Automates

MP/MP* Option info

Automate

Définition

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

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

Automate

Définition

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

- Σ est un alphabet (fini)
- Q est un ensemble fini d'**états**
- $I \subseteq Q$ est l'ensemble des **états initiaux**
- $F \subseteq Q$ est un ensemble des **états finaux** (ou acceptants)
- $E \subseteq Q \times \Sigma \times Q$ 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, et représenté par $q_1 \xrightarrow{a} q_2$

Automate

Définition

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

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

À la place de l'ensemble E de transitions, on peut utiliser une **fonction de transition** $\delta : Q \times \Sigma \longrightarrow \mathcal{P}(Q)$.
$\delta(q, a) = \{q' \in Q \mid (q, a, q') \in E\}$ est l'ensemble des états atteignables depuis q en lisant a.

Exemple

Soit $A = (\Sigma, Q, I, F, E)$ où:

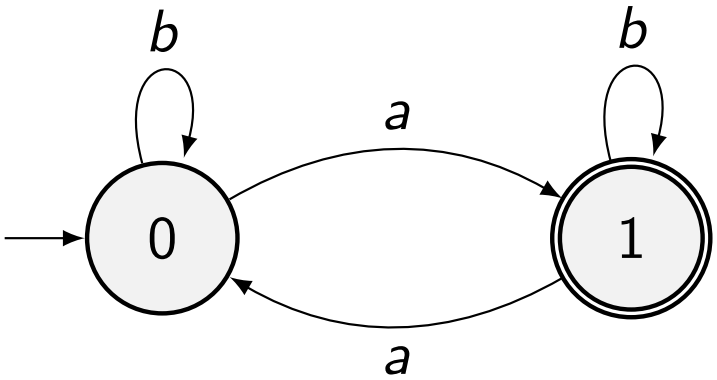
- $\Sigma = \{a, b\}$
- $Q = \{0, 1\}$
- $I = \{0\}$
- $F = \{1\}$
- $E = \{(0, a, 1), (0, b, 0), (1, a, 0), (1, b, 1)\}$

Exemple

Soit $A = (\Sigma, Q, I, F, E)$ où:

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

A est représenté par:



Définition

Un **chemin** dans un automate (Σ, Q, I, F, E) est une suite de transitions consécutives de la forme:

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

Ce chemin est **acceptant** si $q_0 \in I$ et $q_n \in F$.
L'**étiquette** de ce chemin est le mot $a_1 a_2 \ldots a_n$.

Définition

Un **chemin** dans un automate (Σ, Q, I, F, E) est une suite de transitions consécutives de la forme:

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

Ce chemin est **acceptant** si $q_0 \in I$ et $q_n \in F$.
L'**étiquette** de ce chemin est le mot $a_1 a_2 \ldots a_n$.

Définition

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

Définition

Un **chemin** dans un automate (Σ, Q, I, F, E) est une suite de transitions consécutives de la forme:

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

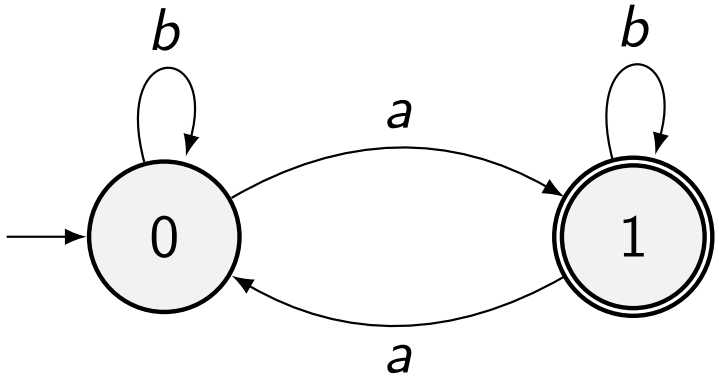
Ce chemin est **acceptant** si $q_0 \in I$ et $q_n \in F$.
L'**étiquette** de ce chemin est le mot $a_1 a_2 \ldots a_n$.

Définition

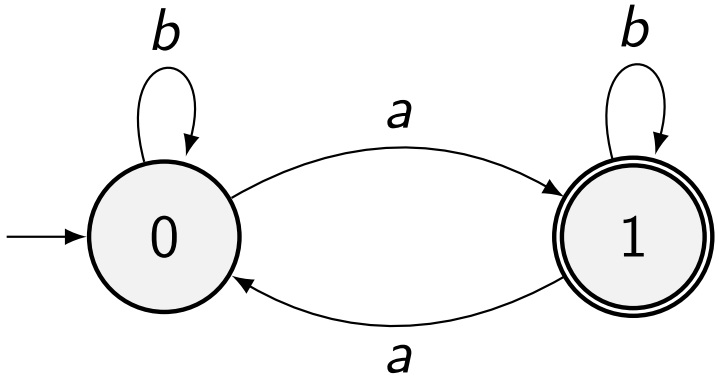
Soit A un automate.
Un mot est **accepté** par A si il est l'étiquette d'un chemin acceptant.
Le **langage** $L(A)$ **accepté** (ou **reconnu**) par A est l'ensemble des mots acceptés par A
Un langage est **reconnaissable** s'il est le langage accepté par un automate

On verra qu'**un langage est reconnaissable ssi il est rationnel**.

Exemple

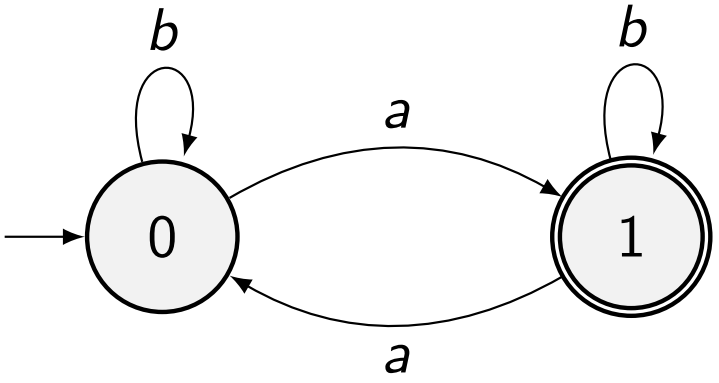


Exemple



$0 \xrightarrow{b} 0 \xrightarrow{b} 0 \xrightarrow{a} 1 \xrightarrow{b} 1$ est un chemin acceptant, donc $bbab \in L(A)$

Exemple



$0 \xrightarrow{b} 0 \xrightarrow{b} 0 \xrightarrow{a} 1 \xrightarrow{b} 1$ est un chemin acceptant, donc $bbab \in L(A)$

Pour montrer $L(A) = \{\text{mot avec un nombre impair de a}\}$, on peut prouver par récurrence sur n que:

- les étiquettes des chemins de 0 à 0 de longueurs n sont exactement les mots de n lettres avec un nombre pair de a
- les étiquettes des chemins de 0 à 1 de longueurs n sont exactement les mots de n lettres avec un nombre impair de a

Exemple

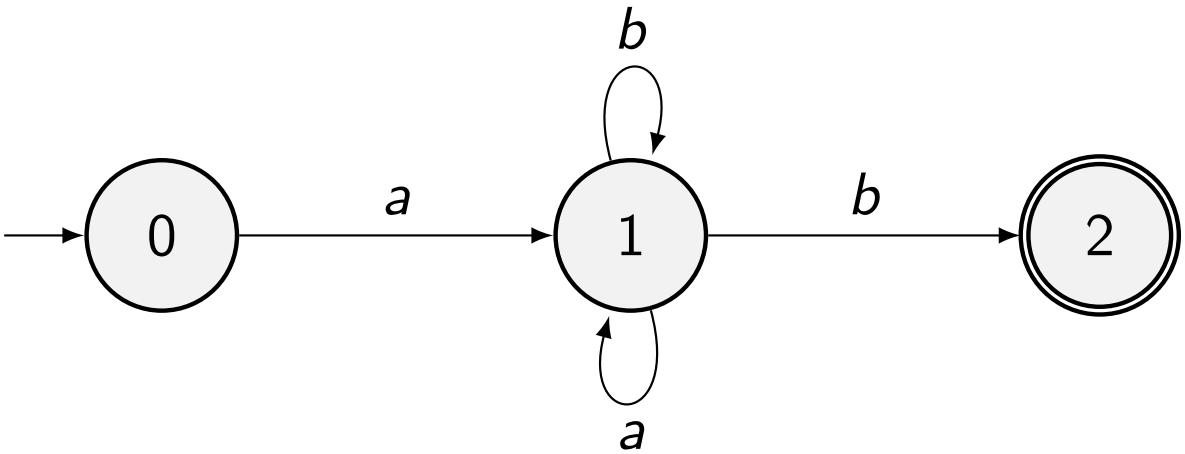
Question

Soit $\Sigma = \{a, b\}$. Donner un automate dont le langage est $a(a + b)^* b$.

Exemple

Question

Soit $\Sigma = \{a, b\}$. Donner un automate dont le langage est $a(a + b)^* b$.



Implémentation

Par soucis de simplicité, on utilise un entier pour chaque état:

```
type 'a automate =
  { initiaux : int list;
    finaux : int list;
    delta : int * 'a -> int list };;
```

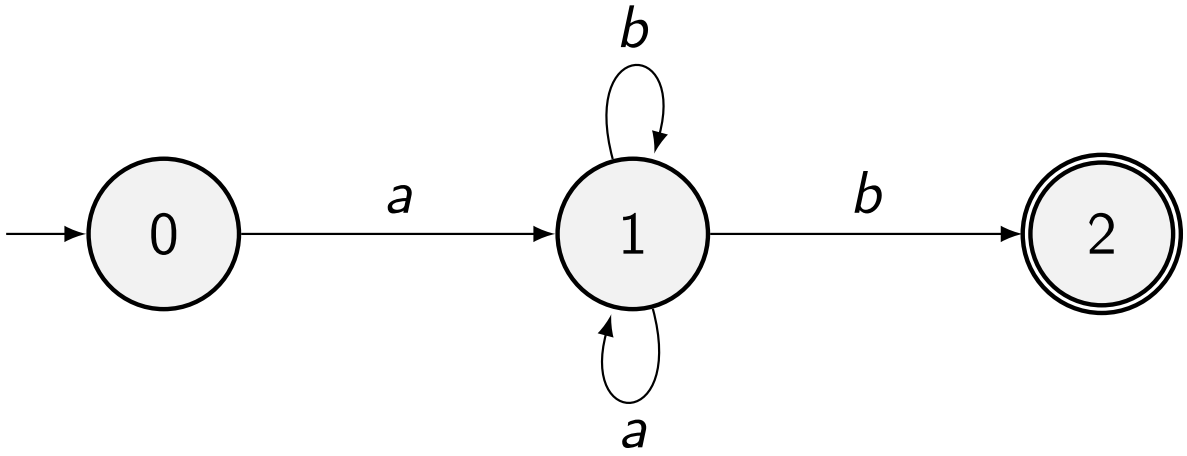
Implémentation

Par soucis de simplicité, on utilise un entier pour chaque état:

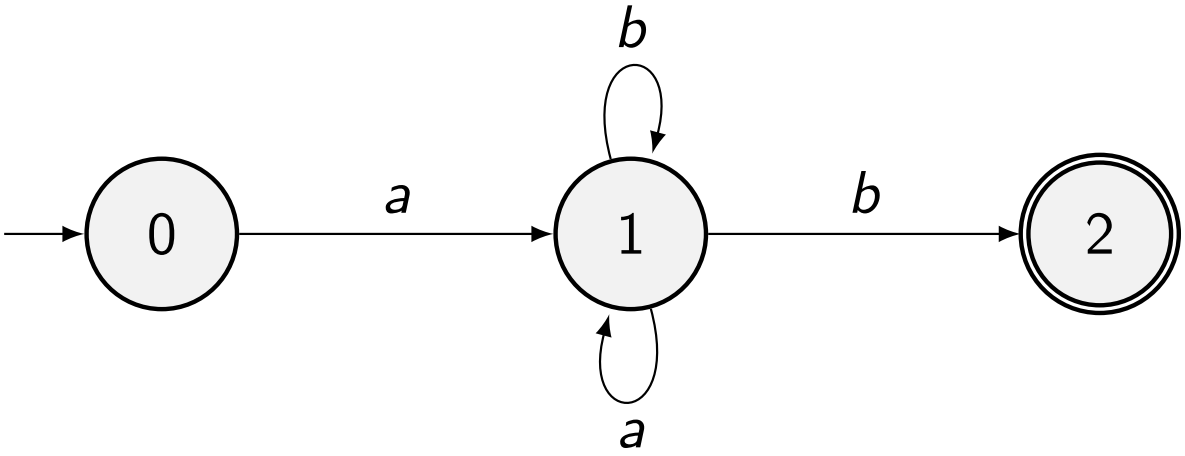
```
type 'a automate =
  { initiaux : int list;
    finaux : int list;
    delta : int * 'a -> int list };;
```

On pourrait utiliser n'importe quelle structure d'ensemble à la place de `list`, et un dictionnaire pour δ.

Implémentation

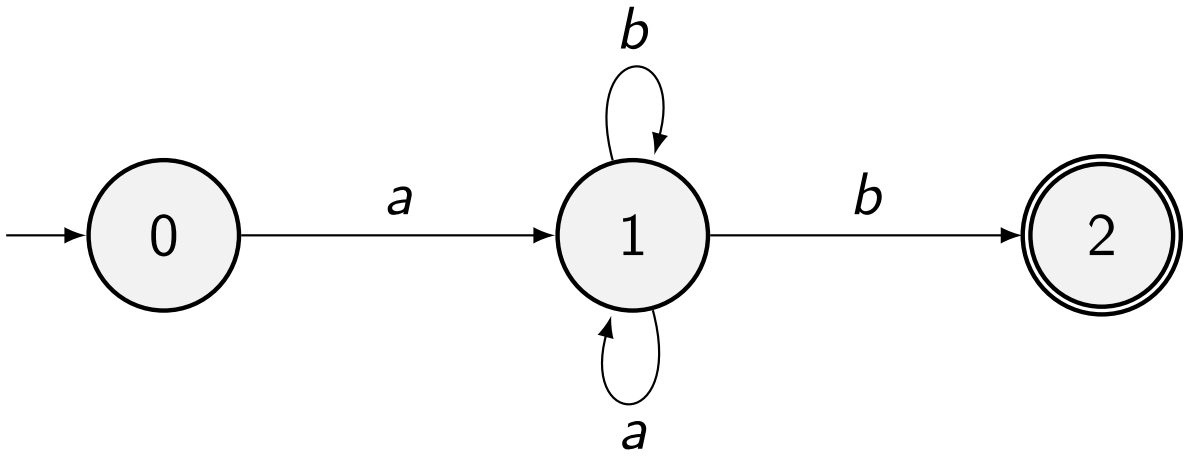


Implémentation




```
let ex_auto =
  { initiaux = [0];
    finaux = [2];
    delta = function
    | 0, "a" -> [1]
    | 0, "b" -> []
    | 1, "a" -> [1]
    | 1, "b" -> [1; 2]
    | 2, "a" -> []
    | 2, "b" -> [] };;
```

Implémentation



Plus simplement, en regroupant les transitions qui n'existent pas:

```
let ex_auto =
  { initiaux = [0];
    finaux = [2];
    delta = function
    | 0, "a" -> [1]
    | 1, "a" -> [1]
    | 1, "b" -> [1; 2]
    | _, _ -> [] };;
```

Acceptation

Question

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

Acceptation

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.

Acceptation

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.

Acceptation

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.

Acceptation

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.

Acceptation

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.

Pour renvoyer la liste des états accessibles depuis `etats` en lisant `lettre`:

```
let rec lire_lettre a lettre etats = match etats with
  | [] -> []
  | e::q -> union (a.delta (e, lettre)) (lire_lettre a lettre q);;
```

Pour renvoyer la liste des états accessibles depuis `etats` en lisant `lettre`:

```
let rec lire_lettre a lettre etats = match etats with
  | [] -> []
  | e::q -> union (a.delta (e, lettre)) (lire_lettre a lettre q);;
```

Pour renvoyer la liste des états accessibles depuis `etats` en lisant `mot`:

```
let rec lire_mot a mot etats = match mot with
  | [] ->  etats
  | m1::m -> lire_mot a m (lire_lettre a m1 etats);;
```

Pour renvoyer la liste des états accessibles depuis `etats` en lisant `lettre`:

```
let rec lire_lettre a lettre etats = match etats with
  | [] -> []
  | e::q -> union (a.delta (e, lettre)) (lire_lettre a lettre q);;
```

Pour renvoyer la liste des états accessibles depuis `etats` en lisant `mot`:

```
let rec lire_mot a mot etats = match mot with
  | [] ->  etats
  | m1::m -> lire_mot a m (lire_lettre a m1 etats);;
```

```
let rec intersect l1 l2 = match l1 with
  | [] -> false
  | e::q -> List.mem e l2 || intersect q l2;;
```

Pour renvoyer la liste des états accessibles depuis `etats` en lisant `lettre`:

```
let rec lire_lettre a lettre etats = match etats with
  | [] -> []
  | e::q -> union (a.delta (e, lettre)) (lire_lettre a lettre q);;
```

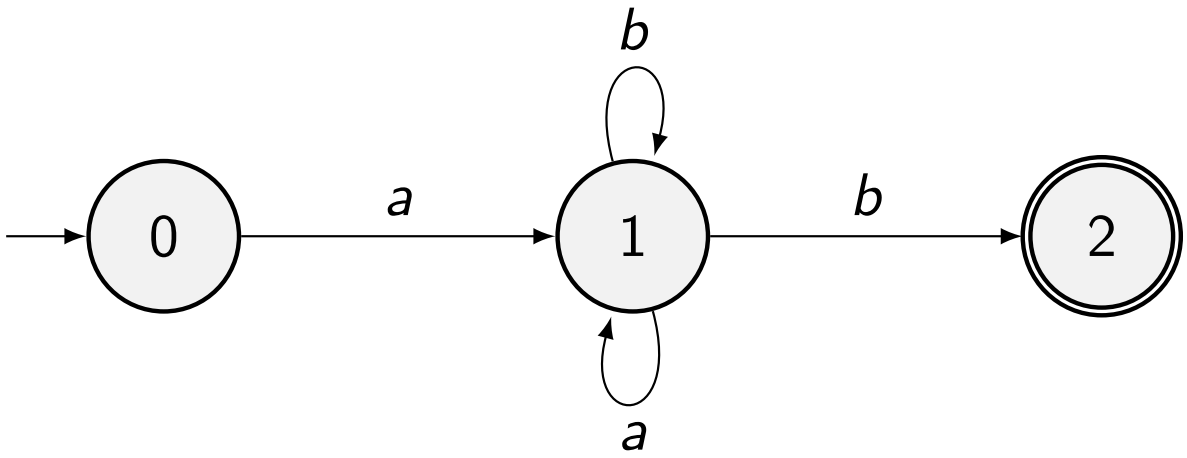
Pour renvoyer la liste des états accessibles depuis `etats` en lisant `mot`:

```
let rec lire_mot a mot etats = match mot with
  | [] ->  etats
  | m1::m -> lire_mot a m (lire_lettre a m1 etats);;
```

```
let rec intersect l1 l2 = match l1 with
  | [] -> false
  | e::q -> List.mem e l2 || intersect q l2;;
```

```
let accept a mot =
  intersect (lire_mot a mot a.initiaux) a.finaux;;
```

Acceptation: exemple




```
let accept a mot =
  intersect (lire_mot a mot a.initiaux) a.finaux;;
```

```
# accept ex_auto ["a"; "b"; "b"; "a"; "b"];;
- : bool = true
# accept ex_auto ["a"; "b"; "b"; "a"; "a"];;
- : bool = false
```

Simplification d'automate

Définition

Deux automates sont **équivalents** si ils ont le même langage.

Simplification d'automate

Définition

Deux automates sont **équivalents** si ils ont le même langage.

Étant donné un automate, il est intéressant de trouver un automate équivalent plus simple.

Automate complet

Définition

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

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

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 il existe une transition étiquetée par a (pas de blocage).

Automate complet

Théorème

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

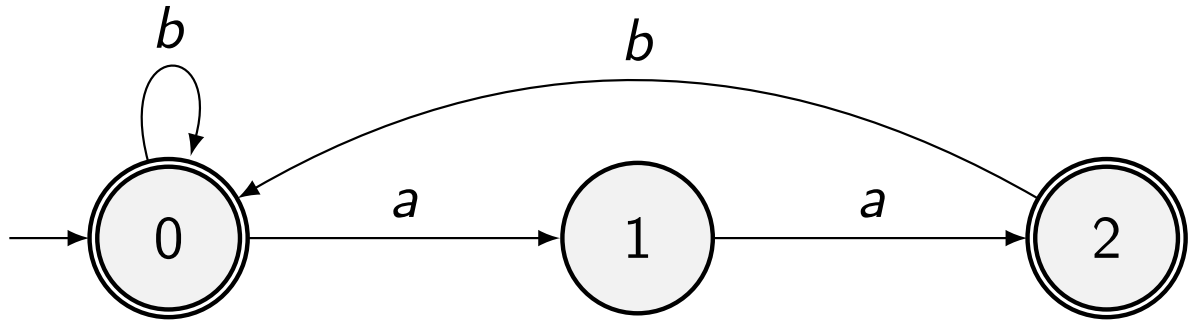
Automate complet

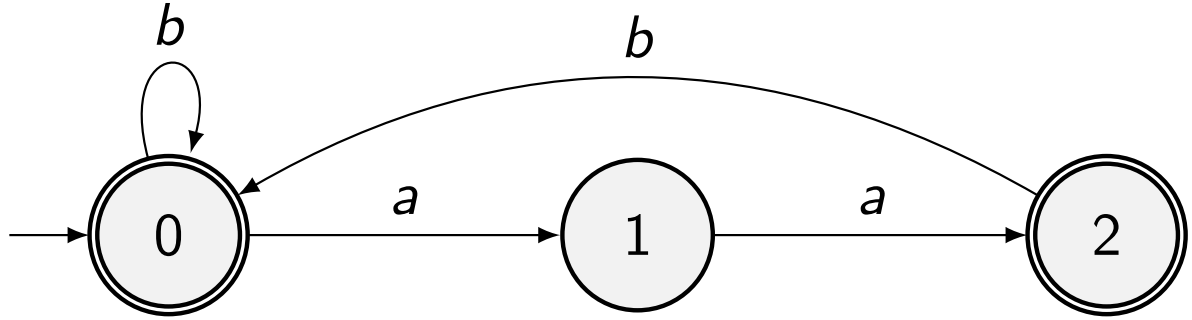
Théorème

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

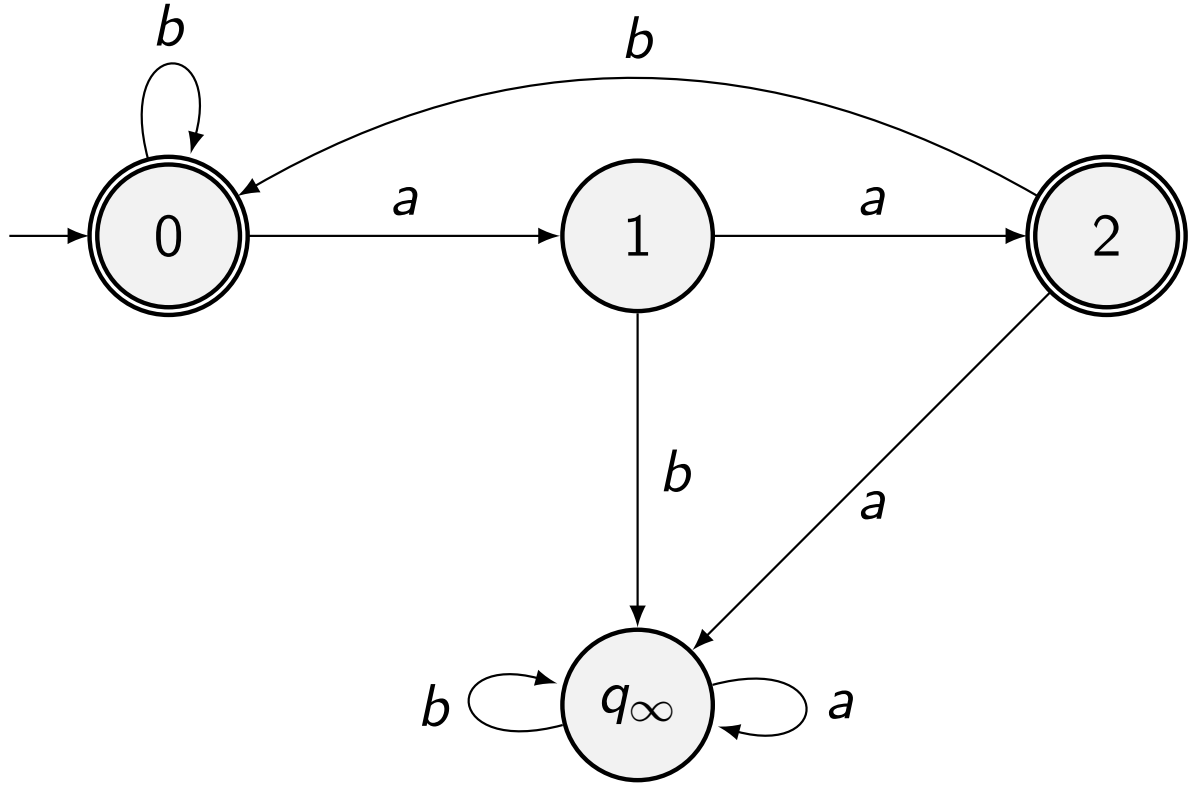
Preuve:

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 $q \xrightarrow{a} q_\infty$


b
b
a
a
0
1
2



Automate complet équivalent:



Automate déterministe

Définition

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

1. $|I| = 1$: il n'y a qu'un seul état initial
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

Automate déterministe

Définition

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

1. $|I| = 1$: il n'y a qu'un seul état initial
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.

Automate déterministe

Définition

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

1. $|I| = 1$: il n'y a qu'un seul état initial
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

Définition

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

1. $|I| = 1$: il n'y a qu'un seul état initial
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 $\delta : Q \times \Sigma \longrightarrow Q$.
On définit par récurrence δ^*:

$$\delta^*(q, \varepsilon) = q \text{ et } \delta^*(q, m_1 m) = \delta^*(\delta(q, m_1), m)$$

$\delta^*(q, m)$ est l'**état obtenu à partir de *q* en lisant le mot *m***.

Calculabilité

Les automates déterministes modélisent les machines à calculer / algorithmes « sans mémoire »: 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.

Calculabilité

Les automates déterministes modélisent les machines à calculer / algorithmes « sans mémoire »: 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.

Calculabilité

Les automates déterministes modélisent les machines à calculer / algorithmes « sans mémoire »: 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 complet

Un automate déterministe complet est particulièrement simple:

```
type 'a afdc =
  { initial : int;
    finaux : int list;
    delta : int * 'a -> int };;
```

Automate déterministe complet

Un automate déterministe complet est particulièrement simple:

```
type 'a afdc =
  { initial : int;
    finaux : int list;
    delta : int * 'a -> int };;
```

Pour renvoyer l'état accessible depuis `etat` en lisant `mot`:

```
let rec afdc_lire a mot etat = match mot with
  | [] -> etat
  | m1::m -> afdc_lire a m (a.delta (etat, m1));;
```

Automate déterministe complet

Un automate déterministe complet est particulièrement simple:

```
type 'a afdc =
  { initial : int;
    finaux : int list;
    delta : int * 'a -> int };;
```

Pour renvoyer l'état accessible depuis `etat` en lisant `mot`:

```
let rec afdc_lire a mot etat = match mot with
  | [] -> etat
  | m1::m -> afdc_lire a m (a.delta (etat, m1));;
```

Pour savoir si un mot est accepté:

```
let afdc_accept a mot =
  List.mem (afdc_lire a mot a.init) a.finaux;;
```

Automate déterministe complet

Un automate déterministe complet est particulièrement simple:

```
type 'a afdc =
  { initial : int;
    finaux : int list;
    delta : int * 'a -> int };;
```

Pour renvoyer l'état accessible depuis `etat` en lisant `mot`:

```
let rec afdc_lire a mot etat = match mot with
  | [] -> etat
  | m1::m -> afdc_lire a m (a.delta (etat, m1));;
```

Pour savoir si un mot est accepté:

```
let afdc_accept a mot =
  List.mem (afdc_lire a mot a.init) a.finaux;;
```

Complexité: $O(|\texttt{mot}|)$

Théorème

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

Théorème

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

Preuve:
Soit $A' = (\Sigma, \mathcal{P}(Q), \{I\}, F', \delta')$

Théorème

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

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

1. les états de A' sont les sous-ensembles de Q

Théorème

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

Preuve:

Soit $A' = (\Sigma, \mathcal{P}(Q), \{I\}, F', \delta')$ tel que:

1. les états de A' sont les sous-ensembles de Q
2. le seul état initial de A' est $I \subseteq Q$

Théorème

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

Preuve:

Soit $A' = (\Sigma, \mathcal{P}(Q), \{I\}, F', \delta')$ 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

Théorème

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

Preuve:

Soit $A' = (\Sigma, \mathcal{P}(Q), \{I\}, F', \delta')$ 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. $\forall X \subseteq Q, \forall a \in \Sigma, \delta'(X, a) = \bigcup_{q \in X} \delta(q, a)$

Théorème

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

Preuve:

Soit $A' = (\Sigma, \mathcal{P}(Q), \{I\}, F', \delta')$ 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.

Théorème

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

Preuve:

Soit $A' = (\Sigma, \mathcal{P}(Q), \{I\}, F', \delta')$ 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.

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

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

Si $|m| = 0$: la propriété est vraie (les conditions sont équivalentes à $q \in I$)

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

Si $|m| = 0$: la propriété est vraie (les conditions sont équivalentes à $q \in I$)
Supposons la propriété vraie pour des mots de longueur $n-1$.
Soit $m = m_1 ... m_n$ un mot de longueur n.

$\Longrightarrow$ S'il y a un chemin $q_1 \in I \xrightarrow{m_1} q_2 \xrightarrow{m_2} ... \xrightarrow{m_{n-1}} q_n \xrightarrow{m_n} q \in F$ dans A:

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

Si $|m| = 0$: la propriété est vraie (les conditions sont équivalentes à $q \in I$)
Supposons la propriété vraie pour des mots de longueur $n-1$.
Soit $m = m_1 ... m_n$ un mot de longueur n.

$\Longrightarrow$ S'il y a un chemin $q_1 \in I \xrightarrow{m_1} q_2 \xrightarrow{m_2} ... \xrightarrow{m_{n-1}} q_n \xrightarrow{m_n} q \in F$ dans A:
Alors il y a un chemin de q_1 vers q_n d'étiquette $m_1 ... m_{n-1}$

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

Si $|m| = 0$: la propriété est vraie (les conditions sont équivalentes à $q \in I$)
Supposons la propriété vraie pour des mots de longueur $n-1$.
Soit $m = m_1 ... m_n$ un mot de longueur n.

$\boxed{\Longrightarrow}$ S'il y a un chemin $q_1 \in I \xrightarrow{m_1} q_2 \xrightarrow{m_2} ... \xrightarrow{m_{n-1}} q_n \xrightarrow{m_n} q \in F$ 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.

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

Si $|m| = 0$: la propriété est vraie (les conditions sont équivalentes à $q \in I$)
Supposons la propriété vraie pour des mots de longueur $n-1$.
Soit $m = m_1 ... m_n$ un mot de longueur n.

$\Longrightarrow$ S'il y a un chemin $q_1 \in I \xrightarrow{m_1} q_2 \xrightarrow{m_2} ... \xrightarrow{m_{n-1}} q_n \xrightarrow{m_n} q \in F$ 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.

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

Si $|m| = 0$: la propriété est vraie (les conditions sont équivalentes à $q \in I$)
Supposons la propriété vraie pour des mots de longueur $n-1$.
Soit $m = m_1 ... m_n$ un mot de longueur n.

$\boxed{\Longrightarrow}$ S'il y a un chemin $q_1 \in I \xrightarrow{m_1} q_2 \xrightarrow{m_2} ... \xrightarrow{m_{n-1}} q_n \xrightarrow{m_n} q \in F$ 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.

$\boxed{\Longleftarrow}$ Réciproque similaire.

On a donc montré que $L(A) = L(A')$.

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

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

Algorithme de déterminisation

Initialement: `next` contient seulement I

Tant que `next` $\neq \emptyset$:

  Extraire un élément X de `next`

  Pour toute lettre a de Σ:

    Soit $X' = \bigcup_{q \in X} \delta(q, a)$

    Ajouter une transition $X \xrightarrow{a} X'$

    Si X' n'a pas déjà été visité:

      Ajouter X' à `next`

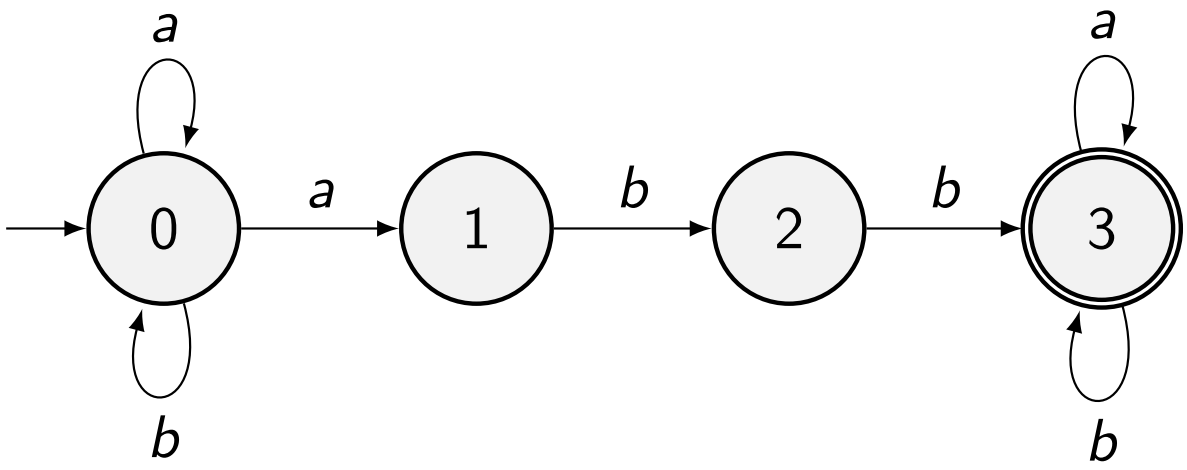
On peut construire le déterminisé de proche en proche, par parcours de l'automate à construire:

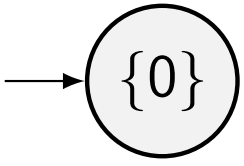
Algorithme de déterminisation

Initialement: `next` contient seulement I
Tant que `next` $\neq \emptyset$:
    Extraire un élément X de `next`
    Pour toute lettre a de Σ:
        Soit $X' = \bigcup_{q \in X} \delta(q, a)$
        Ajouter une transition $X \xrightarrow{a} X'$
        Si X' n'a pas déjà été visité:
            Ajouter X' à `next`

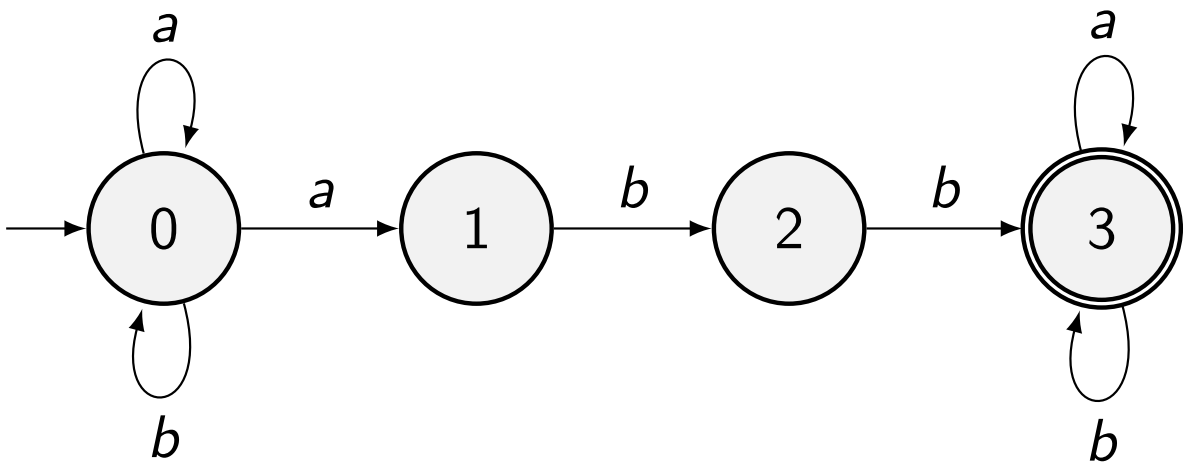
On ne construit que les états atteignables depuis I.

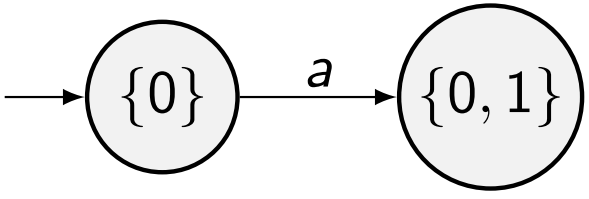
Déterminisation: exemple



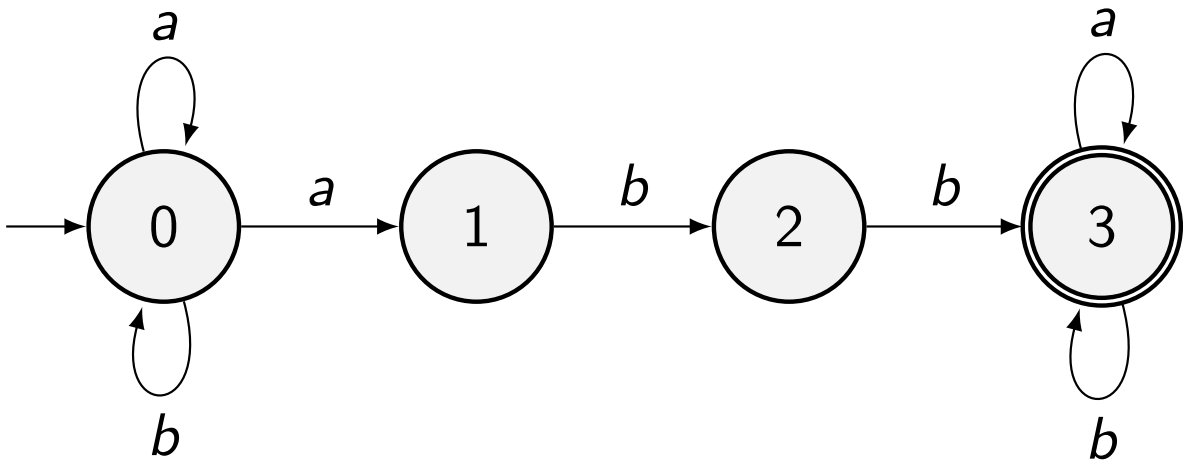


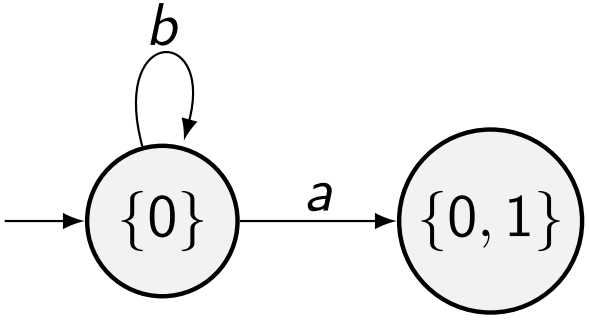
Déterminisation: exemple



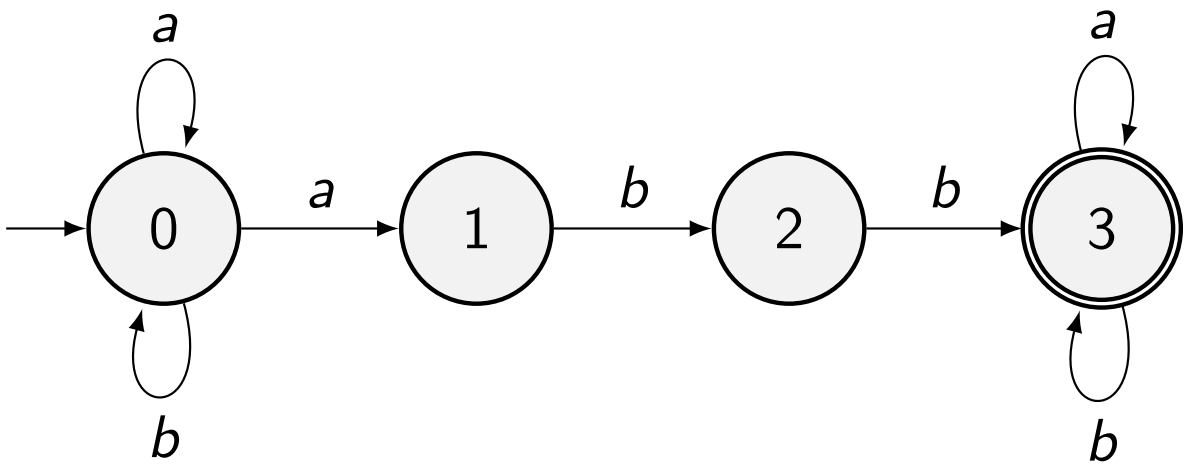


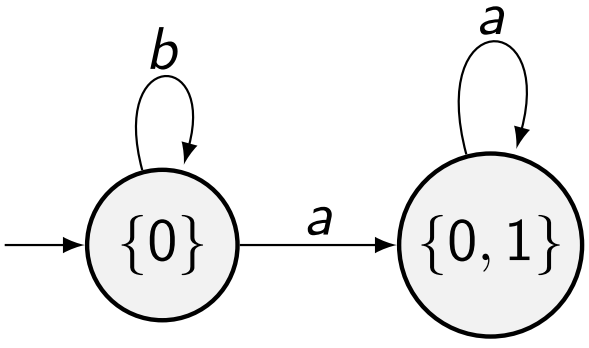
Déterminisation: exemple



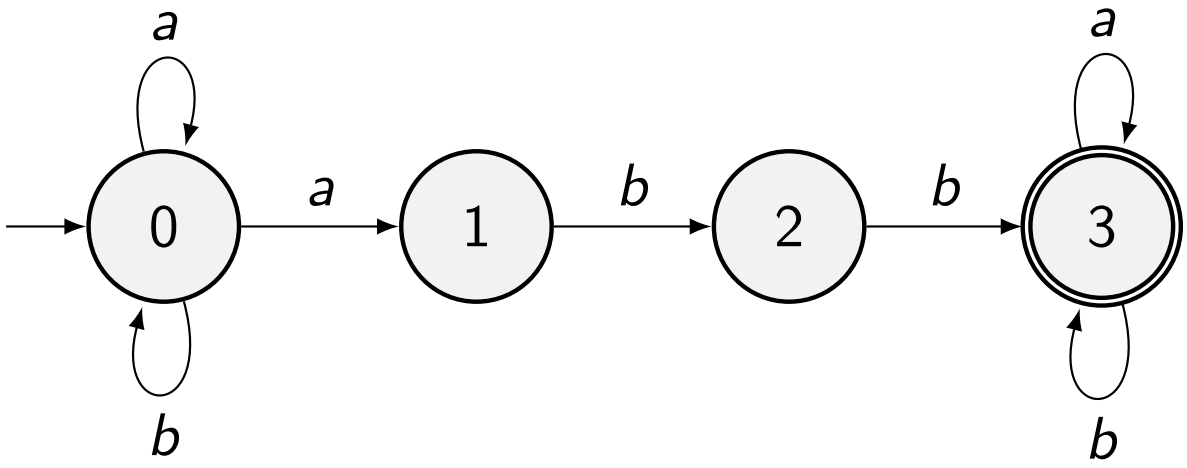


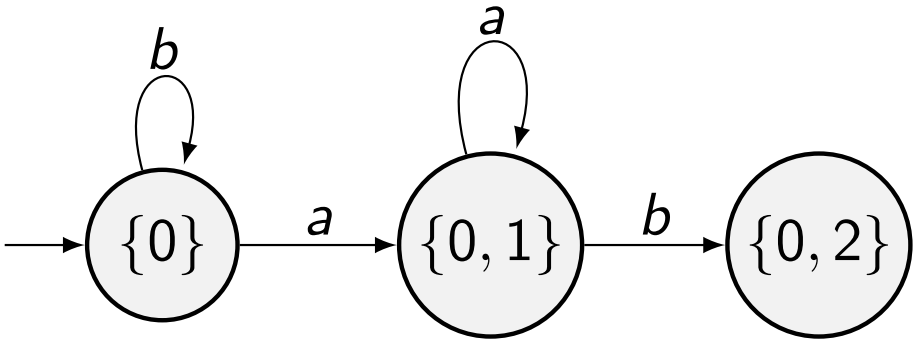
Déterminisation: exemple



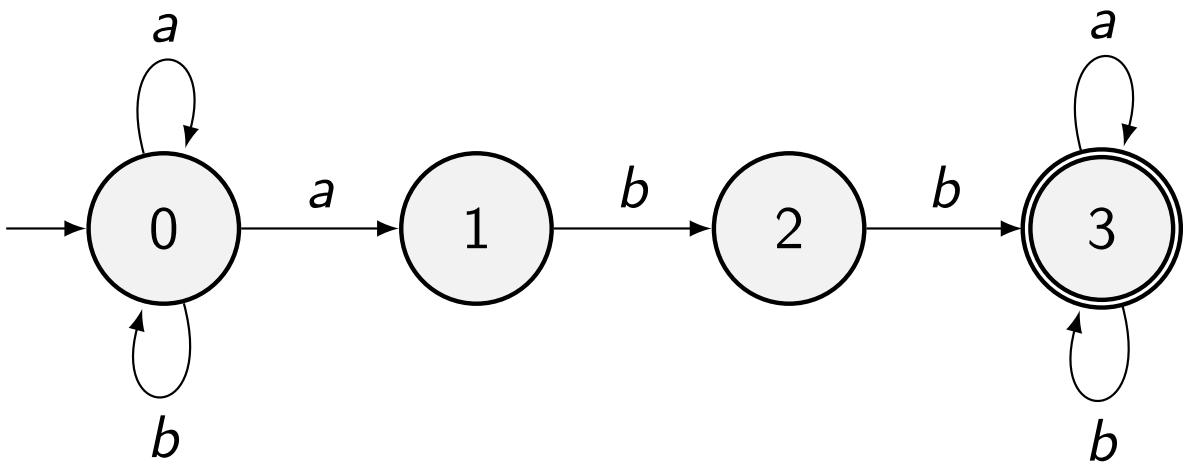


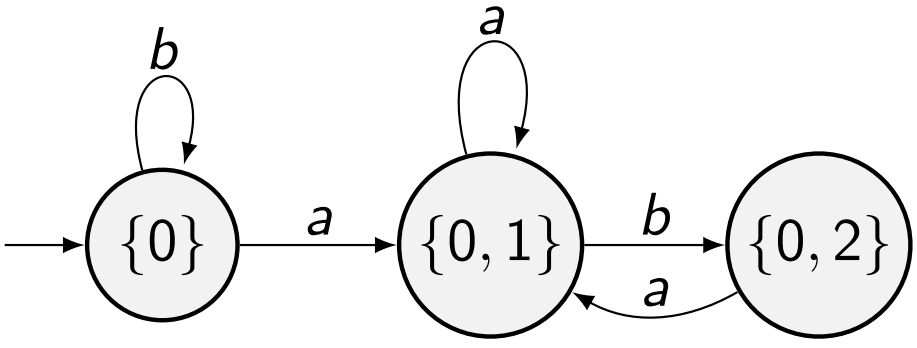
Déterminisation: exemple



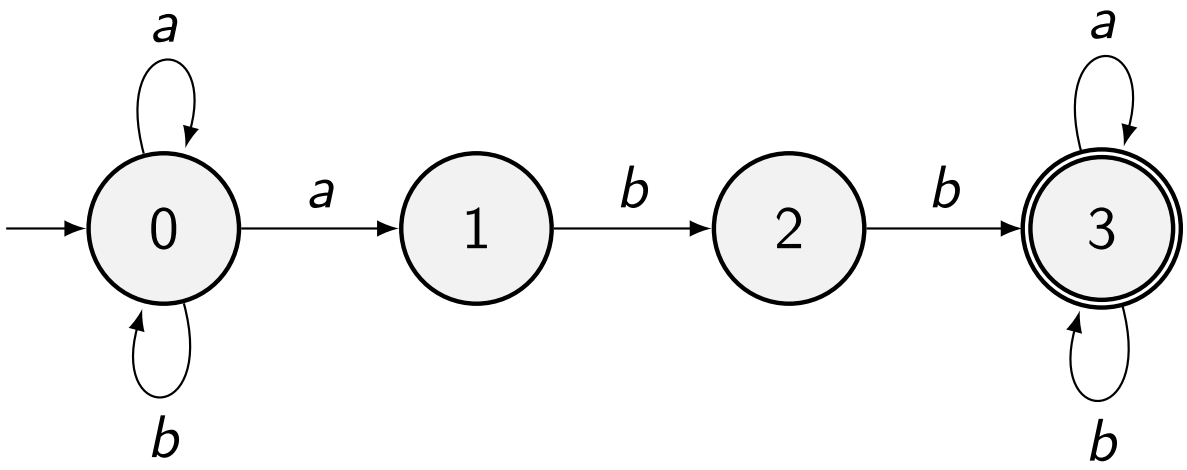


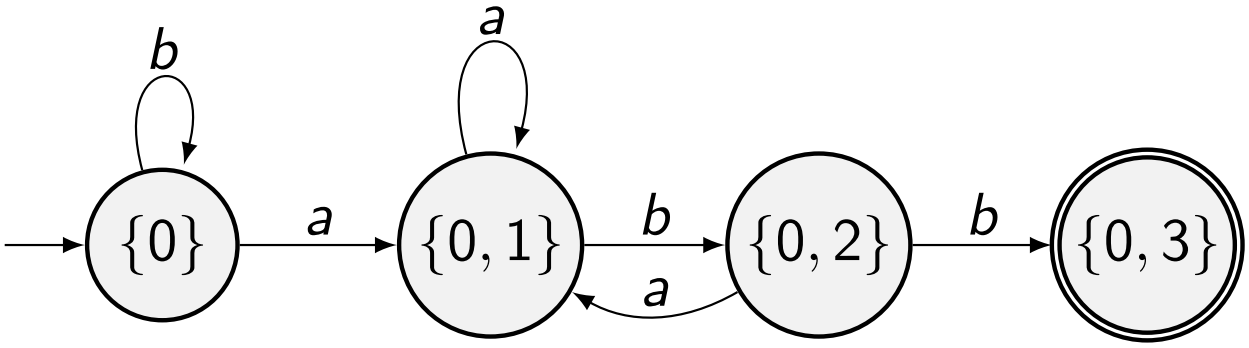
Déterminisation: exemple



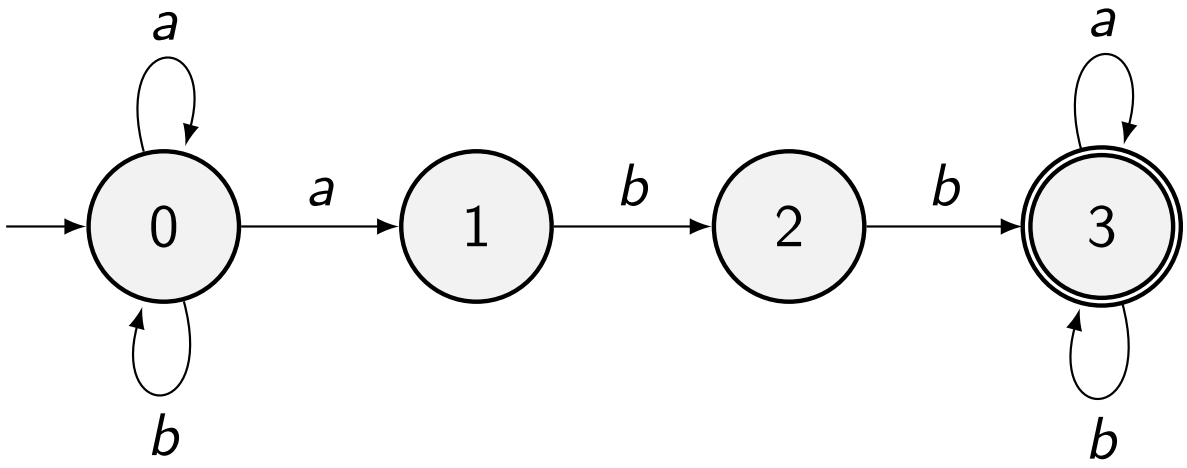


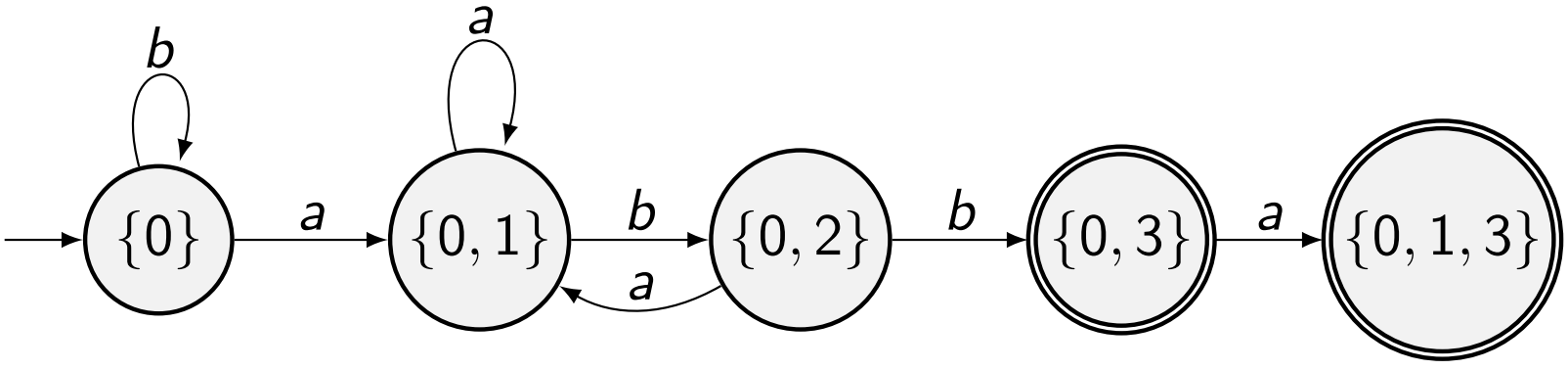
Déterminisation: exemple



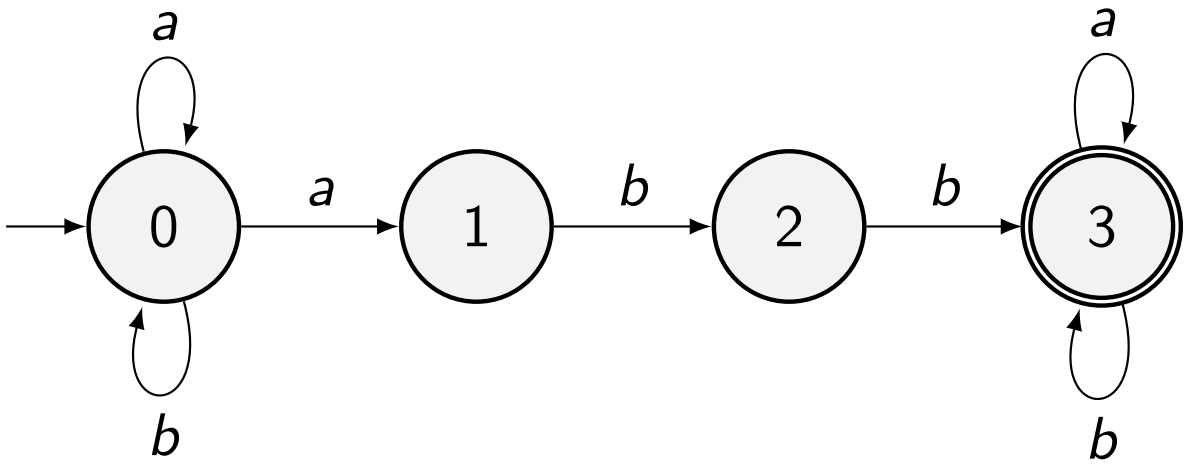


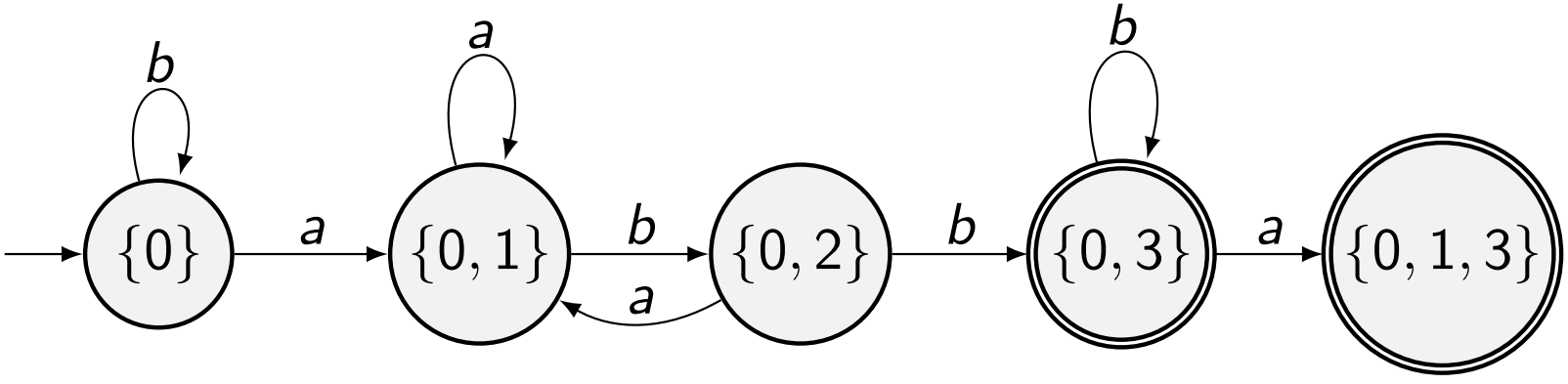
Déterminisation: exemple



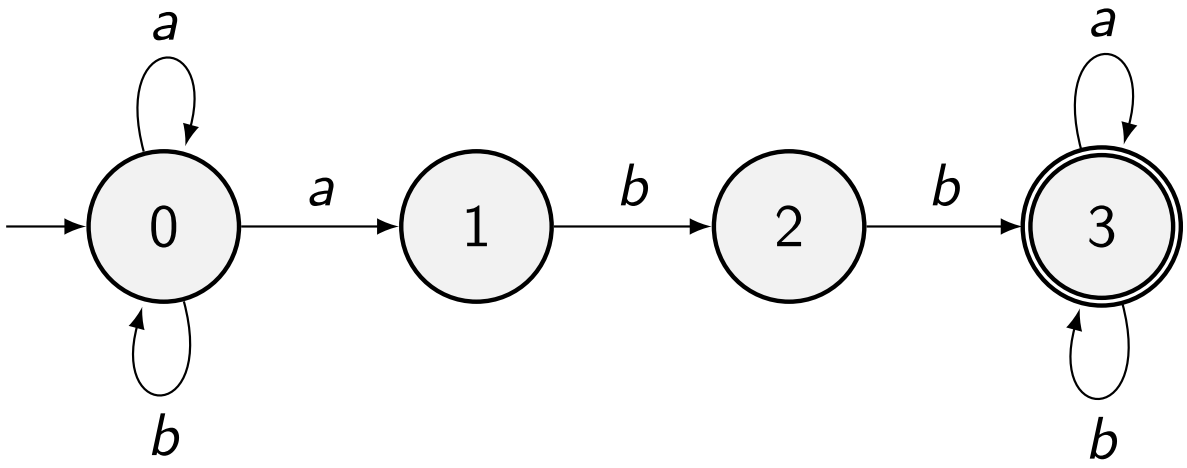


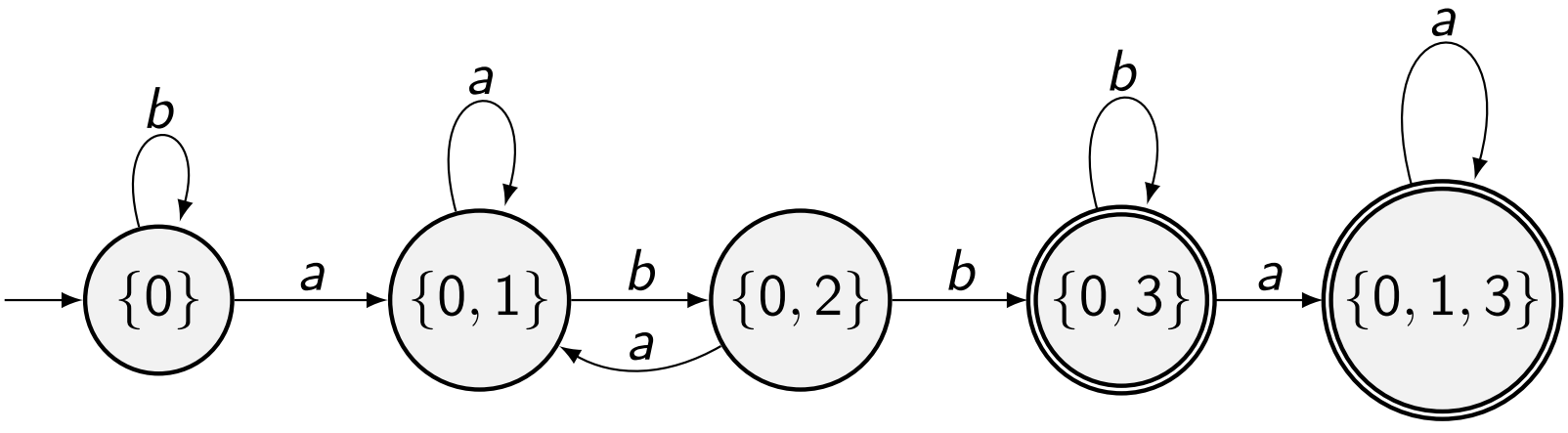
Déterminisation: exemple



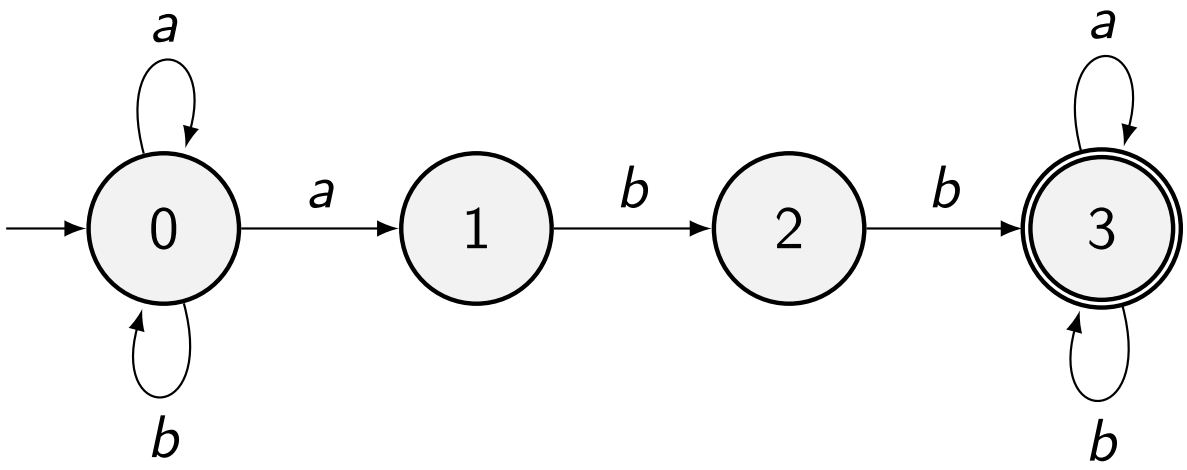


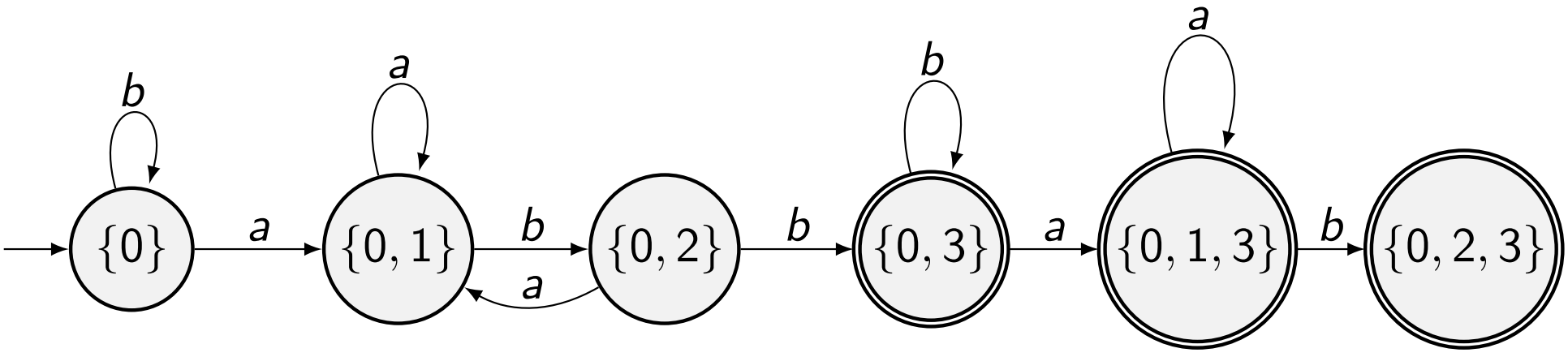
Déterminisation: exemple



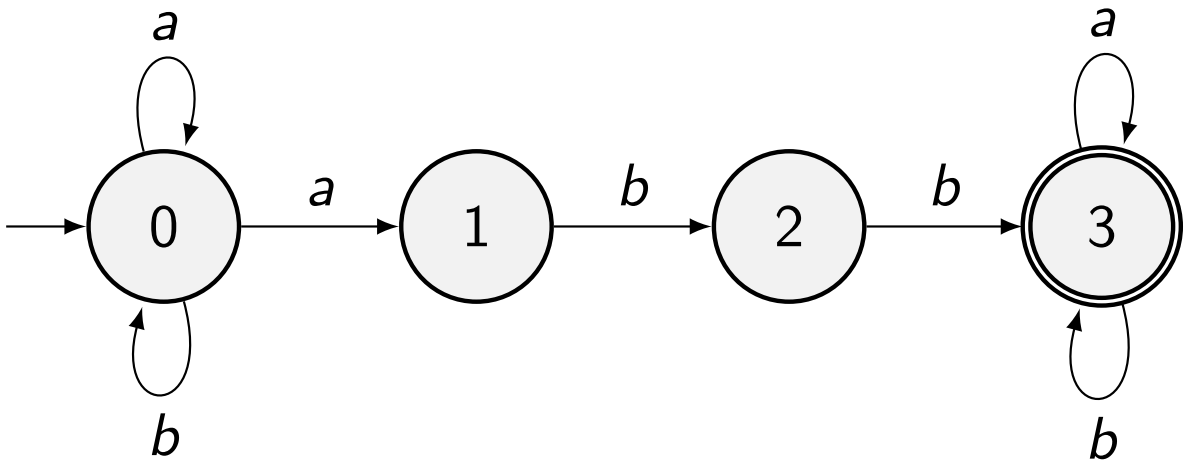


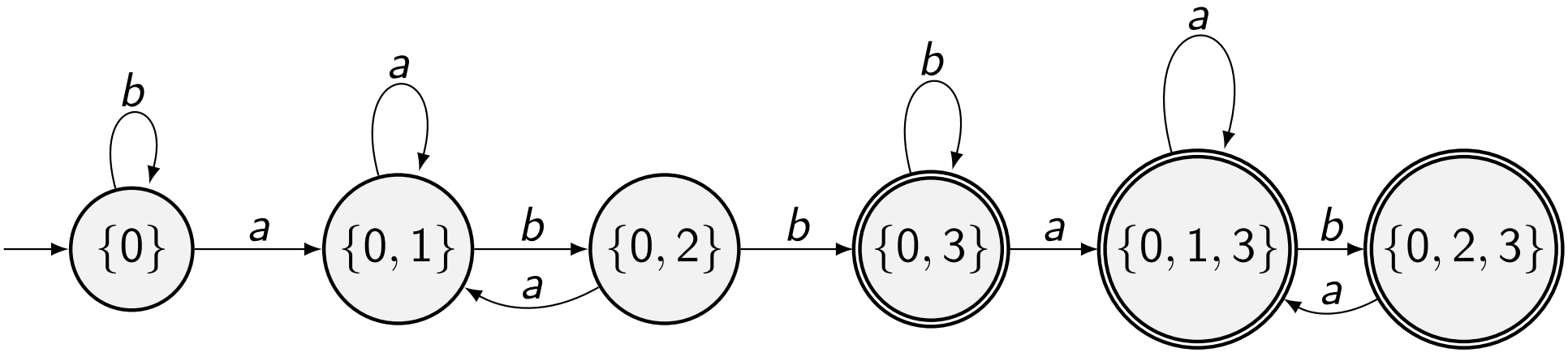
Déterminisation: exemple



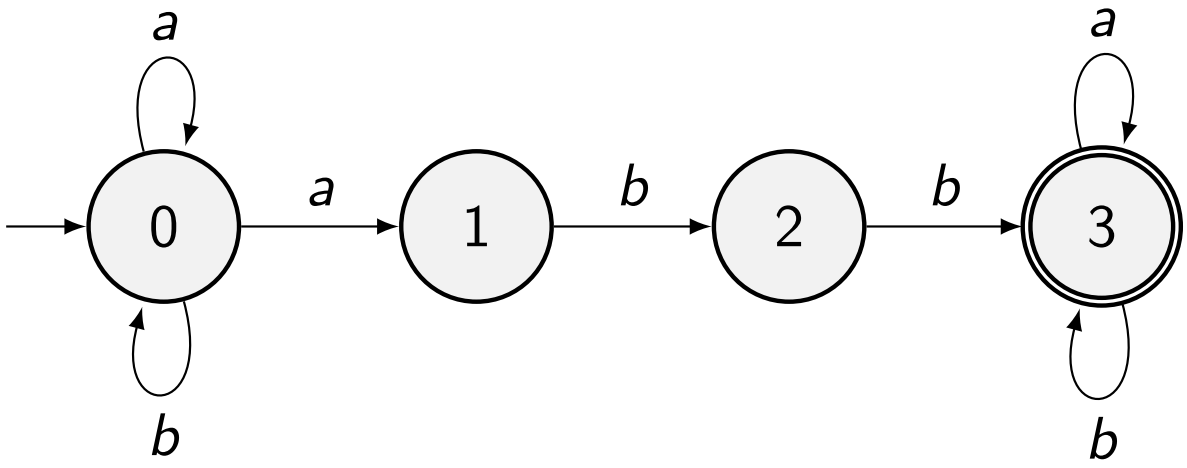


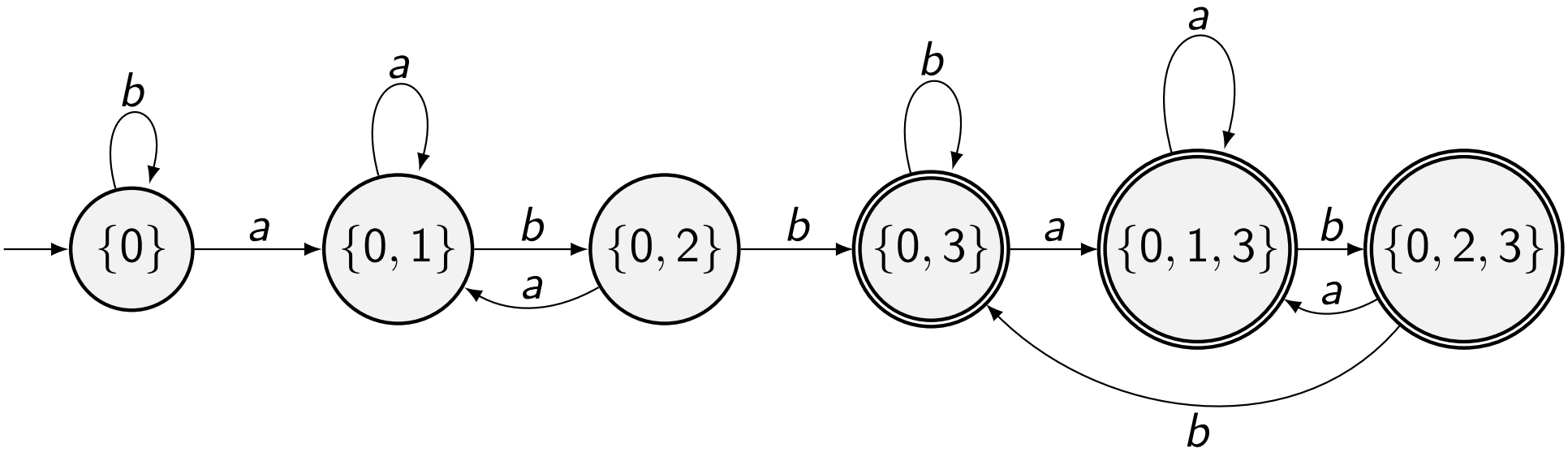
Déterminisation: exemple





Déterminisation: exemple





Complexité de la déterminisation

Théorème

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

Complexité de la déterminisation

Théorème

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

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

Complexité de la déterminisation

Théorème

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

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

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

Complexité de la déterminisation

Théorème

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

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 $O(|m|)$.

Complexité de la déterminisation

On peut se demander s'il y a une méthode pour trouver un automate déterministe avec moins d'états (non exponentiel).

Complexité de la déterminisation

On peut se demander s'il y a une méthode pour trouver un automate déterministe avec moins d'états (non exponentiel).

Soit $\Sigma = \{a, b\}$. Alors:

1. $\Sigma^* a \Sigma^{n-1}$ est reconnu par un automate non déterministe à $n+1$ états
2. Tout automate déterministe reconnaissant $\Sigma^* a \Sigma^{n-1}$ possède au moins 2^n états.

Preuve:

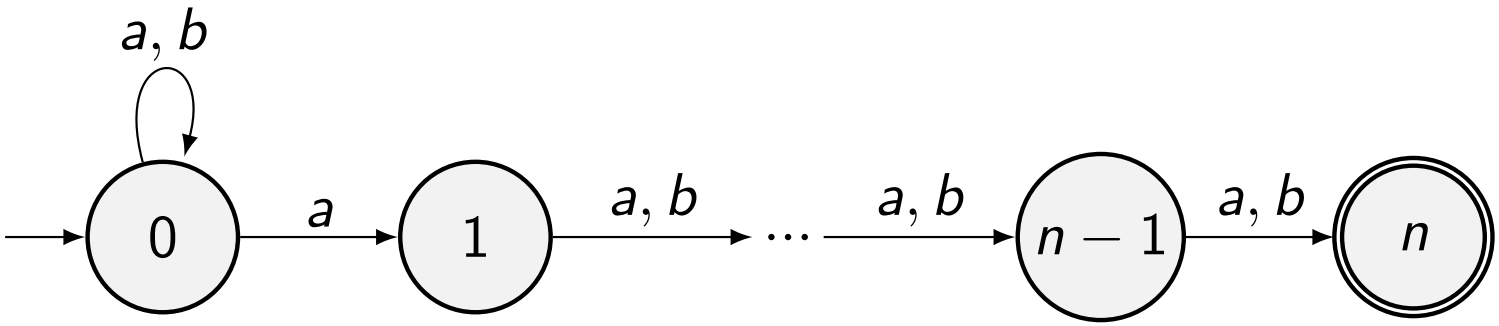
Complexité de la déterminisation

On peut se demander s'il y a une méthode pour trouver un automate déterministe avec moins d'états (non exponentiel).

Soit $\Sigma = \{a, b\}$. Alors:

1. $\Sigma^* a \Sigma^{n-1}$ est reconnu par un automate non déterministe à $n+1$ états
2. Tout automate déterministe reconnaissant $\Sigma^* a \Sigma^{n-1}$ possède au moins 2^n états.

Preuve:



Complexité de la déterminisation

On peut se demander s'il y a une méthode pour trouver un automate déterministe avec moins d'états (non exponentiel).

Soit $\Sigma = \{a, b\}$. Alors:

1. $\Sigma^* a \Sigma^{n-1}$ est reconnu par un automate non déterministe à $n+1$ états
2. Tout automate déterministe reconnaissant $\Sigma^* a \Sigma^{n-1}$ possède au moins 2^n états.

Preuve:
Soit $A = (\Sigma, Q, q_i, F, \delta)$ un automate déterministe reconnaissant $\Sigma^* a \Sigma^{n-1}$.
La fonction f suivante est injective, donc $|Q| \geq |\Sigma^n| = 2^n$:

$$\begin{array}{rccc} f & : & \Sigma^n & \longrightarrow & Q \\ & & m & \longmapsto & \delta^*(q_i, m) \end{array}$$

Complémentaire

Théorème

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

Complémentaire

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.

Complémentaire

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.
Alors $A' = (\Sigma, Q, q_i, Q \backslash F, \delta)$ a pour langage $\overline{L}$ (on inverse états finaux et non-finaux).

Automate produit

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.

Automate produit

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 AFDC reconnaissant 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 $\delta((q_1, q_2), a) = (\delta_1(q_1, a), \delta_2(q_2, a))$

Automate produit

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 AFDC reconnaissant 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 $L_1 \cap L_2$.

Automate produit

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 AFDC reconnaissant 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 $L_1 \cup L_2$.

Automate produit

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 AFDC reconnaissant 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 $L_1 \backslash L_2$.

Remarque

Si on a déjà démontré que les langages reconnaissables (resp. rationnels) sont stables par union et passage au complémentaire, on peut en déduire la stabilité par intersection en remarquant que:

$$L_1 \cap L_2 = \overline{\overline{L_1} \cup \overline{L_2}}$$

Remarque

Si on a déjà démontré que les langages reconnaissables (resp. rationnels) sont stables par union et passage au complémentaire, on peut en déduire la stabilité par intersection en remarquant que:

$$L_1 \cap L_2 = \overline{\overline{L_1} \cup \overline{L_2}}$$

(loi de De Morgan pour les ensembles)