

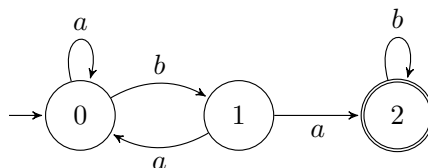
I Automate (non déterministe)

Définition : Automate (non déterministe)

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

Exemple : $A_1 = (\Sigma, Q, I, F, E)$ où $\Sigma = \{a, b\}$, $Q = \{0, 1, 2\}$, $I = \{0\}$, $F = \{2\}$ et $E = \{(0, a, 0), (0, b, 1), (1, a, 0), (1, a, 2), (2, b, 2)\}$.



Représentation graphique de A_1 . Les états finaux sont représentés par des doubles cercles et les états initiaux par des flèches entrantes.

Définition : Fonction de transition

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

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

On dit qu'il y a un blocage lorsque $\delta(q, a) = \emptyset$ (pas de transition possible depuis q avec la lettre a).

Exercice 1.

Donner la fonction de transition de A_1 dans le tableau suivant.

état q	0	1	2
$\delta(q, a)$			
$\delta(q, b)$			

Quelques possibilités d'implémentation de la fonction de transition :

- une matrice (en stockant le tableau ci-dessus)
- une fonction OCaml de type `int -> char -> int list`
- un dictionnaire où chaque clé est un couple (q, a) auquel est associé $\delta(q, a)$

Implémentations possibles d'un dictionnaire :

- Par arbre binaire de recherche où chaque nœud est un couple (clé, valeur). Les clés doivent être ordonnées. Le module `Map` d'OCaml en fournit une implémentation immuable (une modification renvoie un nouvel arbre).
- Par table de hachage, composée d'un tableau contenant les valeurs et 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 . Les clés doivent être hachables. C'est une implémentation mutable (on peut modifier les éléments de la table de hachage).

Exemple avec la table de hachage `Hashtbl` de la bibliothèque standard OCaml :

```

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

Fonction	Type	Description
create	<code>int -> ('a, 'b) Hashtbl.t</code>	crée une table de hachage
add	<code>('a, 'b) Hashtbl.t -> 'a -> 'b -> unit</code>	ajoute une clé et sa valeur
find	<code>('a, 'b) Hashtbl.t -> 'a -> 'b</code>	renvoie la valeur associée à une clé (exception si non trouvée)
find_opt	<code>('a, 'b) Hashtbl.t -> 'a -> 'b option</code>	renvoie Some v où v est la valeur associée à une clé ou None si non trouvée
mem	<code>('a, 'b) Hashtbl.t -> 'a -> bool</code>	teste si une clé est présente

Fonctions du module `Hashtbl` (non exigibles)

II Langage reconnaissable

Soit $A = (\Sigma, Q, I, F, E)$ un automate.

Définition : Chemin acceptant

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 $q_n \in F$.

Définition : Langage reconnu par un automate

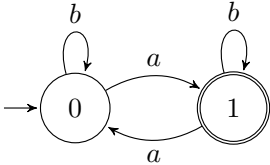
Un mot u est accepté par A s'il est l'étiquette d'un chemin acceptant.

Le langage $L(A)$ reconnu (ou accepté) par A est l'ensemble des mots acceptés par A .

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

Exercice 2.

Le langage reconnu par l'automate ci-dessous est : _____



Exercice 3.

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 reconnaissable.

III Test d'appartenance au langage d'un automate

Une possibilité pour savoir si un automate A accepte un mot $m = m_1...m_n$:

- On part de l'ensemble $Q_0 = I$ des états initiaux (ce qui traite aussi le mot vide).
- 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 $Q_n \cap F \neq \emptyset$).

Exercice 4.

1. 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`.
2. Écrire une fonction `accepte (a : automate) (m : string)` qui détermine si le mot `m` est accepté par l'automate `a`.
3. Quelle est la complexité de `accepte` ?

Remarque : on peut aussi utiliser du backtracking (retour sur trace) en essayant, pour chaque lettre, une transition possible et en revenant en arrière si on est bloqué ou si la lecture se termine dans un état non final.

IV Automate complet

Définition : Automates équivalents

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

Définition : Automate complet

Un automate (Σ, Q, I, F, E) est complet si : $\forall q \in Q, \forall a \in \Sigma, \exists q' \in Q, (q, a, q') \in E$

Autrement dit : un automate est complet s'il n'a pas de blocage.

Théorème

Tout automate est équivalent à un automate complet.

Preuve :

V Automate déterministe

Définition : Automate déterministe ♡

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

Si A 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 $\delta : Q \times \Sigma \longrightarrow Q$.

Exercice 5.

Si A est déterministe complet alors son nombre de transitions est : _____

Définition : Fonction de transition étendue

Si A est déterministe et 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 :

- $\delta^*(q, \varepsilon) = q$
- Pour $a \in \Sigma$ et $v \in \Sigma^*$, $\delta^*(q, av) = \delta^*(\delta(q, a), v)$

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

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

Attention : La définition ci-dessus de δ^* suppose l'automate déterministe et complet.

Théorème

Soit A un automate déterministe complet, q un état et u, v des mots. Alors :

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

Preuve :

Un automate déterministe complet peut être représenté par le type plus simple :

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

Pour un automate déterministe complet, le code suivant teste l'acceptation d'un mot `m` en temps moyen $O(|m| + |F|)$, sous l'hypothèse d'accès en temps moyen constant à la table de hachage. Le terme $|F|$ vient du test final dans une liste :

```
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
```

Théorème : Déterminisation ♡

Soit A un automate. Alors A est équivalent à un automate déterministe complet.

Preuve : A est équivalent à l'automate des parties $A' = (\Sigma, \mathcal{P}(Q), \{I\}, F', \delta')$ où $F' = \{X \subseteq Q \mid X \cap F \neq \emptyset\}$ et

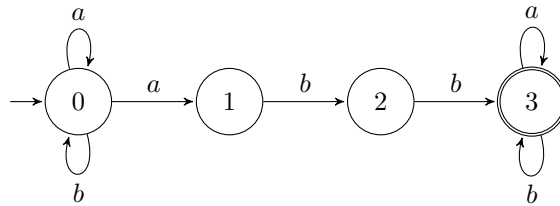
$$\delta'(X, a) = \bigcup_{q \in X} \delta(q, a).$$

Remarques :

- L'état $\emptyset$ est similaire à l'état q_∞ utilisé pour rendre un automate complet.
- A' possède $2^{|Q|}$ états et $2^{|Q|} \times |\Sigma|$ transitions, donc est de taille exponentielle en la taille de A .
- En pratique, on construit l'automate des parties de proche en proche en partant de l'état initial (comme un parcours de graphe) et on ne dessine que les états de $\mathcal{P}(Q)$ qui sont atteignables.

Exercice 6.

Déterminiser l'automate suivant. ♡

**Exercice 7.**

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.

VI Stabilité des langages reconnaissables

Théorème : Stabilité par complémentaire ♡

Soit L un langage reconnaissable, sur un alphabet Σ . Alors $\overline{L} \stackrel{\text{def}}{=} \Sigma^* \setminus L$ est reconnaissable.

Preuve :

Exercice 8.

Montrer qu'il est nécessaire d'utiliser un automate déterministe complet dans la preuve précédente.

Exercice 9.

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.

Théorème : Stabilité par intersection, union et différence ♡

Soient L_1 et L_2 deux langages reconnaissables sur le même alphabet Σ . Alors :

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

Preuve ♡:

Pour $k \in \{1, 2\}$, soit $A_k = (\Sigma, Q_k, i_k, F_k, \delta_k)$ un automate déterministe complet reconnaissant L_k .

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 $F =$ _____
- $L_1 \cup L_2$ si $F =$ _____
- $L_1 \setminus L_2$ si $F =$ _____

Remarques :

- $A_1 \times A_2$ simule les deux automates A_1 et A_2 en parallèle, sur chaque composante du couple.
- Si L_1 et L_2 sont sur deux alphabets différents, Σ_1 et Σ_2 , on peut considérer $\Sigma_1 \cup \Sigma_2$, puis compléter les automates sur cet alphabet avant de construire leur produit.
- On peut généraliser aux intersections et unions finies de langages.

Exercice 10.

La preuve fonctionne-t-elle si A_1 et A_2 ne sont pas déterministes complets ?

Exercice 11.

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

VII États accessibles et co-accessibles

Définition : États accessibles et co-accessibles

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.

Exercice 12.

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

Définition : Automate émondé

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

Théorème

Tout automate est équivalent à un automate émondé.

Preuve : On peut supprimer les états inaccessibles et les états non co-accessibles, sans changer le langage reconnu.

Exercice 13.

Est-il vrai que tout automate est équivalent à un automate émondé déterministe complet ?

VIII Lemme de l'étoile

Théorème : 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 :

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

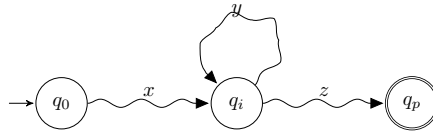
Preuve ♡:

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 de la forme :

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

En posant $p = |u|$, les $n + 1$ premiers états de C contiennent deux occurrences d'un même état : $q_i = q_j$ avec $0 \leq i < j \leq n$. La partie de C entre q_i et q_j forme donc un cycle :



Soit $x = u_0u_1\dots u_{i-1}$, $y = u_i\dots u_{j-1}$ et $z = u_j\dots u_{p-1}$. On a $u = xyz$, $|xy| = j \leq n$ et $|y| = j - i > 0$. Alors xy^kz est l'étiquette du chemin acceptant obtenu à partir de C en passant k fois dans le cycle. D'où : $\forall k \in \mathbb{N}, xy^kz \in L$. $\square$

Attention :

- La réciproque du lemme de l'étoile est fausse : on ne peut pas l'utiliser pour montrer qu'un langage est reconnaissable.
- La décomposition xyz est donnée par le lemme de l'étoile, on ne peut pas la choisir.

Schéma de preuve pour montrer qu'un langage L n'est pas reconnaissable :

- Supposons que L soit reconnaissable par un automate à n états.
- Soit $u = \dots \in L$ tel que $|u| \geq n$.
- Soit $u = xyz$ la décomposition donnée par le lemme de l'étoile.
- On remarque que $xy^kz \notin L$ pour $k = \dots$
- C'est absurde : L n'est pas reconnaissable.

Exercice 14.

1. Montrer que $L_1 = \{a^n b^n \mid n \in \mathbb{N}\}$ n'est pas reconnaissable.
2. Montrer que $L_2 = \{a^n b^p \mid n \neq p\}$ n'est pas reconnaissable.
