Chap. 1 Automates finis

Introduction :

Bien avant l'apparition des ordinateurs, les mathématiciens se sont intéressés à l'étude des processus de calcul.

Alan Turing a décrit à 1936 un modèle de "machine idéale" auquel il a laissé son nom. D'autres modèles furent proposés, qui sont tous équivalents à la machine de Turing. Church a démontré dans sa thèse que ces modèles sont les plus généraux possibles.

Ces travaux amenèrent à distinguer entre les fonctions qui sont calculables en théorie de celles qui ne le seraient même pas.

Si le logicien est fasciné par le tracé de la frontière entre ces deux familles de fonctions l'informaticien a déjà fort à faire avec celles qui sont réputées calculables.
On est naturellement conduit à classer les fonctions calculables suivant la "complexité" du calcul nécessaire pour les évaluer. Cette complexité peut être mesurée suivant 2 critères : le nombre des opérations élémentaires – ce qu'on appelle "le temps" et la taille de la mémoire nécessaire – "l'espace".

La complexité en temps est la plus directement compréhensible. La complexité en espace amène à des distinctions subtiles.

Le modèle de Turing suppose que la machine dispose d'une mémoire potentiellement infinie pour effectuer ses calculs, ce qui n'est pas réalisable matériellement.

Le modèle le plus simple est celui dans lequel la taille de la mémoire utilisable est fixée à priori et indépendamment de la taille de la donnée sur laquelle on effectue les calculs. C'est ce modèle qu'on appelle automate fini.

Automates finis fournissent un outil de construction d'algorithmes particulièrement simple.

Définitions

mots sur l'alphabet A

La nature de l'alphabet peut être très variable.

$w = abacda$ – mot de longueur 6 sur l'alphabet latin.

mot vide est noté par ε ou par 1. C'est le seul mot de longueur 0.

A^* - l'ensemble de tous les mots sur l'alphabet A.

Un automate fini sur l'alphabet A est donné par

- un ensemble fini Q dont les éléments sont des états de l'automate.

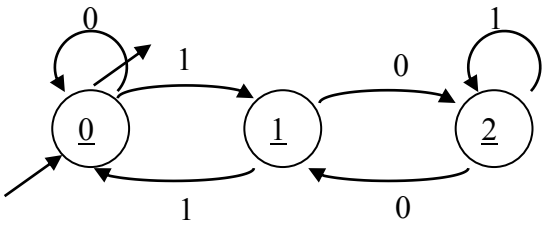
- un ensemble $E \subset Q \times A \times Q$ de triplets ***(p.a.q)*** appelés les flèches ou les transitions de l'automate.
- un ensemble $I \subset Q$ d'états initiaux
- un ensemble $T \subset Q$ d'états terminaux

Exemple 1: un automate pour reconnaitre les multiples de n :
Cet automate a n états numérotés de 0 à n-1, chaque état correspond au reste de division par n.
Par exemple : n=3 (écriture binaire) Etats : 0 , 1 , 2 A={0,1}

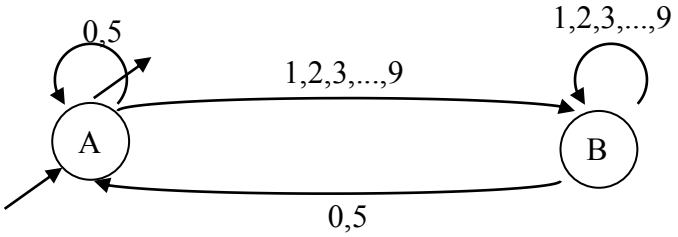
Q={0 , 1 , 2 } T={0 } terminal I={0} initial

K ∈ N

	0	**1**
0 (3k)	0 (6k)	1 (6k+1)
1 (3k+1)	2 (6k+2)	0 (6k+3)
2 (3k+2)	1 (6k+4)	2 (6k+5)



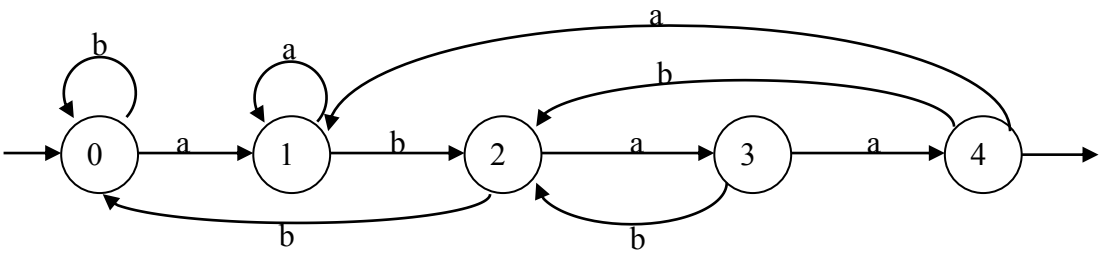
Exemple 2: un automate pour reconnaitre les multiples de 5 en décimal :
A= {0,1,2,…,9}
Q= {A,B}
T= {A} terminal
I= {A} initial



L'interprétation intuitive :

A chaque instant l'automate se trouve dans l'un des états ***p*** de ***Q***. Il lit alors l'une des lettres ***a*** de l'alphabet ***A*** et passe dans un état ***q*** tel que ***(p.a.q)*** soit une flèche.

Exemple : mot que se termine par abaa



Q= {0,1,2,3,4}
T= {4}terminal
I= {0} initial
E= {0.a.1 , 0.b.0 , 1.a.1 , 1.b.2 , 2.a.3 , 2.b.0 , 3.a.4 , 3.b.2 , 4.a.1 , 4.b.2}
La lecture du mot W se termine par 4 SSI W se termine par abaa.
a a a b a a b a b b a b a a

L'automate est dans l'état i SSI le suffixe le plus long du mot déjà lu que est en même temps le préfixe de a b a a est de longueur i .

Suffixes	Préfixes
0 : abb	abaa
b	a
bb	ab
abb	aba
	abaa

	Suffixes					
1:	baaa	2 : aab		3 : aaab	4 : abaa	
	a	b		aba		
	aa	ab				
	aaa	aab				
	baaa					

La signification :

Si vous regardez cet automate de près, vous verrez qu'il se trouve dans l'état numéro **i** SSI le plus long **suffixe du mot déjà lu** qui est en même temps un **préfixe de abaa,** est de **longueur i**.

Cet automate réalise de façon optimale l'algorithme de recherche du mot ***abaa*** dans un texte : il résout pour ce mot l'algorithme du "string matching" utilisé par exemple dans les éditeurs de textes.

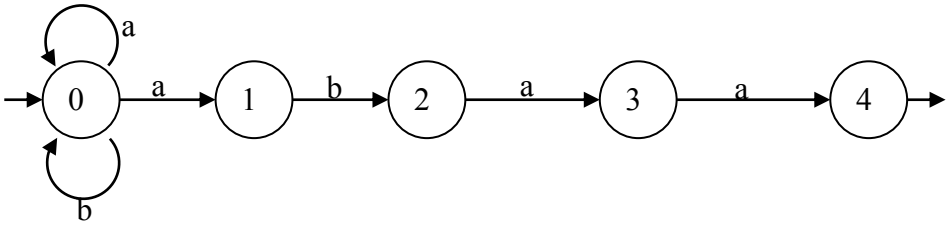
Définitions :

La façon dont on rend compte de façon précise de ce que "fait" un automate fini est la suivante : on dit qu'un mot **w ∈ A*** est reconnu par l'automate fini s'il existe un moyen en partant d'un état initial **i ∈ I** et en lisant le mot **w** d'atteindre un état terminal **t ∈ T** à la fin de la lecture.

Le **langage reconnu** par l'automate fini est l'ensemble de tous les mots reconnus.

Automates déterministes

Voici l'exemple d'un automate ***non-déterministe*** :



A = {a.b}
I ={0}
T = {4}
E = {0.a.b , 0.b.0 , 0.a.1 , 1.b.2 , 2.a.3 , 3.a.4 }
Il reconnaît tous les mots qui se terminent par *abaa*. Très facile à construire.

Pourquoi n'est-il pas déterministe ?
Ambigüité : la lecture de **a** à partir de 0 peut nous ramener dans l'état 0 ou l'état 1 !
Pour atteindre 4 à partir de 0, il n'existe qu'un seul moyen : lire **abaa**.
Ceci n'est pas possible pour un automate déterministe (définition formelle un peu plus bas).

On va montrer que l'on peut toujours construire, à partir d'un automate fini (non déterministe), un autre automate qui est déterministe et qui reconnaît le même langage.

Soit $A = (Q, I, T, E)$ un automate fini.

Pour un mot $w \in A^*$ on note $p \rightarrow q$ s'il existe un chemin de l'état p à l'état q obtenu en lisant w.

Définitions formelles :

Pour $\forall a \in A$, on a $p \xrightarrow{a} q$ (flèche de p à q étiquetée par a) SSI il existe une flèche $(p.a.q) \in E$ (ensemble des flèches).
Ensuite, pour $u \in A^*$ et $a \in A$ on a $p \xrightarrow{ua} q$ ssi $\exists\, r \in Q$ tq $p \xrightarrow{u} r$ et $r \xrightarrow{a} q$.
Quand on a $p \xrightarrow{w} q$ on va dire qu'il existe un chemin de p à q d'étiquette w.
On peut facilement voir (par récurrence) que pour $u,v \in A^*$ on a $p \xrightarrow{uv} q$
SSI $\exists\, r \in Q$ tq $p \xrightarrow{u} r$ et $r \xrightarrow{v} q$.
a et u sont des lettres et s'il y a un mot(v) et une lettre à la fin, il y a un état intermédiaire.

Automate déterministe, définition

L'automate est déterministe SSI pour $\forall p \in Q$ et $\forall a \in A$, il existe au plus un seul état d'arrivée quelconque $q \in Q$ tel que $p.a.q$ ($p \xrightarrow{a} q$), et qu'il y ait un seul état initial.

A n'importe quel automate non-déterministe, correspond un automate déterministe qui reconnait le même langage.

On notera ***p.a=q***

Si ***A*** est un automate déterministe on peut vérifier par récurrence sur la longueur des mots que pour $\forall p \in Q$ et $\forall$ mot $u \in A^*$ il existe au plus un état $q \in Q$ tq $p \xrightarrow{u} q$.

On notera ***p.u=q*** en convenant que ***p.u*** n'est pas défini quand il n'existe pas de tel état ***q***; et on aura pour deux mots ***u, v*** $\in A^*$

(p.u).v=p.(uv)

Algorithme de déterminisation

Soit ***A = {Q, I, T, E}*** un automate fini non-déterministe sur l'alphabet A.
Un automate ***D*** déterministe correspond à ***A***. Il prend :

- comme ensemble d'états un sous-ensemble ***P*** de l'ensemble ***π*** des parties de ***Q***.
  exemple : Q={0 , 1 , 2 , 3} ➔ ∅ , (0) , (1) , (2) , (01) , (02) , (12) , (012) ➔ $\pi = 2^3 = 8$
  Si ***Q*** a ***n*** éléments, alors ***π*** a 2^n parties.

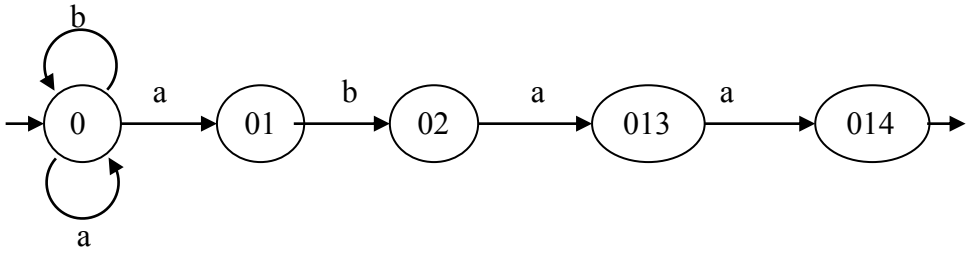
- comme ensemble ***ζ*** d'états terminaux des parties de **Q** qui contiennent au moins un état terminal de ***A***.
  ζ = { u∈P|u∩T≠∅ }

- comme ensemble de flèches $(u \xrightarrow{a} v)$ SSI : $u \in P$ et $v \in P$ est l'ensemble de tous les états $q \in Q$ tels qu'il existe un état p dans u (avec $p \xrightarrow{a} q \in E$).

- comme unique état initial l'ensemble **I** des états initiaux de ***A***

L'algorithme s'explique le plus facilement sur un exemple :

Exemple de déterminisation :

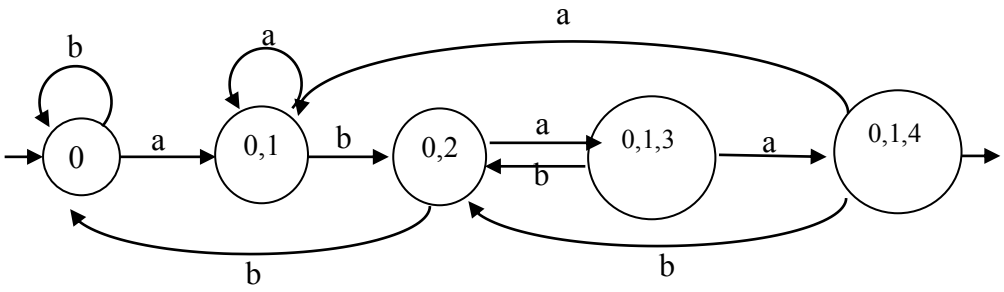


Etat	**a**	**b**
(0)	(0,1)	(0)
(0,1)	(0,1)	(0,2)
(0,2)	(0,1,3)	(0)
(0,1,3)	**(0,1,4)**	(0,2)
(0,1,4)	(0,1)	(0,2)

Explications

On commence par l'état initial qui dans ce cas particulier coïncide avec l'état initial de l'automate non déterministe (0), car notre automate non déterministe n'en a qu'un seul. (S'il en avait plusieurs, on les aurait rejoint pour former l'état initial de l'automate déterministe).

A partir de cet état (0), en ***a*** on va vers les états (0) , (1) de l'automate initial. Pour l'automate déterministe, on les regroupe dans l'état (0,1). En ***b***, on va vers l'état (0) de l'automate initial. Donc, (0) est aussi un état de l'automate déterminisé.
Chaque fois qu'on obtient un état de l'automate déterminisé, on le met dans la colonne de gauche pour agir sur cet état par les lettres de l'alphabet. Ainsi on obtient la deuxième ligne, où figure un nouvel état (0,2). On procède ainsi tant qu'il y reste des états non traités.
Les états finaux sont ceux qui contiennent un état final de l'automate initial. Dans notre cas, il n'y a qu'un seul : (0,1,4).
Voici l'automate déterminisé :



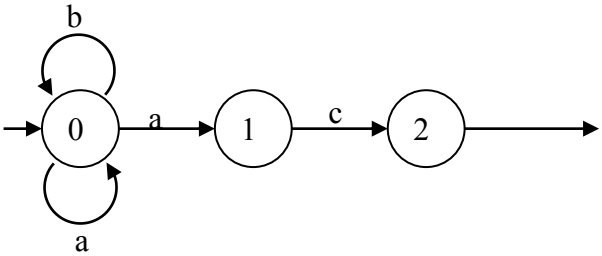
La démonstration montrant que ***D*** et ***A*** reconnaissent le même langage est fait par récurrence.

Automate complet (définition) :

Un automate ***D*** est complet SSI pour $\forall p \in P$ et $a \in A$, il existe $q \in P$ tel que $p \xrightarrow{a} q$
C'est-à-dire que de chaque état sortent des flèches avec toutes les étiquettes possibles.
L'automate ***D*** de l'exemple précédent est un automate complet.

N'importe quel automate non déterministe est équivalent à un automate déterministe et complet : il suffit, dans l'automate déterminisé, d'introduire un état **poubelle**.

Exemple : langage = ***a*** est en avant dernière lettre, ***c*** est la dernière, et les mots contiennent un seul ***c***.



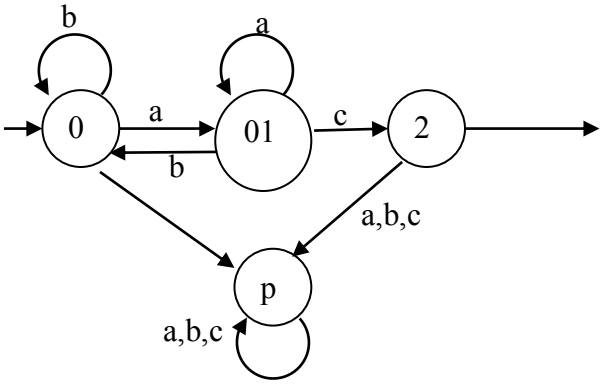
Déterminisation :

état	**a**	**b**	**c**
(0)	(0,1)	(0)	--
(0,1)	(0,1)	(0)	(2)
(2)	--	--	--

Nous remplaçons le « nullepart » dans ce tableau par un nouvel état appelé poubelle (P), qui a la propriété que toutes les transitions à partir de cet état revienne sur lui-même :

Voici donc l'automate déterminisé complet :

état	**a**	**b**	**c**
(0)	(0,1)	(0)	(P)
(0,1)	(0,1)	(0)	(2)
(2)	(P)	(P)	(P)
(P)	(P)	(P)	(P)



Ex : c.a.a➔P
a.a.c.a➔P

Encore quelques définitions :

- Un automate est **accessible** SSI n'importe quel état est accessible à partir de l'état initial.
- Un automate est **coaccessible** SSI à partir de n'importe quel état on peut accéder à un état terminal. (état poubelle ➔ non coaccessible !)
- Accessible + coaccessible = **émondé**.
- Un ensemble de mots ***X*** est un langage **reconnaissable** SSI il existe un automate fini ***D*** qui le reconnaît.

Le complément d'un langage reconnaissable est encore reconnaissable.

Si ***X*** est un langage reconnaissable sur l'alphabet ***A***, on peut le reconnaître avec un automate déterministe et complet (car l'automate ***D*** figurant dans la définition du langage reconnaissable est équivalent à un automate déterministe et complet ***F***).

F=(Q, i, T, E) (ici, ***i*** est l'entrée – un automate déterministe n'en a qu'une seule)

On a alors $w \in X$ SSI $i.w \in T$ et donc $w \notin X$ SSI $i.w \notin T$.

Rappel ***Tous*** les automates que nous considérons dans ce cours, sont des automates finis ! U

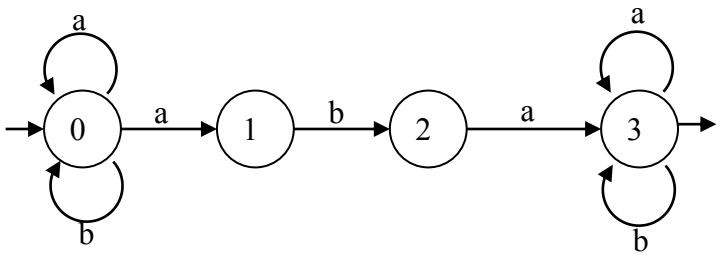
Le complément d'un langage reconnaissable.

Soit ***D*** un automate déterministe et complet reconnaissant le langage ***X***.
Pour construire un automate reconnaissant le complément de ***X*** , il suffit de prendre comme ensemble d'états terminaux le complément de l'ensemble d'états terminaux de ***D***.

Exemple.

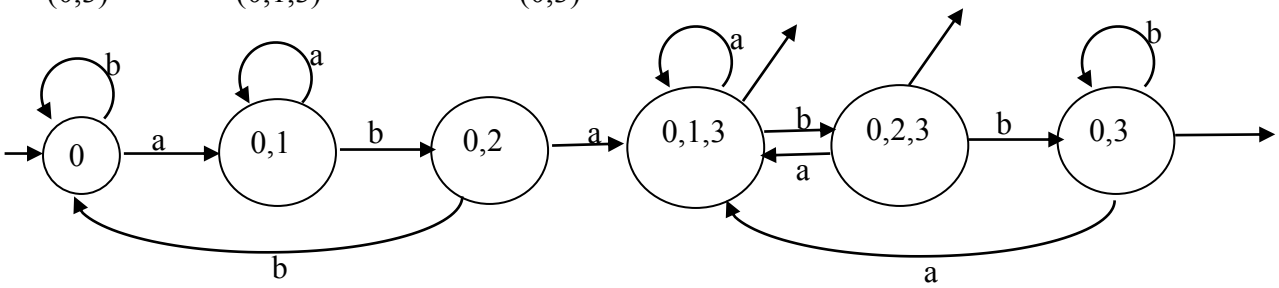
L'automate sur A={a,b} qui reconnaît l'ensemble des mots n'ayant pas ***aba*** en facteur.

On commence par la construction d'un automate (non déterministe) reconnaissant tous les mots ayant ***aba*** en facteur.

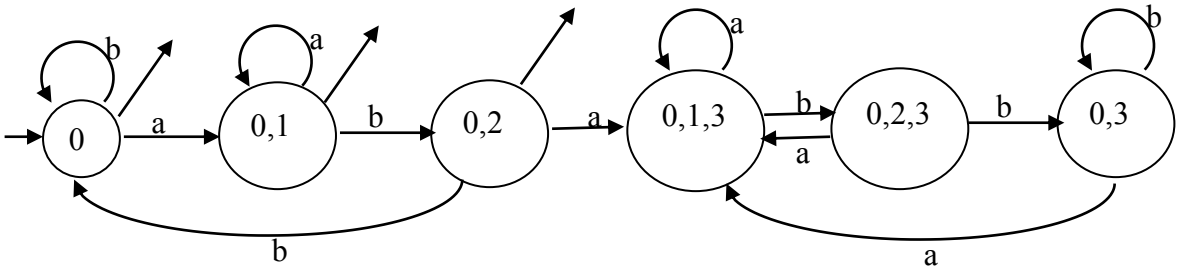


Appliquons l'algorithme de déterminisation.

état	**a**	**b**
(0)	(0,1)	(0)
(0,1)	(0,1)	(0,2)
(0,2)	(0,1,3)	(0)
(0,1,3)	(0,1,3)	(0,2,3)
(0,2,3)	(0,1,3)	(0,3)
(0,3)	(0,1,3)	(0,3)



Cet automate déterministe et complet reconnaît tous les mots ayant ***aba*** en facteur. Il a trois états terminaux : (0,1,3), (0,2,3) et (0,3).
Maintenant pour obtenir un automate (lui aussi déterministe et complet) reconnaissant tous les mots qui n'ont pas ***aba*** en facteur, il suffit de faire de tous les états terminaux, des états non terminaux, et vice versa :



Remarque très importante :
Ici, on a obtenu un automate complet sans qu'il y ait besoin de le compléter en introduisant l'état poubelle. Evidemment, ceci n'est pas toujours le cas. **Si on a affaire à un automate non complet, on n'a pas le droit de remplacer directement les états finaux par des états**

non finaux et vice versa : il faut d'abord le compléter. Alors l'état poubelle (qui n'est pas final) devient un état final de l'automate reconnaissant le complément du langage.

Minimisation

Pour tout langage reconnaissable il existe ***le plus petit*** (c.à.d. contenant le plus petit nombre d'états) automate déterministe qui le reconnaît, et il est ***unique*** : c'est l'automate minimal du langage.

Algorithme de minimisation par la méthode des équivalences est basé sur la notion de partitionnement de l'ensemble d'états de l'automate.

Principe

Tout d'abord , si l'automate à minimiser n'est pas complet, il faut lui ajouter l'état poubelle pour qu'il le devienne. Sinon on risque de faire une erreur grave, qui se manifeste facilement au cas d'un automate déterministe non complet dont tous les états sont des états terminaux (essayez le voir vous-mêmes après avoir compris l'algorithme de minimisation).

On sépare tous les états de l'automate déterministe initial en deux groupes : terminaux et non-terminaux. Puis, on passe par des étapes (par itérations). A chaque étape, on repartitionne un ou des groupes, jusqu'à ce qu'on arrive à ne plus pouvoir partitionner. Les groupes restants forment l'automate minimal.

Partitionnement d'un groupe d'états

Soit un groupe d'états $M = \{s_1, s_2, \ldots, s_k\}$ appartenant à la partition courante, et une lettre a.

Si les transitions sur a tombent dans deux ou plus groupes différents de la partition courante (c.à.d. de l'étape en cours), il faut séparer M de telle façon que toutes les transitions à partir des états qui appartiennent à un seul groupe tombent dans le même groupe de la partition courante.

Supposons qu'en lisant a à partir de s_1 et s_2 , on arrive dans t_1 et t_2 qui appartiennent à deux groupes différents de la partition courante. Alors il faut couper M dans au moins deux sous-ensembles, de telle façon que s_1 appartienne à un sous-ensemble, et s_2 appartienne à l'autre.

Puis on répète ce processus de séparation jusqu'à ce qu'on ne puisse plus séparer.

Description du processus de partitionnement itératif

Soit
un automate fini déterministe complet $D=(Q, i, T, E)$
Résultat à obtenir: Un un automate fini déterministe complet D' qui reconnaît le même langage que D et qui a aussi peu d'états que possible.

Méthode

1. Construire une partition initiale Θ_0 de l'ensemble des états avec deux groupes : les états terminaux T et les états non-terminaux $Q-T$.

2. Procédure applicable à la partition courante Θ, à commencer par Θ_0 :

POUR chaque groupe G de Θ **FAIRE**

début

partitionner G en sous-groupes de manière que deux états e et t de G soient dans le même sous-groupe SSI pour tout symbole $a \in A$, les états e et t ont des transitions sur a vers des états du même groupe de Θ;

/* au pire, un état formera un sous-groupe par lui-même */

remplacer G dans par tous les sous-groupes ainsi formés ; on obtient la partition de l'étape suivant, Θ_{new} ;

fin

3. Si $\Theta_{new} = \Theta$ alors $\Theta_{final} = \Theta$ et continuer à l'étape 4. Sinon, répéter l'étape (2) avec $\Theta = \Theta_{new}$.

4. Choisir un état dans chaque groupe de la partition Θ_{final} en tant que représentant de ce groupe. Les représentants seront les états de l'automate fini déterministe réduit D'.
   Soit e un état représentatif, et supposons que dans D il y a une transition $e \xrightarrow{a} t$.

   Soit r le représentant du groupe de t (r peut être égal à t).
   Alors D' a une transition de e vers r sur a ($e \xrightarrow{a} r$).

L'état initial de D' est le représentant du groupe qui contient l'état initial i de D ; les états terminaux de D' sont des représentants des états de T.

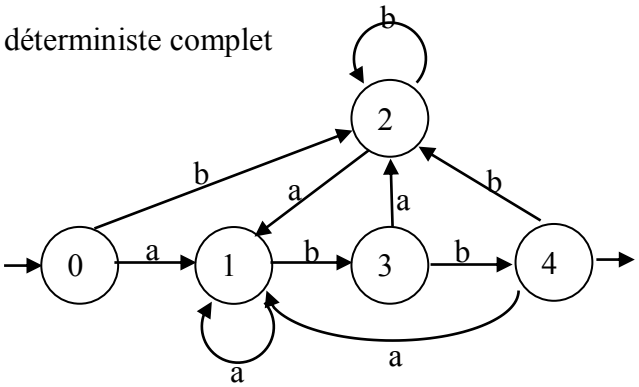
Pour tout groupe G de Θ_{final} , soit G est entièrement composé d'états de T, soit G n'en contient aucun.

Remarque

En réalité, il est parfois plus facile, au lieu de choisir un représentant, marquer les groupes finaux comme tels, ou bien les renommer par, par exemple, A,B, C … , et remplacer dans les transitions tout état par le nom de son groupe :

Si $e \xrightarrow{a} r$, $e \in$A, $r \in$B où les groupes A,B$\in \Theta_{final}$, alors dans l'automate minimisé on a A$\xrightarrow{a}$B où maintenant on considère A et B comme états de l'automate minimisé. Cette remarque deviendra tout à fait claire à la fin de l'exemple suivant.

Exemple 1. Minimisons l'automate déterministe complet



décrit par le tableau de transitions suivant :

état	a	b	
0	1	2	
1	1	3	
2	1	2	
3	2	**4**	
4	1	2	L'unique état terminal : 4.

Donc, la partition initiale : $\Theta = \{(0, 1, 2, 3), (4)\}$

Notons les groupes non terminal et terminal : I = {(0, 1, 2, 3)} et II = {(4)}

On ne peut pas, évidemment, essayer de séparer le groupe II qui consiste déjà en un seul état. On regarde donc dans quel groupe tombent les transitions à partir des états du groupe I :

état	a	b
0	I	I
1	I	I
2	I	I
3	I	II

Donc, le groupe I se sépare en deux, et on a $\Theta_{new} = \{(0, 1, 2\}, (3), (4)\}$.

Notons les groupes (en recyclant la notation) : I = {(0, 1, 2)}, II = {(3)}, III = {(4)}.

Maintenant essayons de séparer le groupe I en regardant où tombent les transitions à partir de ses états dans les termes de la séparation {(0, 1, 2}, (3), (4)}.

état	a	b
0	I	I
1	I	II
2	I	I

Donc, le groupe I se sépare en deux, et on a $\Theta_{new} = \{(0, 2\}, (1), (3), (4)\}$.

Notons les groupes (en recyclant la notation) : I = {(0, 2)}, II = {(1)}, III = {(3)}, III = {(4)}.

Essayons de séparer le groupe I en regardant où tombent les transitions à partir de ses états dans les termes de la séparation {(0, 2}, (1), (3), (4)}.

état	a	b
0	II	I
2	II	I

Le groupe I ne se sépare pas, Θ_{new} reste identique à la séparation de l'étape précédent : $\Theta_{new} = \{(0, 2\}, (1), (3), (4)\}$.

Fin de la procédure itérative de séparation.

Voici les itérations qu'on a effectuées :

Etape 1 : $\Theta = \{(0, 1, 2, 3), (4)\}$
$\Theta_{new} = \{(0, 1, 2\}, (3), (4)\}$

Etape 2 : $\Theta = \{(0, 1, 2\}, (3), (4)\}$
$\Theta_{new} = \{(0, 2\}, (1), (3), (4)\}$

Etape 3 : $\Theta = \{(0, 2\}, (1), (3), (4)\}$
$\Theta_{new} = \{(0, 2\}, (1), (3), (4)\} = \Theta$

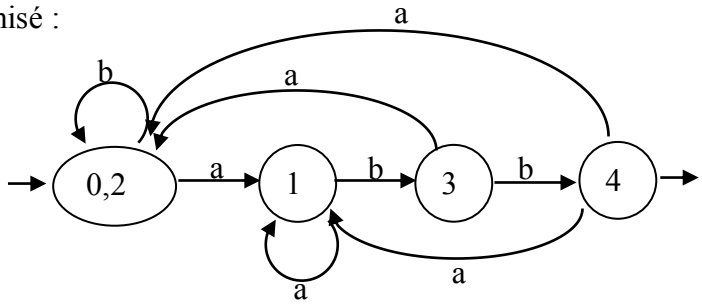
Passons aux représentants :

groupe	représentant

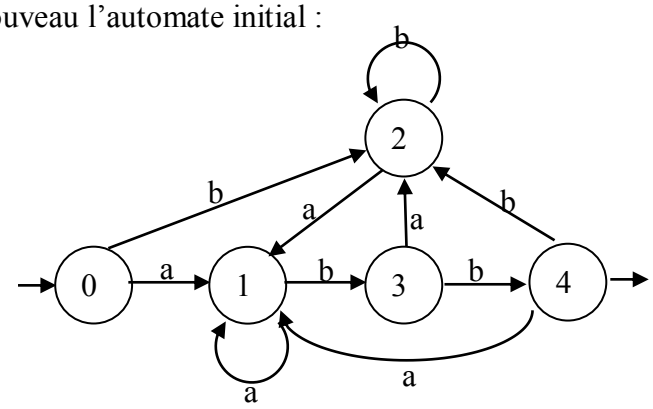
I 0 ou 2
II 1
III 3
IV 4

Il devient clair maintenant qu'on peut marquer les états de l'automate minimisé soit par un représentant, soit par les chiffres I, II, III, IV suivant le groupe du dernier étape, soit par le contenu du groupe (dans ce cas, l'état initial sera marqué (0,2) ; cette dernière solution est pratique tant que le groupe consiste en peu d'états).

L'automate minimisé :



On peut maintenant poser la question suivante : pourquoi les états 0 et 2 ont resté ensemble après la minimisation ? Qu'y a-t-il de spécial concernant ces deux états par rapport aux autres ? Regardons à nouveau l'automate initial :



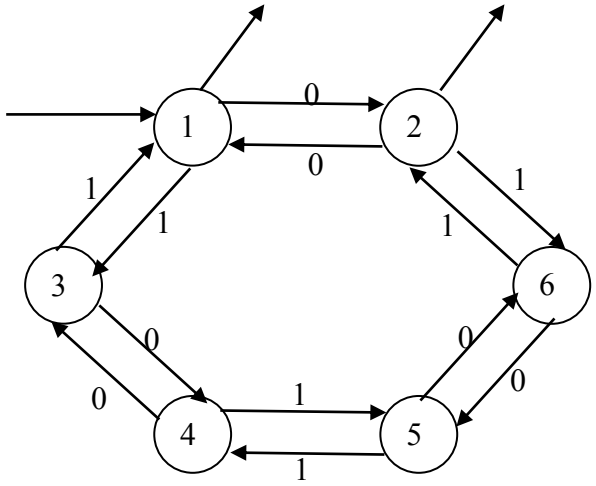
et introduisons la terminologie suivante :

on dit que la chaîne ***w*** **distingue** l'état ***s*** de l'état ***t*** si quand on commence dans l'état ***s*** et lit ***w*** on arrive dans un état terminal, et si on commence dans ***t*** et lit ***w*** on arrive dans un état non terminal ou vice-versa.

Ici, tous les états peuvent être distingués par une chaîne ou une autre, sauf les états 0 et 2. C'est facile à voir : et à partir de 0, et à partir de 2 on arrive en 1 en lisant un nombre quelconque (y compris zéro) de ***b*** et un ***a*** ; puis, comme on est dans le même état (1), il nous restent que les mêmes chaînes pour arriver à l'état final. Donc, les états non distinguables se fondent dans un même état (en l'occurrence (0,2)) de l'automate minimisé.

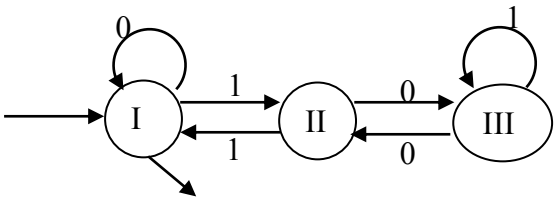
En fait, on peut montrer que l'algorithme de minimisation qu'on a expliqué, est basé sur la recherche des tous les groupes qui peuvent être distingués par une chaîne ou une autre.

Exemple 2 Voici un automate déterministe complet avec deux états terminaux, avec l'alphabet A={0,1} :



La même procédure de minimisation donne :
I = {1} T = {1, 2}
Θ_0={(1,2), (3, 4, 5, 6)}={I, II}
Θ_1= {(1, 2), (3, 6), (4, 5)}={I, II, III} (en recyclant les chiffres romains)
$\Theta_2=\Theta_1$

Voici l'automate minimisé :



Automate asynchrone

Un automate asynchrone c'est un automate fini dans lequel certaines des flèches sont étiquetées par le mot vide (« ***ε-transitions*** »). Ensemble des flèches vérifie donc $E \subset Q \times (A \cup \varepsilon) \times Q$.

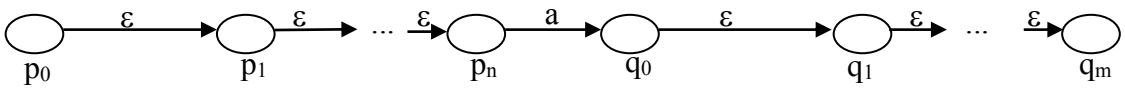
Proposition : Pour tout automate asynchrone, il existe un automate fini ordinaire qui reconnaît le même langage.
Comme tout automate non déterministe est équivalent à un automate déterministe, il en suit que **tout automate asynchrone est équivalent à un automate déterministe**.

Il existe un algorithme de suppressions de ***ε-transitions***.

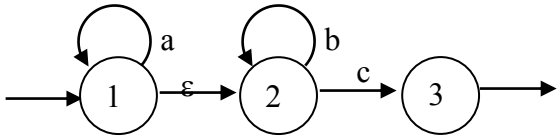
Pour un automate asynchrone $A = (Q, I, T, E)$ avec $E \subset Q \times (A \cup \varepsilon) \times Q$ nous allons construire un automate $B=(Q, I, T, F)$ qui ne diffère de A que par l'ensemble de ses flèches.

Par définition on a $(p.a.q) \in F$ s'il existe un chemin

$c : \quad p_0 \xrightarrow{\varepsilon} p_1 \xrightarrow{\varepsilon} \dots \xrightarrow{\varepsilon} p_n \xrightarrow{\varepsilon} q_0 \xrightarrow{\varepsilon} q_1 \xrightarrow{\varepsilon} \dots \xrightarrow{\varepsilon} q_m$
et $\quad p_0 = p, q_m = q$



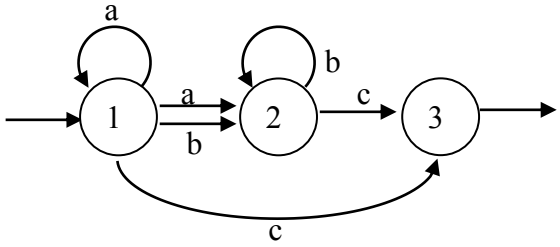
Ex. a) Prenons un automate asynchrone :



(*)

Ici , suivant le raisonnement précédent, il existent les chemins : ***1a1, 1a2, 1b2, 1c3, 2b2 et 2c3.***

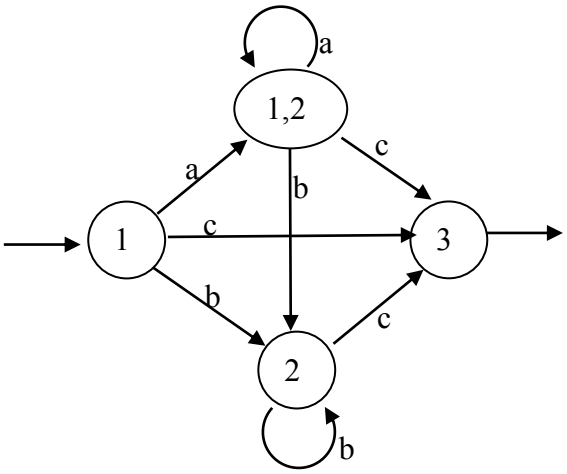
b) Donc cet automate est équivalent à l'automate non déterministe suivant :



(**)

c) Maintenant on peut le déterminiser :

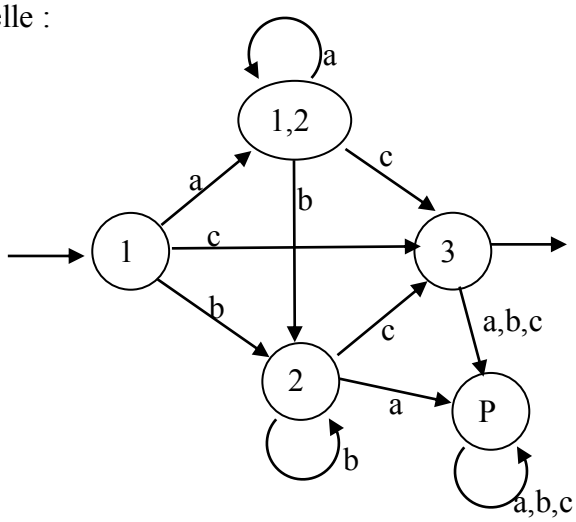
	a	**b**	**c**
1	1,2	2	3
1,2	1,2	2	3
2	-	2	3
3	-	-	-



(***)

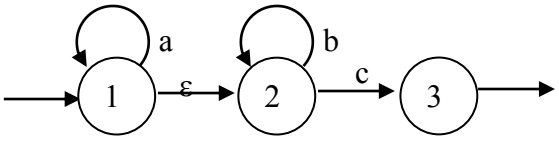
ou bien, en ajoutant l'état poubelle : (****)

	a	**b**	**c**
1	1,2	2	3
1,2	1,2	2	3
2	P	2	3
3	P	P	P
P	P	P	P



d) *Mais en réalité, il est plus simple se passer de l'étape (b) et construire un automate déterministe directement à partir d'un automate asynchrone.*

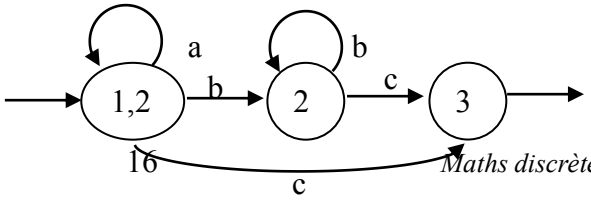
Redessinons l'automate (*) :



(*)

L'état initial de l'automate déterministe correspondant à (*) est (1,2), car en lisant le mot vide on arrive et en 1, et en 2. Continuons à déterminiser en marquant dans les colonnes correspondant à la lecture de chaque caractère, tous les états où on arrive après la lecture du caractère en question, c.à.d. non seulement l'état où mène la flèche étiquetée par ce caractère (disons, ***a***), mais aussi tous les états où on peut arriver en lisant ***aε, aεε*** etc. Dans notre cas particulier, il n'y a pas de telles transitions, mais il faut systématiquement en tenir compte.

état	**a**	**b**	**c**
(1,2)	(1,2)	2	3
2	--	2	3



(*****)

3	--	--	--

On voit que l'automate (*****) n'est pas le même que l'automate (***) ou (****) !
Pourquoi ?
Parce que l'automate déterministe reconnaissant un certain langage n'est pas unique, ce n'est qu'en minimisant qu'on arrive à un automate unique. En minimisant, on doit obtenir la même chose.

Minimisons (****) :

	a	**b**	**c**	
1	1,2	2	3	
1,2	1,2	2	3	Ici , on voit immédiatement que les états 1 et 1,2 ne se
2	P	2	3	sépareront jamais !
3	P	P	P	
P	P	P	P	

Θ_0={(1,(1,2),2,P), (3)}={I, (3)} (Utilisons une notation un peu plus intelligente pour ne pas modifier le sens des chiffres romains sans cesse)

	a	**b**	**c**	
1	I	I	3	
1,2	I	I	3	
2	I	I	3	
P	I	I	I	P se sépare en formant un groupe à part (c'est toujours le cas)

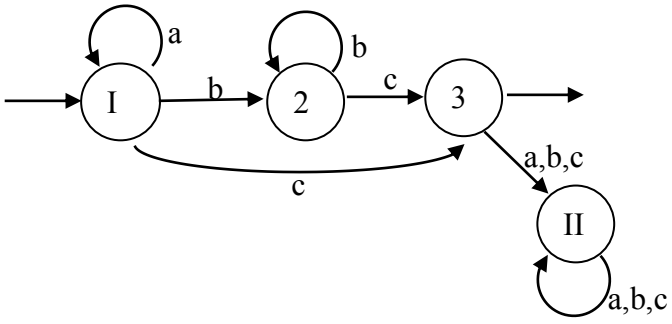
Θ_1={(1,(1,2),2), (P), (3)}={I, (P),(3)}

	a	**b**	**c**	
1	I	I	3	
1,2	I	I	3	
2	P	I	3	2 se sépare

Θ_2={(1,(1,2)), (2), (P), (3)}={I, (2),(P),(3)}

	a	**b**	**c**
1	I	2	3
1,2	I	2	3

Le groupe (1,(1,2)) n'a aucun chance à se scinder en deux, car dès le début 1 et (1,2) ont les mêmes transitions. Donc on a terminé, et on obtient



ce qui est le même automate que le résultat de compléter l'automate (*****) ! (en minimisant l'automate (*****), la seule chose qu'il faut faire c'est de le compléter, autrement il est déjà minimal).