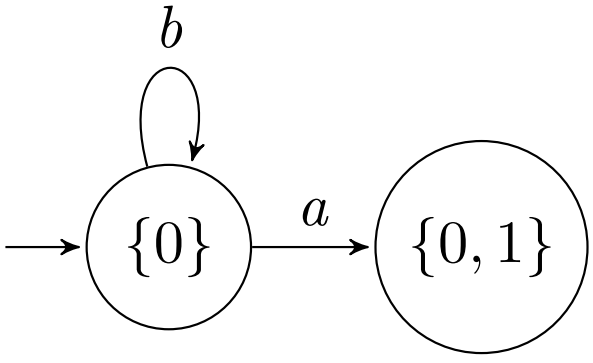
Automate déterministe



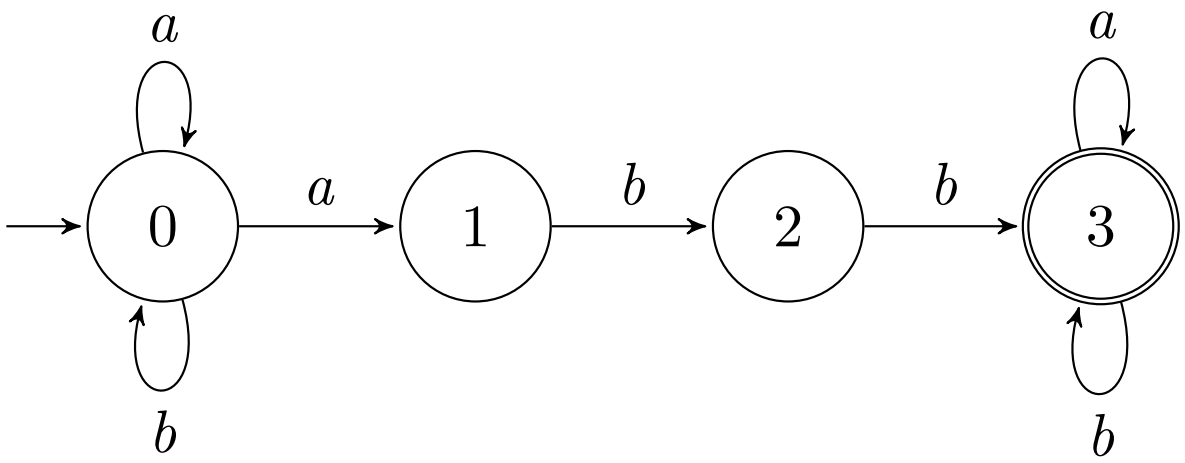


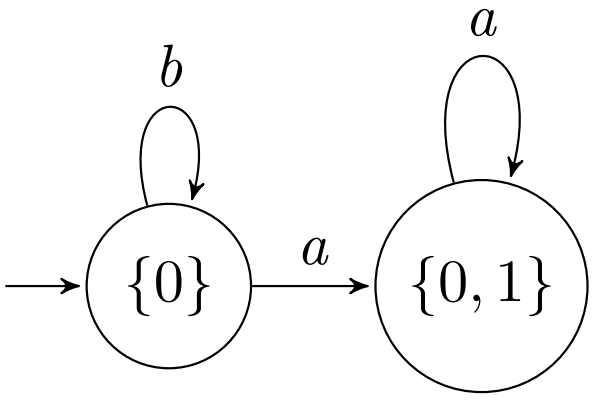
Automate déterministe



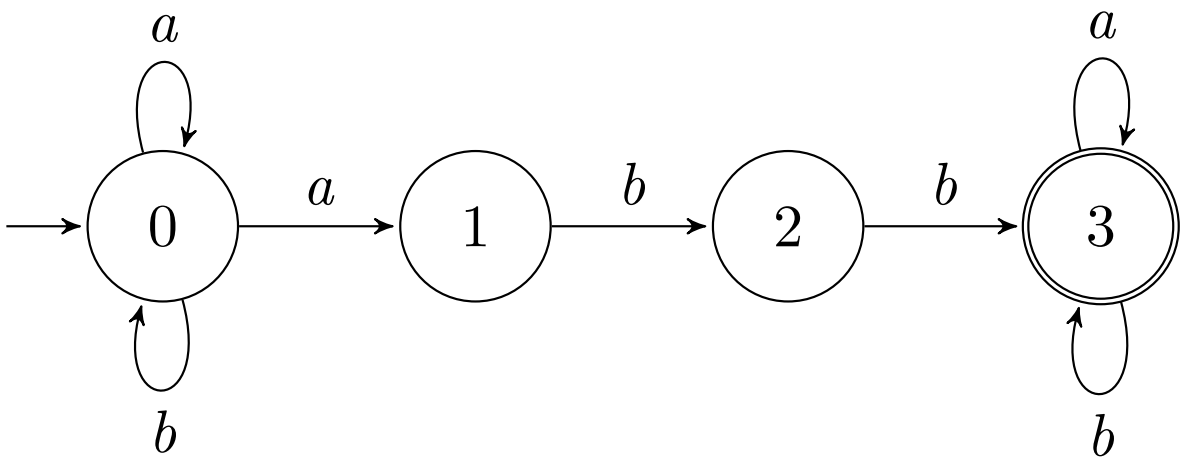


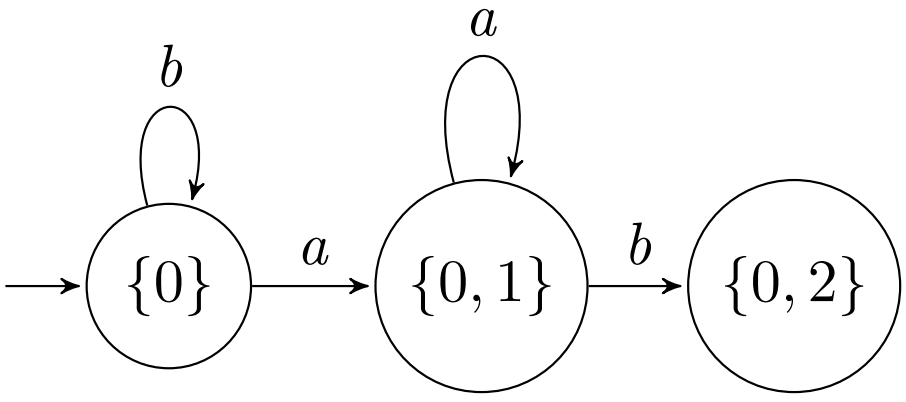
Automate déterministe



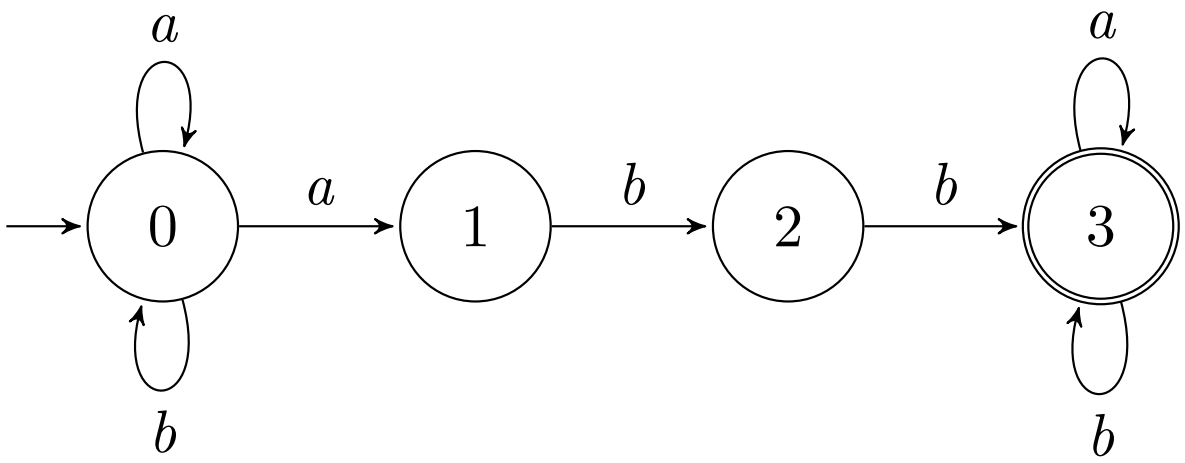


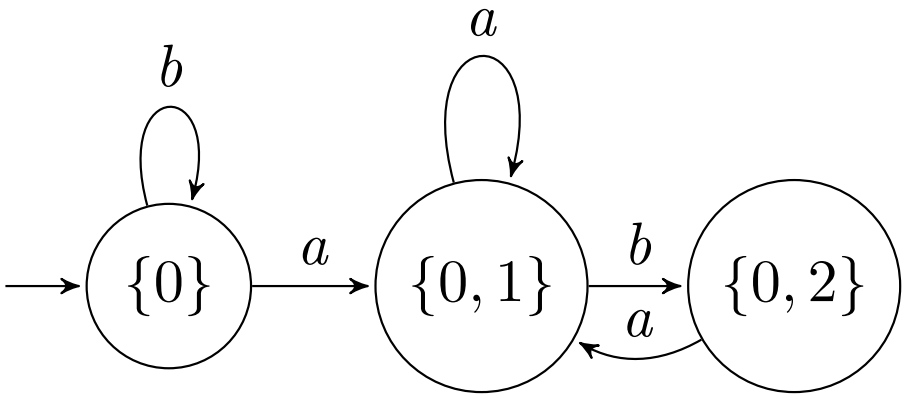
Automate déterministe



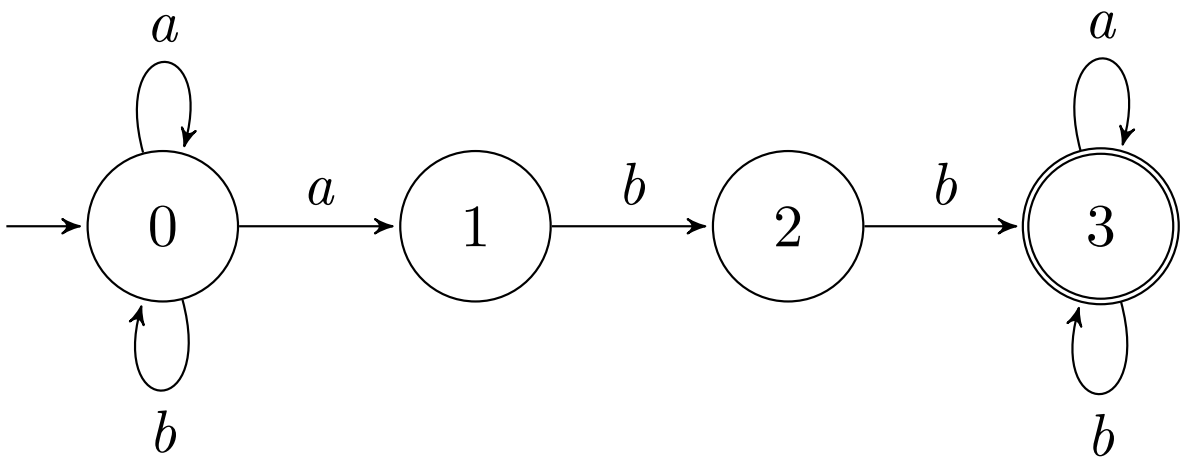


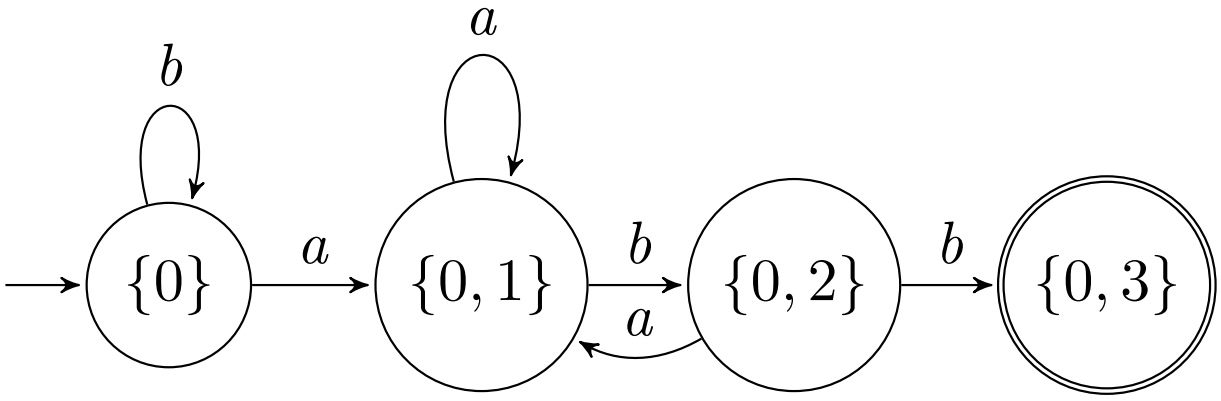
Automate déterministe



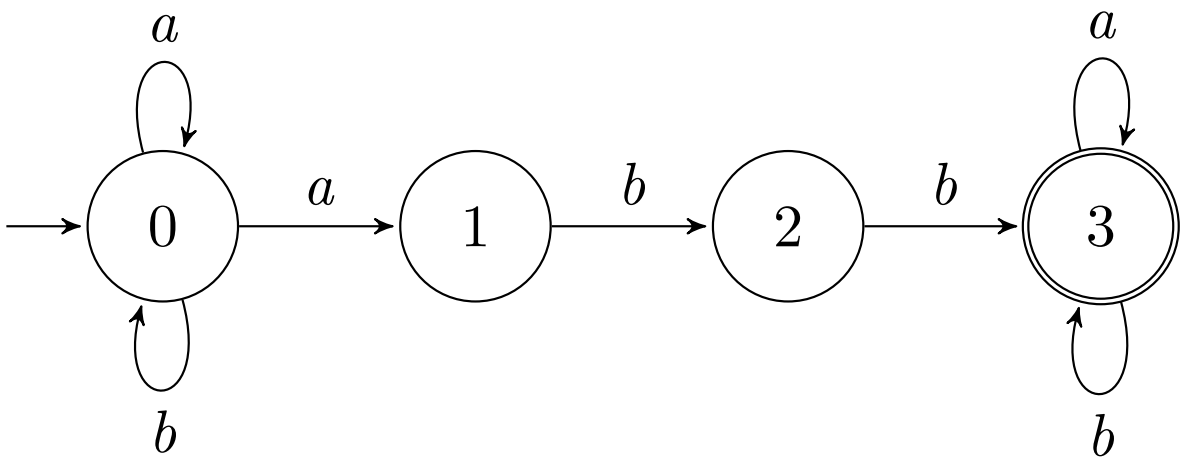


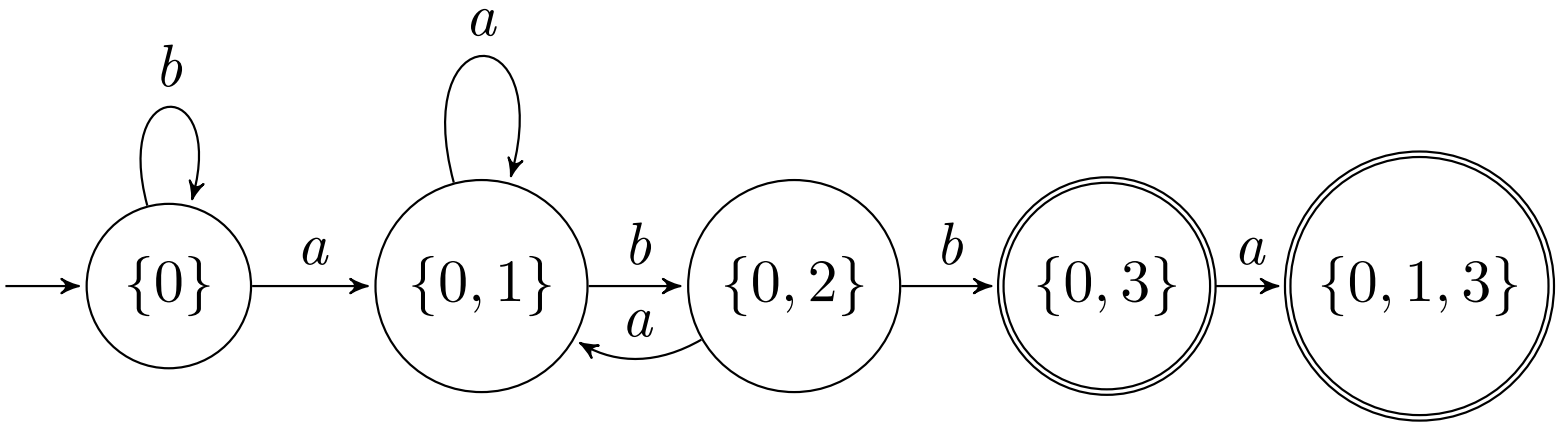
Automate déterministe



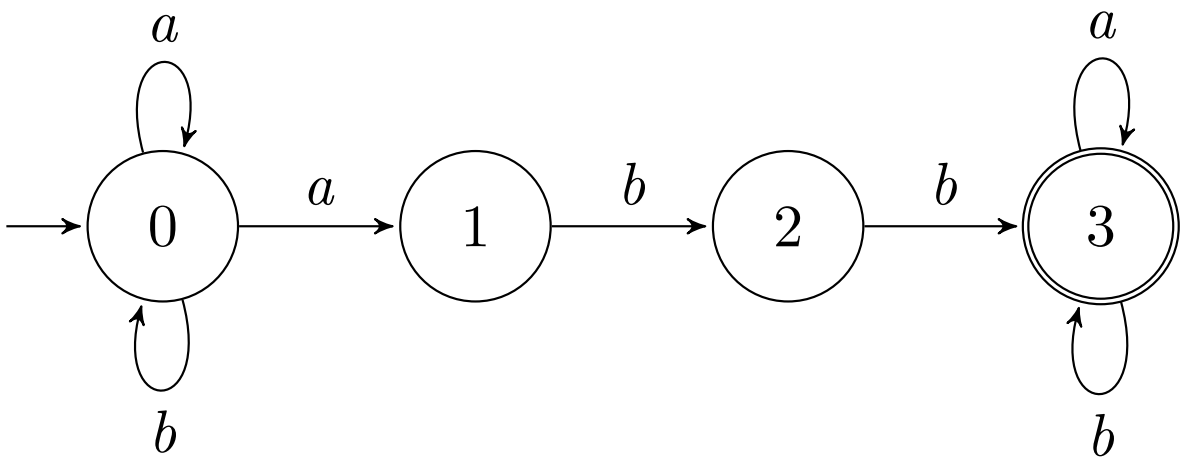


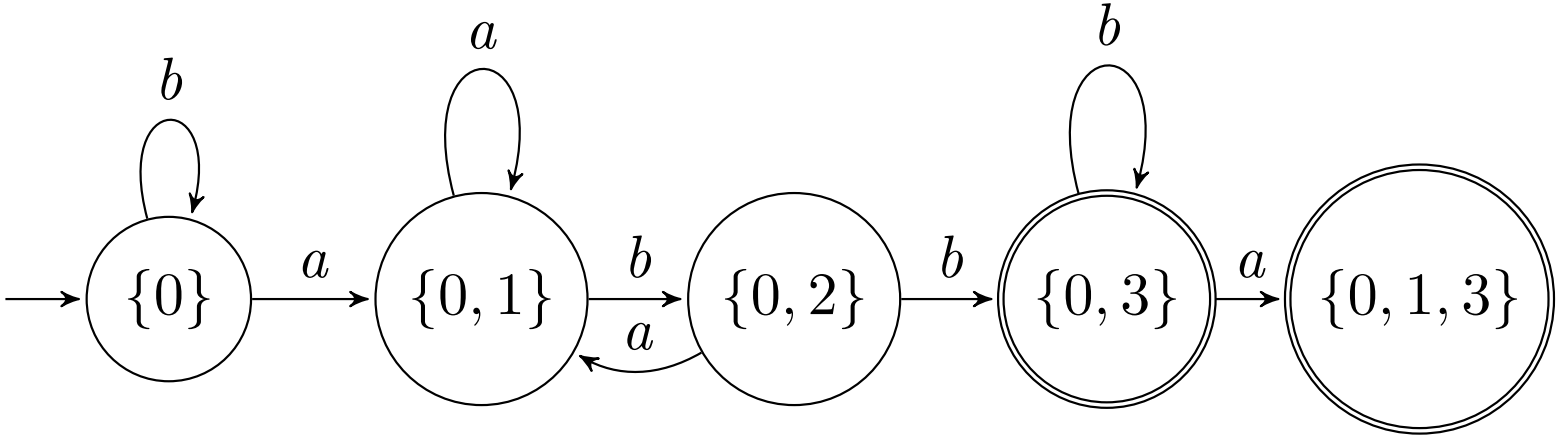
Automate déterministe



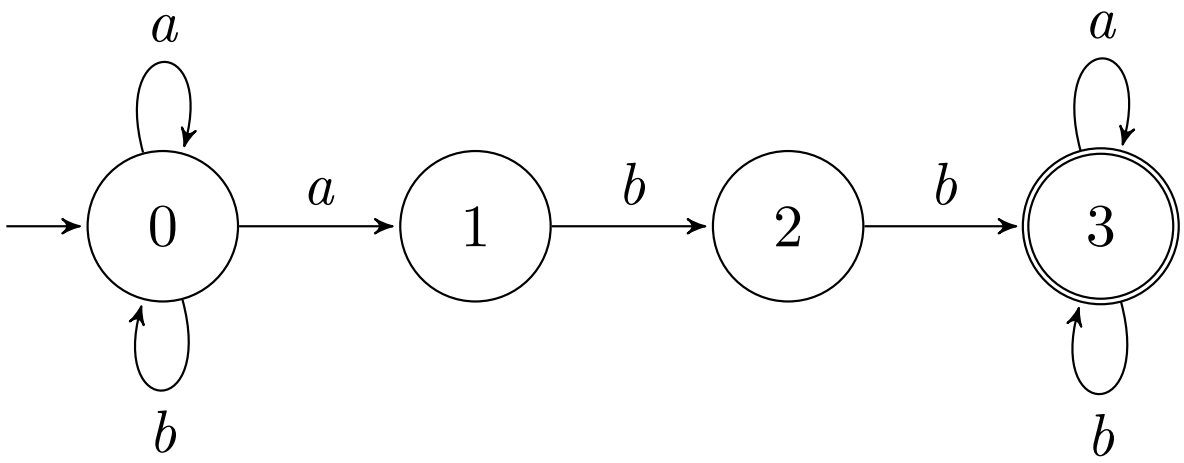


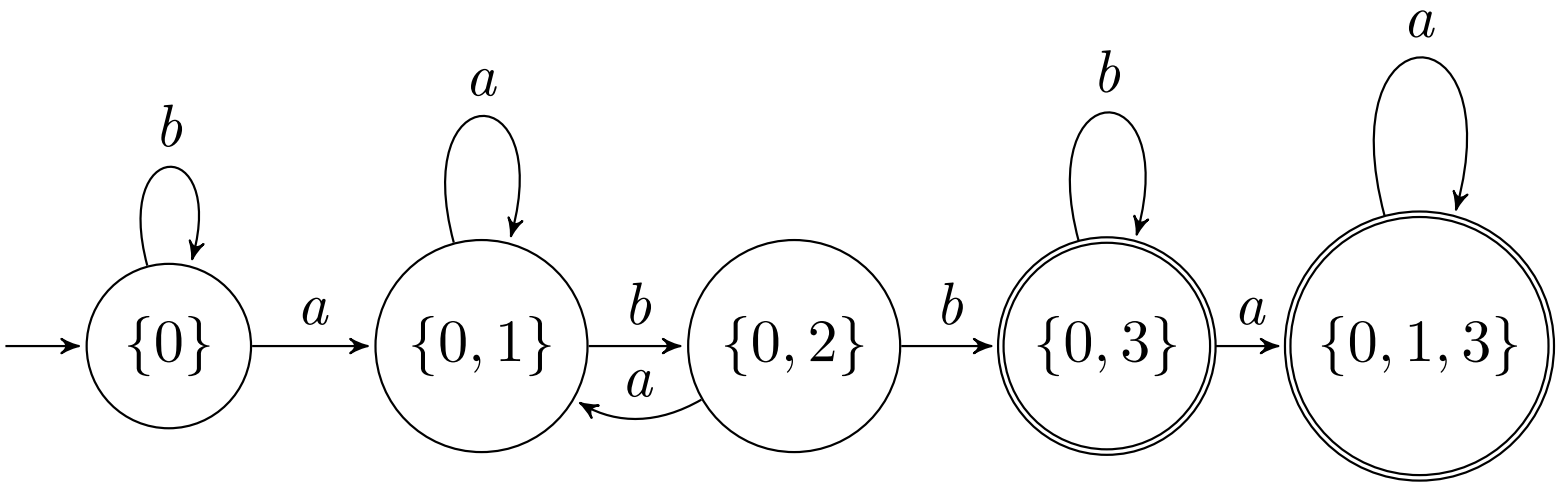
Automate déterministe



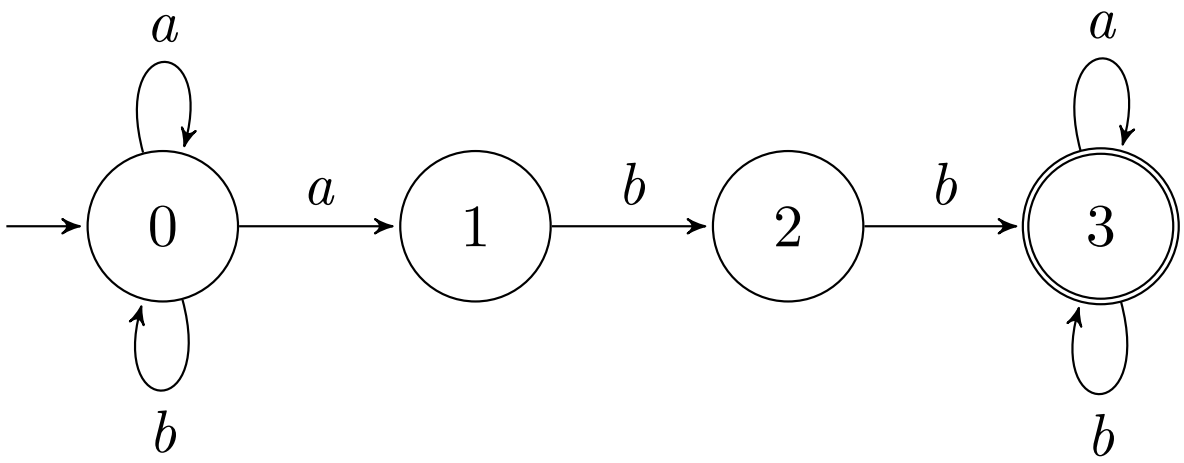


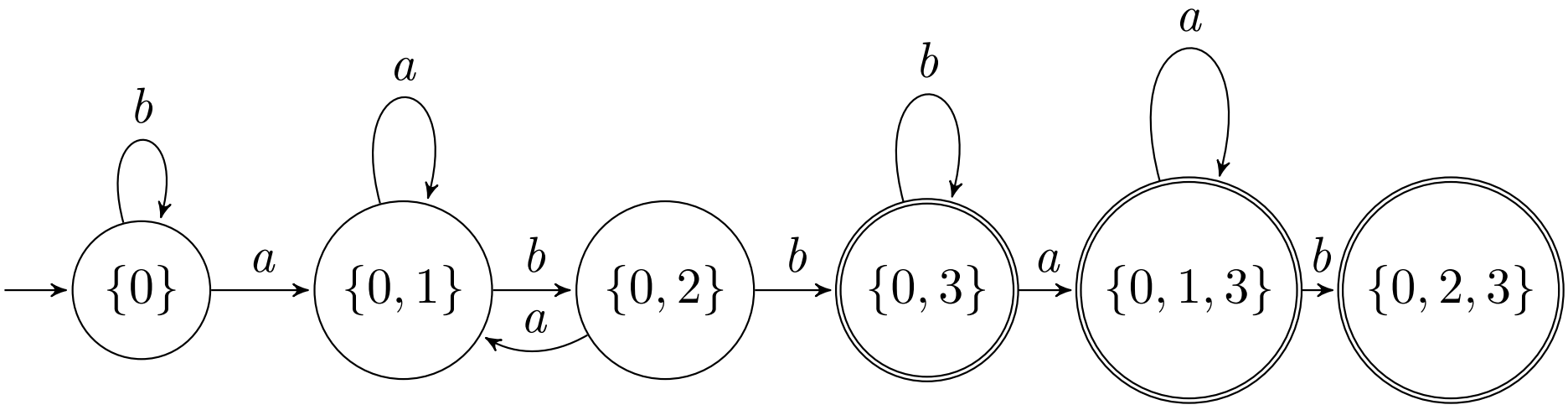
Automate déterministe



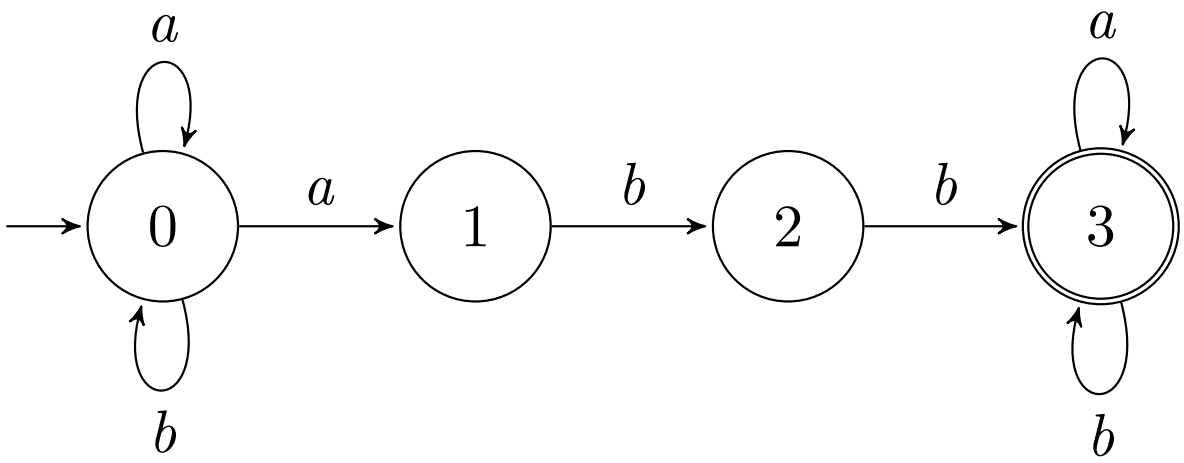


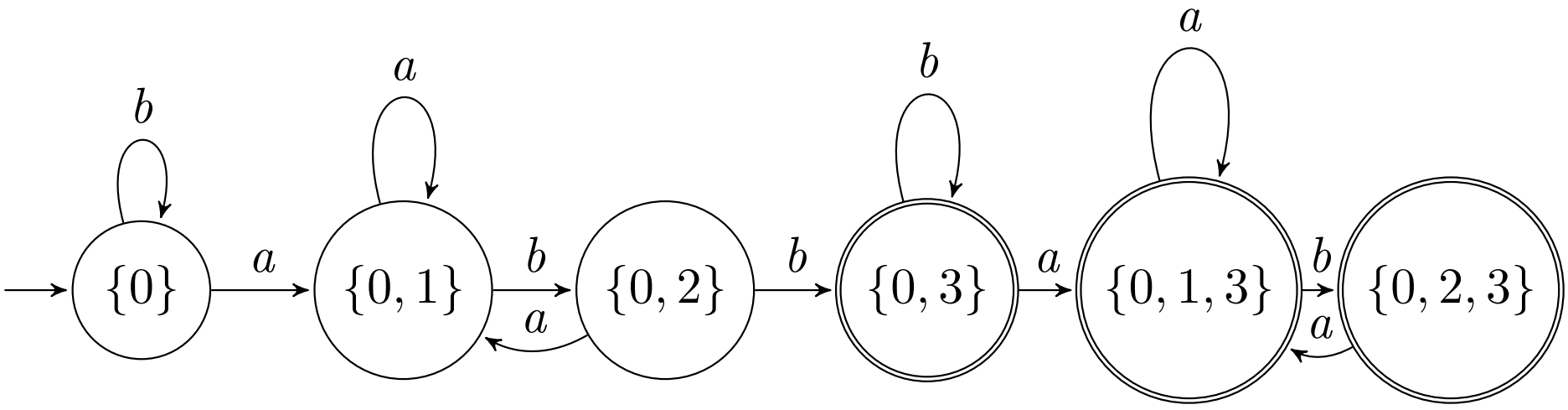
Automate déterministe



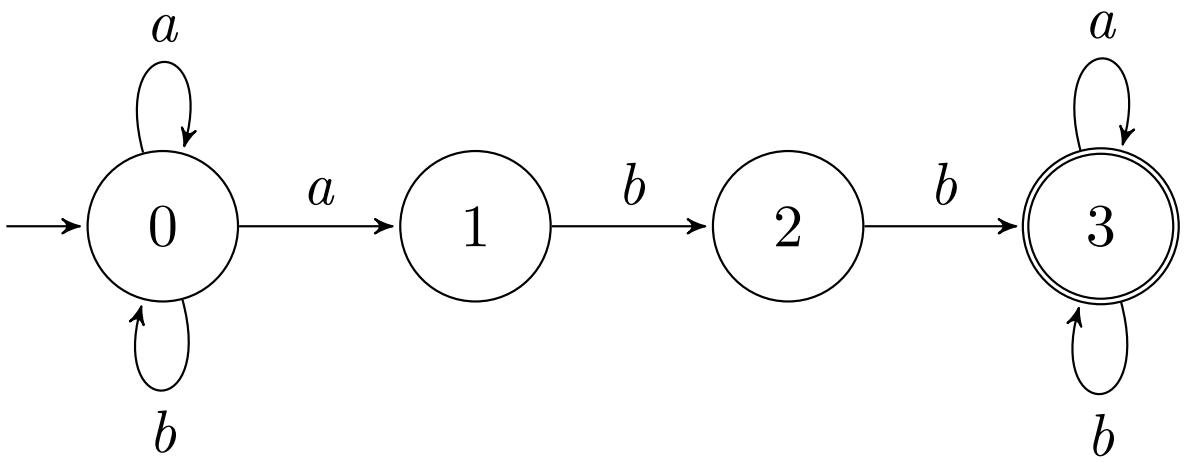


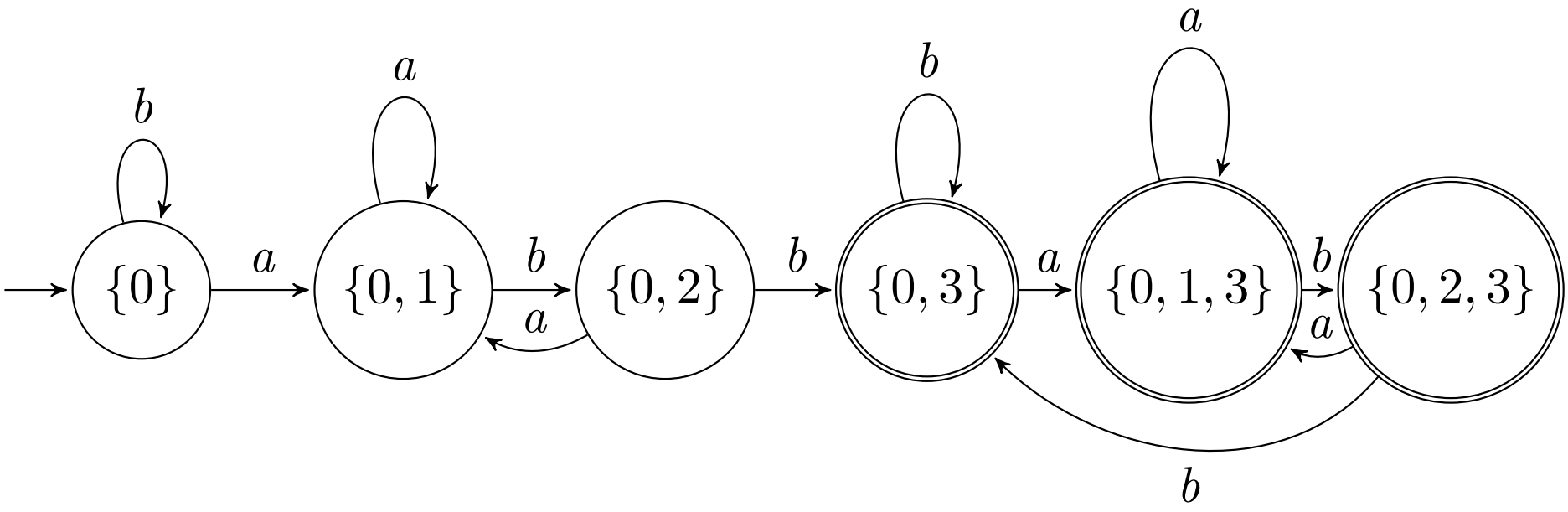
Automate déterministe





Automate déterministe





Automate déterministe

L'automate déterministe complet A' de la preuve précédente possède :

1. $2^{|Q|}$ états.
2. $2^{|Q|} \times |\Sigma|$ transitions.

La construction demande une complexité exponentielle en $|Q|$, mais on a besoin de le faire qu'une seule fois.
Ensuite, savoir si un mot m appartient à $L(A')$ se fait en O($|m|$).

Stabilité des langages reconnaissables

Théorème

Soit L un langage reconnaissable, sur un alphabet Σ.
Alors $\overline{L}$ $(= \Sigma^* \backslash L)$ est reconnaissable.

Preuve :
Soit $A = (\Sigma, Q, q_i, F, \delta)$ un automate déterministe complet reconnaissant L.
Soit $A' = (\Sigma, Q, q_i, Q \backslash F, \delta)$ (on inverse états finaux et non-finaux).
Comme A est déterministe :

$$u \in \overline{L(A)} \iff \delta^*(q_i, u) \notin F \iff \delta^*(q_i, u) \in Q \backslash F \iff u \in L(A')$$

Donc $L(A') = \overline{L(A)}$.

Stabilité des langages reconnaissables

Exercice

On utilise l'alphabet $\Sigma = \{a, b\}$.

1. Dessiner un automate reconnaissant les mots ayant aaa comme facteur.
2. En déduire un automate reconnaissant les mots n'ayant pas aaa comme facteur.

Stabilité des langages reconnaissables

Soient $A_1 = (Q_1, q_1, F_1, \delta_1)$ et $A_2 = (Q_2, q_2, F_2, \delta_2)$ deux automates finis déterministes complets sur un même alphabet.

Définition

On appelle **automate produit** un automate de la forme $(Q_1 \times Q_2, (q_1, q_2), F, \delta)$ où :

- l'ensemble d'états est le produit cartésien $Q_1 \times Q_2$
- l'état initial est (q_1, q_2)
- la fonction de transition δ est définie par $\delta((q, q'), a) = (\delta_1(q, a), \delta_2(q', a))$
- un certain ensemble d'états finaux F.

Stabilité des langages reconnaissables

Théorème

Si L_1 et L_2 sont reconnaissables alors :

- $L_1 \cap L_2$ est reconnaissable.
- $L_1 \cup L_2$ est reconnaissable.
- $L_1 \backslash L_2$ est reconnaissable.

Preuve : soient $A_k = (\Sigma, Q_k, i_k, F_k, \delta_k)$ des automates déterministes complets reconnaissants L_k $(k \in \{1, 2\})$ et
$A_1 \times A_2 \stackrel{\text{def}}{=} (\Sigma, Q_1 \times Q_2, (i_1, i_2), F, \delta)$ où
$\delta((q_1, q_2), a) = (\delta_1(q_1, a), \delta_2(q_2, a))$

- Si $F = F_1 \times F_2$: $A_1 \times A_2$ reconnaît $L_1 \cap L_2$.
- Si $F = \{(q_1, q_2) \mid q_1 \in F_1 \text{ ou } q_2 \in F_2\}$: $A_1 \times A_2$ reconnaît $L_1 \cup L_2$.
- Si $F = \{(q_1, q_2) \mid q_1 \in F_1 \text{ et } q_2 \notin F_2\}$: $A_1 \times A_2$ reconnaît $L_1 \backslash L_2$.

Stabilité des langages reconnaissables

Question

Dessiner un automate reconnaissant les mots sur $\Sigma = \{a, b\}$ contenant un nombre pair de a et un nombre de b égal à 2 modulo 3.

États accessibles et co-accessibles

Définition

Soit $A = (\Sigma, Q, I, F, \delta)$ un automate et $q \in Q$.

1. q est **accessible** s'il existe un chemin depuis un état initial vers q.
2. q est **co-accessible** s'il existe un chemin depuis q vers un état final.

Question

Donner un algorithme en complexité linéaire pour déterminer tous les états accessibles/co-accessibles d'un automate.

États accessibles et co-accessibles

Définition

Un automate est **émondé** si tous ses états sont accessibles et co-accessibles.

Lemme

Montrer que tout automate est équivalent à un automate émondé.

Lemme de l'étoile

Lemme de l'étoile

Soit L un langage reconnaissable.
Alors il existe $n \in \mathbb{N}$ tel que tout mot u de longueur supérieure à n se décompose en $u = xyz$ avec :

- $|xy| \leq n$
- $y \neq \varepsilon$
- $\forall k \in \mathbb{N},\ xy^k z \in L$

Preuve :
Soit $A = (\Sigma, Q, I, F, \delta)$ un automate reconnaissant L et $n = |Q|$.
Soit $u \in L$ tel que $|u| \geq n$. u est donc l'étiquette d'un chemin acceptant C :

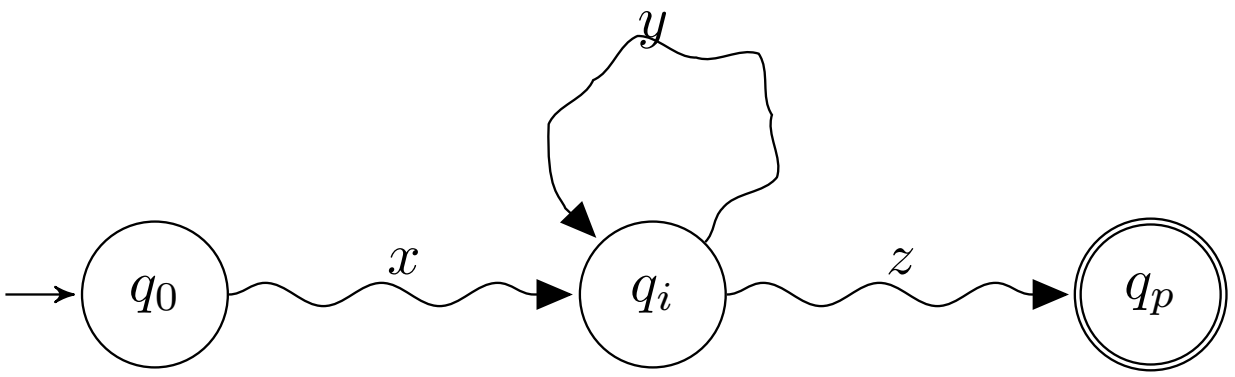
$$q_0 \in I \xrightarrow{u_0} q_1 \xrightarrow{u_1} \ldots \xrightarrow{u_{p-1}} q_p \in F$$

Lemme de l'étoile

Preuve (suite) :

$$C = q_0 \in I \overset{u_0}{\to} q_1 \overset{u_1}{\to} \ldots \overset{u_{p-1}}{\to} q_p \in F$$

Comme $p \geq n$, C a $> n$ sommets donc passe deux fois par un même état $q_i = q_j$ avec $i < n$. La partie de C entre q_i et q_j forme donc un cycle.



Soit $x = u_0 u_1 \ldots u_{i-1}$, $y = u_i \ldots u_j$ et $z = u_{j+1} \ldots u_{p-1}$.
Soit $k \in \mathbb{N}$. Alors $xy^k z$ est l'étiquette du chemin acceptant obtenu à partir de C en passant k fois dans le cycle.

Lemme de l'étoile

Lemme de l'étoile

Soit L un langage reconnaissable.
Alors il existe $n \in \mathbb{N}$ tel que tout mot u de longueur supérieure à n se décompose en $u = xyz$ avec :

- $|xy| \leq n$
- $y \neq \varepsilon$
- $\forall k \in \mathbb{N},\ xy^k z \in L$

Exercice

Montrer que $L_1 = \{a^n b^n \mid n \in \mathbb{N}\}$ n'est pas un langage reconnaissable.

Lemme de l'étoile

Exercice

Montrer que $L_1 = \{a^n b^n \mid n \in \mathbb{N}\}$ n'est pas un langage reconnaissable.

Preuve :
Supposons que L_1 soit reconnaissable.
Soit $n \in \mathbb{N}$ l'entier donné par le lemme de l'étoile.
Soit $u = a^n b^n$. Clairement, $u \in L_1$ et $|u| \geq n$.
Donc u s'écrit $u = xyz$ avec $|xy| \leq n$, $y \neq \varepsilon$ et $\forall k \in \mathbb{N},\ xy^k z \in L$.
Comme $|xy| \leq n$, x et y ne contiennent que des a : $x = a^i$ et $y = a^j$ avec $i + j = n$.
Comme $y \neq \varepsilon$, $j > 0$ et $i < n$.
En prenant $k = 0$: $xy^0 z = xz = a^i b^n \notin L_1$: absurde.

Remarque : On aurait aussi pu prendre $k = 2$.