Automates

Quentin Fortier

September 18, 2024

Automate (non déterministe)

Définition

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

Automate (non déterministe)

Définition

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

- Σ est un alphabet

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**

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**
- $I \subset Q$ est un ensemble d'**états initiaux**

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**
- $I \subset Q$ est un ensemble d'**états initiaux**
- $F \subset Q$ est un ensemble d'**états acceptants** (ou **états finaux**)

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

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**
- $I \subset Q$ est un ensemble d'**états initiaux**
- $F \subset Q$ est un ensemble d'**états acceptants** (ou **états finaux**)
- $E \subset Q \times \Sigma \times Q$ 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 :

$$q_1 \xrightarrow{a} q_2$$

Automate (non déterministe)

Soit $A = (\Sigma, Q, I, F, E)$ où :

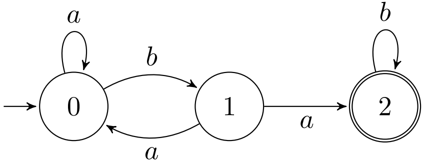
- $\Sigma = \{a, b\}$
- $Q = \{0, 1, 2\}$
- $I = \{0\}$
- $F = \{2\}$
- $E = \{(0, a, 0), (0, b, 1), (1, a, 0), (1, a, 2), (2, b, 2)\}$

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 :

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

Automate (non déterministe)

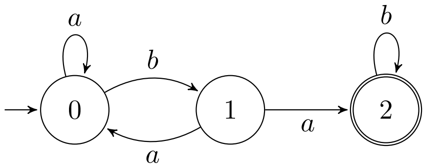
Définition

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 de l'automate suivant.



Implémentation

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

On peut implémenter δ avec un dictionnaire où chaque clé est un couple (q, a) auquel est associé la valeur $\delta(q, a)$.
OCaml possède deux implémentations de dictionnaires :

- `Map` : par arbre binaire de recherche où chaque nœud est un couple $(\text{cle}, \text{valeur})$. Les clés doivent être ordonnées.
- `Hashtbl` : par table de hachage. Les clés doivent être hashables et donc immuables (non modifiables).

Implémentation : Tables de hachage

Définition

Une **table de hachage** est composée de :

- Un **tableau** contenant les valeurs.
- Une **fonction de hachage** h telle que, si k est une clé, $h(k)$ est l'indice du tableau où se trouve la valeur associée à k.

Implémentation : Tables de hachage

Définition

Une **table de hachage** est composée de :

- Un **tableau** contenant les valeurs.
- Une **fonction de hachage** h telle que, si k est une clé, $h(k)$ est l'indice du tableau où se trouve la valeur associée à k.

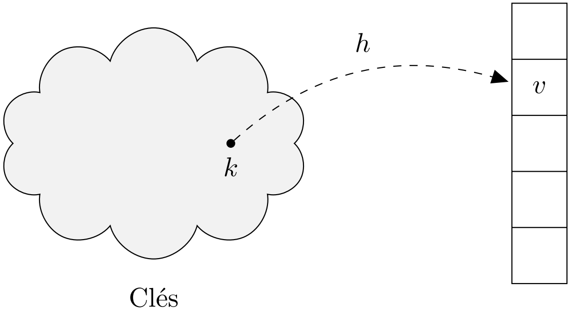
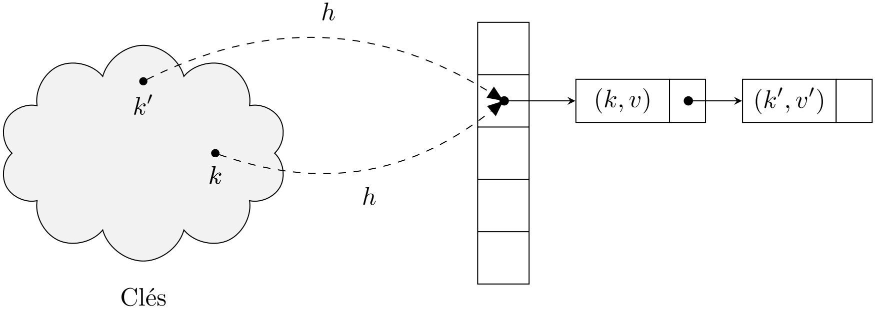


Table de hachage $\approx$ tableau dont les indices (clés) ne sont pas forcément des entiers consécutifs.

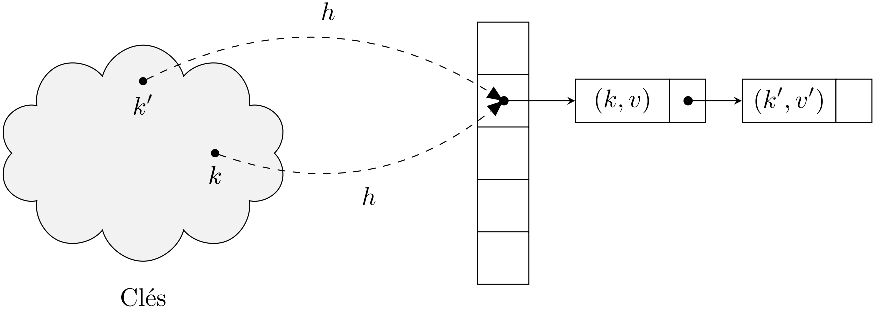
Implémentation : Tables de hachage

Souvent, il y a beaucoup plus de clés possibles que de cases du tableau, ce qui conduit à des **collisions** : plusieurs clés ayant la même image par h. On peut résoudre ces collisions par **chaînage**, en stockant une liste à chaque position de la table de hachage :



Implémentation : Tables de hachage

Souvent, il y a beaucoup plus de clés possibles que de cases du tableau, ce qui conduit à des **collisions** : plusieurs clés ayant la même image par h. On peut résoudre ces collisions par **chaînage**, en stockant une liste à chaque position de la table de hachage :



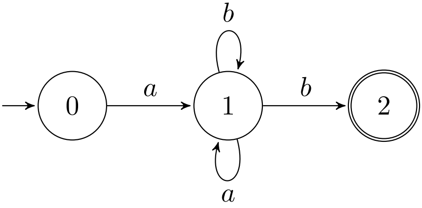
Autre possibilité de résolution de collisions : **adressage ouvert** où l'on ajoute une valeur à la première case libre après l'indice donné par h.

Implémentation : Tables de hachage

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.
On peut alors représenter un automate par le type suivant :

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

Implémentation : Tables de hachage




```
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 $q_n \in F$.

Langage reconnaissable

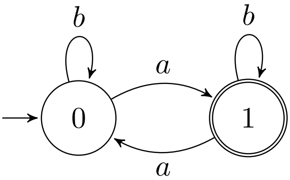
Définition

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

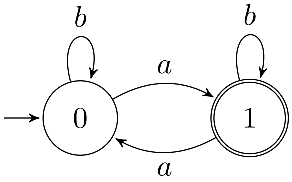
Soit A un automate.

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

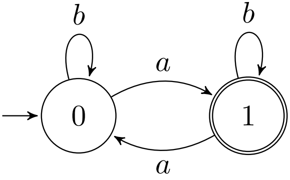
Quel est le langage reconnu par l'automate A ci-dessous ?



Langage reconnaissable

Exercice

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

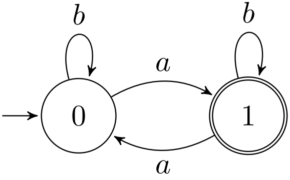


$L(A) =$ mots avec un nombre impair de a $=$

Langage reconnaissable

Exercice

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



$L(A) =$ mots avec un nombre impair de $a = b^*ab^*(b^*ab^*ab^*)^*$.

Langage reconnaissable

Définition

Un langage est **reconnaissable** s'il est reconnu par un automate.

Langage reconnaissable

Définition

Un langage est **reconnaissable** s'il est reconnu par un automate.

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

Langage reconnaissable

Définition

Un langage est **reconnaissable** s'il est reconnu par un automate.

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

Question

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

Langage reconnaissable

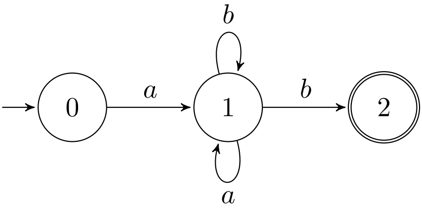
Définition

Un langage est **reconnaissable** s'il est reconnu par un automate.

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

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 $(b \mid ab \mid aba)^*$ est reconnnaissable.

Test d'appartenance au langage d'un automate

Question

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

Test d'appartenance au langage d'un automate

Question

Comment déterminer informatiquement si un automate non-déterministe 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.
  m est accepté par A si et seulement si Q_n contient un état final (c'est-à-dire $Q_n \cap F \neq \emptyset$).

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`.

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
```

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

Test d'appartenance au langage d'un automate

Question

Écrire une fonction `accepte (a : automate) (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 :

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

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

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).
Attention : ne pas confondre avec un graphe complet !

Automate complet

Théorème

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

Automate complet

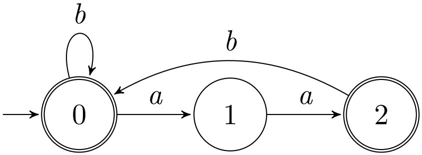
Théorème

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 $q \xrightarrow{a} q_\infty$

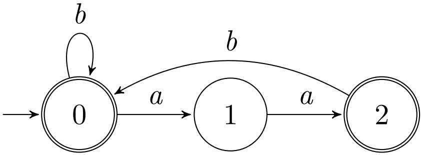
Automate complet



Question

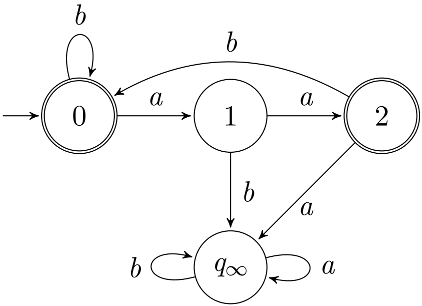
Donner un automate complet équivalent à l'automate ci-dessus ($\Sigma = \{a, b\}$).

Automate complet



Question

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



Automate déterministe

Définition

Un automate $(\Sigma, Q, \{q_i\}, F, E)$ 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

Automate déterministe

Définition

Un automate $(\Sigma, Q, \{q_i\}, F, E)$ 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.

Automate déterministe

Définition

Un automate $(\Sigma, Q, \{q_i\}, F, E)$ 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

Définition

Un automate $(\Sigma, Q, \{q_i\}, F, E)$ 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 $\delta : Q \times \Sigma \longrightarrow Q$.

Question

Quel est le nombre de transitions d'un automate déterministe complet ?

Automate déterministe

Définition

Un automate $(\Sigma, Q, \{q_i\}, F, E)$ 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 $\delta : Q \times \Sigma \longrightarrow Q$.

Question

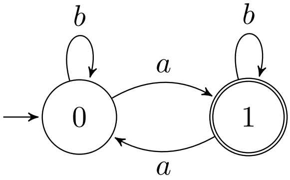
Quel est le nombre de transitions d'un automate déterministe complet ?

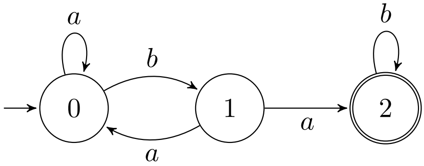
Réponse : $\mathrm{O}(|Q| \times |\Sigma|)$.

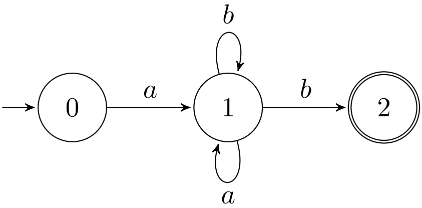
Automate déterministe

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

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.

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

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 :

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

<u>Attention</u> : δ^* n'est pas défini pour un automate non déterministe.

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.

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.

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.

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.

Automate déterministe

Question

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

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

Complexité :

Automate déterministe

Question

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

```
let accepte a m =
  let etat = ref a.initial in
  for i = 0 to String.length m - 1 do
    etat := Hashtbl.find a.delta (!etat, m.[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.

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)**

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.

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 $I \subset Q$.

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 $I \subset Q$.
3. $F' = \{X \subset Q \mid X \cap F \neq \emptyset\}$: les états finaux de A' sont ceux contenant au moins un état final de A.

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 $I \subset Q$.
3. $F' = \{X \subset Q \mid X \cap F \neq \emptyset\}$: les états finaux de A' sont ceux contenant au moins un état final de A.
4. $\forall X \subset Q$, $\forall a \in \Sigma$, $\delta'(X, a) = \bigcup_{q \in X} \delta(q, a)$

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 $I \subset Q$.
3. $F' = \{X \subset Q \mid X \cap F \neq \emptyset\}$: les états finaux de A' sont ceux contenant au moins un état final de A.
4. $\forall X \subset Q$, $\forall a \in \Sigma$, $\delta'(X, a) = \bigcup_{q \in X} \delta(q, a)$: 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.

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 $I \subset Q$.
3. $F' = \{X \subset Q \mid X \cap F \neq \emptyset\}$: les états finaux de A' sont ceux contenant au moins un état final de A.
4. $\forall X \subset Q$, $\forall a \in \Sigma$, $\delta'(X, a) = \bigcup_{q \in X} \delta(q, a)$: 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

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

$$\iff$$

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

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

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.

$\boxed{\Longrightarrow}$ 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.

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.

$\boxed{\Longrightarrow}$ 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 $m_1...m_{n-1}$

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.

$\boxed{\Longrightarrow}$ 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.

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.

$\boxed{\Longrightarrow}$ 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

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

$$\iff$$

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

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

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.

$\boxed{\Longleftarrow}$: 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 :

Automate déterministe

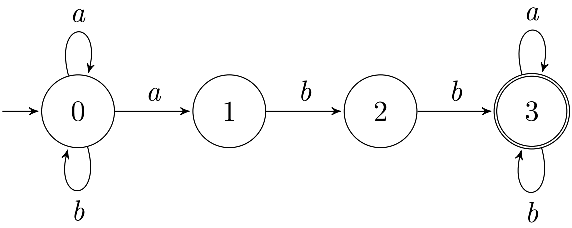
On peut construire le déterminisé de proche en proche, par parcours de l'automate des parties :

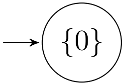
Algorithme de déterminisation

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

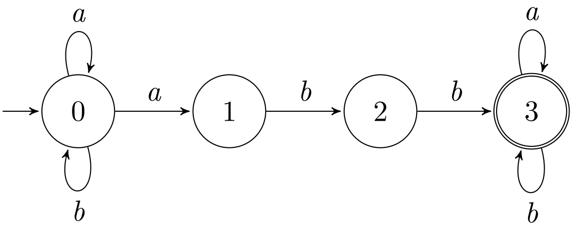
next $\leftarrow \{I\}$
Tant que next $\neq \emptyset$ **:**
  Extraire un élément X de `next`
  Pour $a \in \Sigma$ **:**
    $X' \leftarrow$ ensemble des états accessibles depuis un état de X en lisant a
    Si X' n'a pas déjà été visité **:**
      Ajouter une transition $X \xrightarrow{a} X'$ à A'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

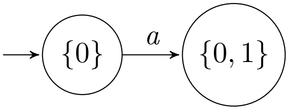
Automate déterministe



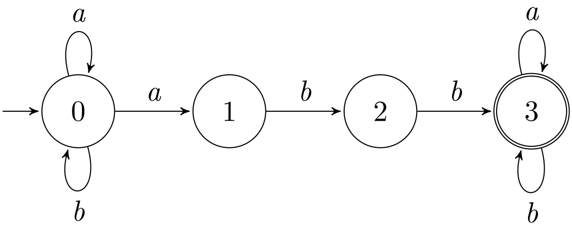


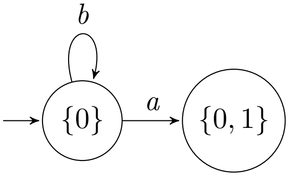
Automate déterministe



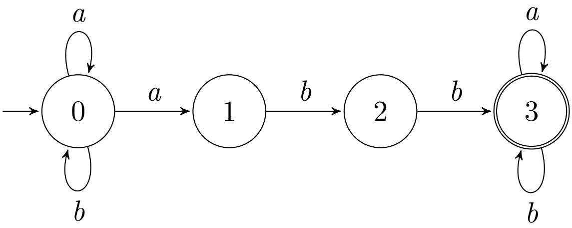


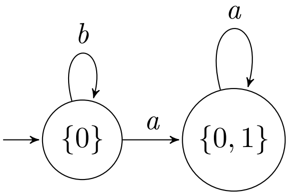
Automate déterministe



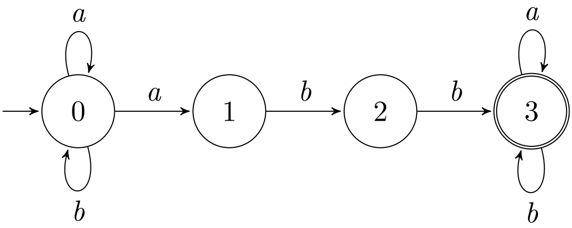


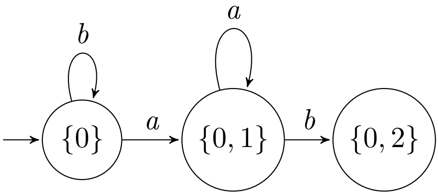
Automate déterministe



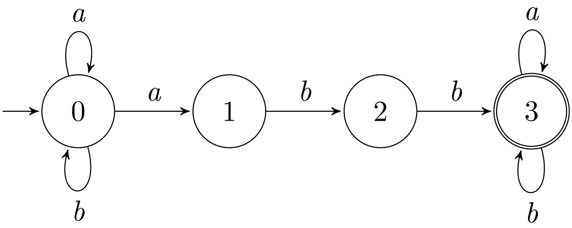


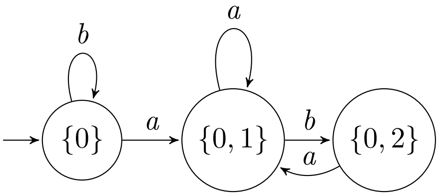
Automate déterministe



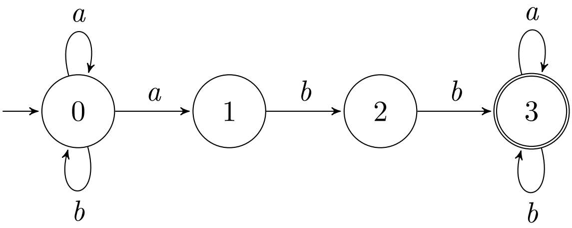


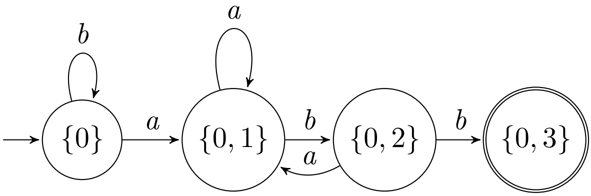
Automate déterministe



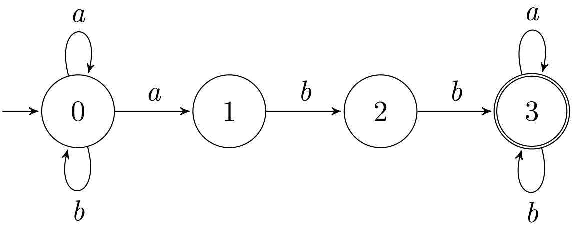


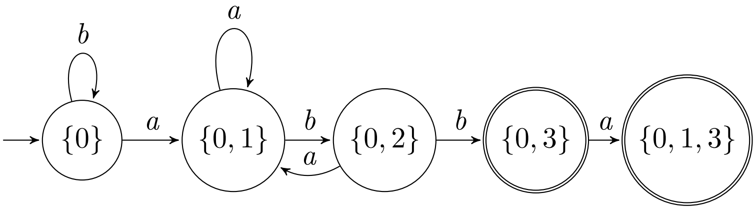
Automate déterministe



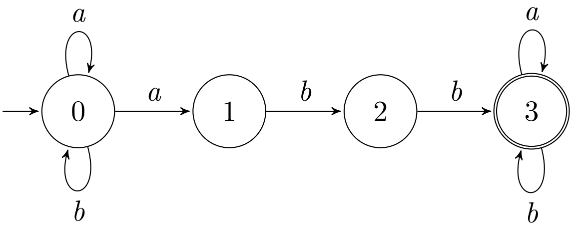


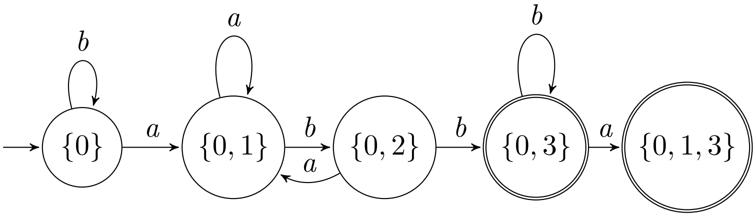
Automate déterministe



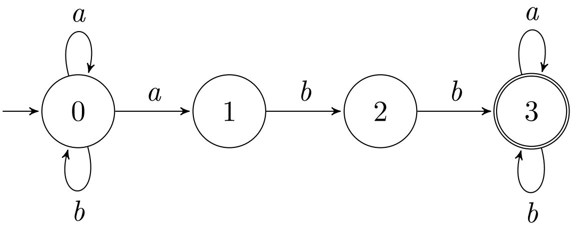


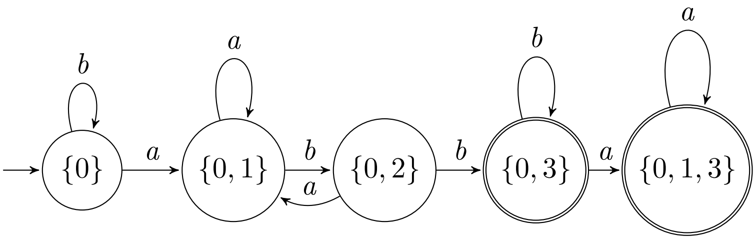
Automate déterministe



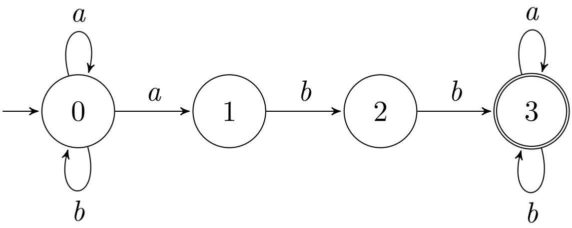


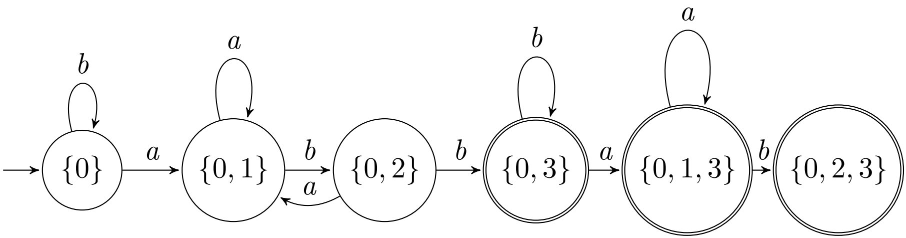
Automate déterministe



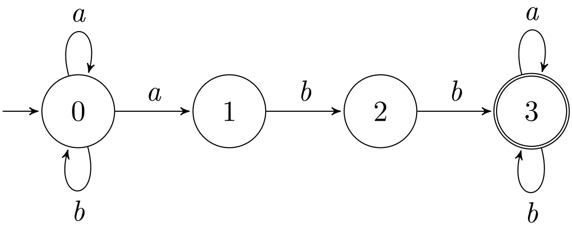


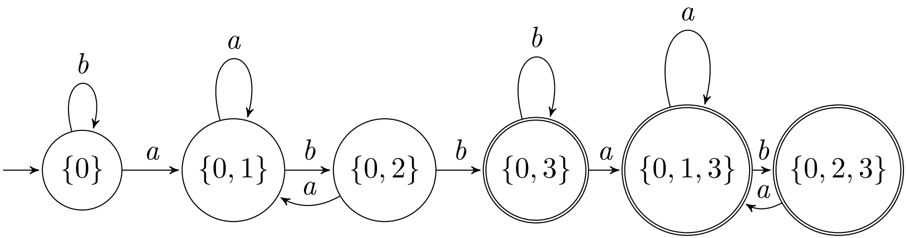
Automate déterministe



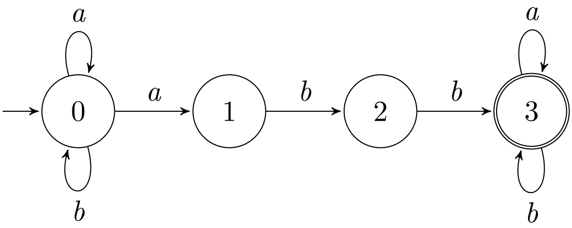


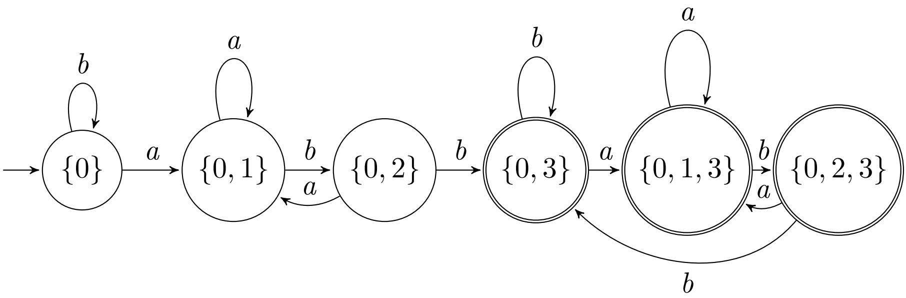
Automate déterministe





Automate déterministe





Automate déterministe

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

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 $O(|m|)$.

Automate déterministe

Exercice

Soit $\Sigma = \{a, b\}$, $n \in \mathbb{N}$ et $L_n = \Sigma^* a \Sigma^n$.

1. Montrer que L_n est reconnaissable par un automate non-déterministe à $n + 2$ états.
2. Montrer que L_n est reconnu par un automate déterministe à 2^{n+2} états.
3. Montrer que L_n ne peut pas être reconnu par un automate déterministe à moins de 2^n états.

Stabilité des langages reconnaissables

Théorème

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

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.

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).

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

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

Théorème

Soient L_1 et L_2 deux langages reconnaissables. Alors :

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

Stabilité des langages reconnaissables

Théorème

Soient L_1 et L_2 deux langages 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 $A_k = (\Sigma, Q_k, i_k, F_k, \delta_k)$ un automate déterministe complet reconnaissant L_k où $k \in \{1, 2\}$.

L'**automate produit** $A_1 \times A_2 \stackrel{\text{def}}{=} (\Sigma, Q_1 \times Q_2, (i_1, i_2), F, \delta)$, où $\delta((q_1, q_2), a) = (\delta_1(q_1, a), \delta_2(q_2, a))$, reconnaît :

Stabilité des langages reconnaissables

Théorème

Soient L_1 et L_2 deux langages 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 $A_k = (\Sigma, Q_k, i_k, F_k, \delta_k)$ un automate déterministe complet reconnaissant L_k où $k \in \{1, 2\}$.

L'**automate produit** $A_1 \times A_2 \stackrel{\text{def}}{=} (\Sigma, Q_1 \times Q_2, (i_1, i_2), F, \delta)$, où $\delta((q_1, q_2), a) = (\delta_1(q_1, a), \delta_2(q_2, a))$, reconnaît :

- $L_1 \cap L_2$ si $F = F_1 \times F_2$
- $L_1 \cup L_2$ si $F = F_1 \times Q_2 \cup Q_1 \times F_2$
- $L_1 \backslash L_2$ si $F = F_1 \times (Q_2 \backslash F_2)$

Stabilité des langages reconnaissables

Question

Donner 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.

Stabilité des langages reconnaissables

Question

Donner 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.

Question

Un préfixe strict d'un mot u est un mot v tel que $u \neq v$ et $u = vw$ pour un certain mot w.
On définit $\min(L)$ comme l'ensemble des mots du langage L dont aucun préfixe strict n'est dans L.
Montrer que si L est reconnaissable alors $\min(L)$ est reconnaissable.

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.

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

Décrire un algorithme en complexité linéaire pour déterminer les états accessibles et co-accessibles d'un automate.

États accessibles et co-accessibles

Définition

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

Théorème

Tout automate est équivalent à un automate émondé.

Lemme de l'étoile

Lemme de l'étoile

Soit L un langage reconnaissable par un automate à n états.
Si $u \in L$ et $|u| \geq n$ alors il existe des mots x, y, z tels que :

- $u = xyz$
- $|xy| \leq n$
- $y \neq \varepsilon$
- $xy^*z \subset L$ (c'est-à-dire : $\forall k \in \mathbb{N},\ xy^kz \in L$)

Lemme de l'étoile

Lemme de l'étoile

Soit L un langage reconnaissable par un automate à n états.
Si $u \in L$ et $|u| \geq n$ alors il existe des mots x, y, z tels que :

- $u = xyz$
- $|xy| \leq n$
- $y \neq \varepsilon$
- $xy^*z \subset L$ (c'est-à-dire : $\forall k \in \mathbb{N},\ xy^kz \in L$)

<u>Preuve</u> :
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 :

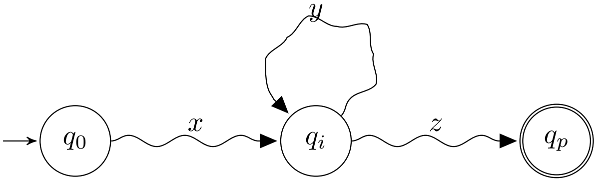
$$q_0 \in I \xrightarrow{u_0} q_1 \xrightarrow{u_1} \dots \xrightarrow{u_{p-1}} q_p \in F$$

Lemme de l'étoile

Preuve (suite) :

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

C a $p+1 > n$ 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.

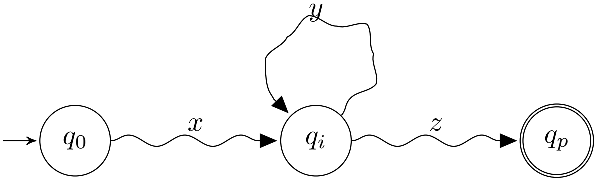


Lemme de l'étoile

Preuve (suite) :

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

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

Lemme de l'étoile

Exercice

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

Preuve :

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 par un automate à n états.
Soit $u = a^n b^n$. Clairement, $u \in L_1$ et $|u| \geq n$.
D'après le lemme de l'étoile : il existe x, y, z tels que $u = xyz$, $|xy| \leq n$, $y \neq \varepsilon$ et $xy^*z \subset L$.
Comme $|xy| \leq n$, x et y ne contiennent que des a : $x = a^i$ et $y = a^j$.
Comme $y \neq \varepsilon$, $j > 0$.
En prenant $k = 0$: $xy^0z = xz = a^{n-j}b^n \notin L_1$: absurde.

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

Lemme de l'étoile

Exercice

Montrer que $L_2 = \{a^n b^p \mid n \neq p\}$ n'est pas un langage reconnaissable.

Lemme de l'étoile

Exercice

Montrer que $L_2 = \{a^n b^p \mid n \neq p\}$ n'est pas un langage reconnaissable.

Preuve :

Si L_2 était reconnaissable alors $\overline{L_1}$ le serait aussi, ce qui est faux.