efrei | ÉCOLES D'INGÉNIEUR généralistes du numérique | esigetel

Chapitre 1. Automates finis

INTRODUCTION

L'informatique est une discipline assez récente. Il est difficile de déterminer le moment exact de sa naissance car cela dépend du point de vue ; mais tout le monde est d'accord qu'un des évènements clé c'est l'invention de la **machine de Turing** : la « machine idéale » décrite par Alan Turing en 1936.

Alan Turing (23 juin 1912 -7 juin 1954), un mathématicien, cryptologue et informaticien britannique

Il est l'auteur, en 1936, d'un article de logique mathématique qui est devenu plus tard un texte fondateur de la science informatique. Pour résoudre le problème fondamental de la décidabilité en arithmétique, il y présente une expérience de pensée que l'on nommera ensuite machine de Turing et des concepts de programmation et de programme qui prendront tout leur sens avec la diffusion des ordinateurs, dans la seconde moitié du XXe siècle.

Avec d'autres logiciens (Church, Kleene, etc.), Turing est ainsi à l'origine de la formalisation des concepts d'algorithme et de calculabilité, qui fonderont cette discipline.

Durant la Seconde Guerre mondiale, il joue un rôle majeur dans les recherches sur les cryptographies générées par la machine Enigma, utilisée par les nazis. Ses découvertes permirent, selon plusieurs historiens, de raccourcir la capacité de résistance du régime nazi de deux ans.

Après la guerre, il travaille sur un des tout premiers ordinateurs, puis contribue au débat sur la capacité des machines à penser, en établissant le test de Turing.

En 1952, un fait divers lié à son homosexualité lui vaut des poursuites judiciaires. Pour éviter la prison, il choisit la castration chimique par prise d'œstrogènes.

Suicide ou accident, Turing est retrouvé mort dans la chambre de sa maison à Manchester, par empoisonnement au cyanure, le 7 juin 1954. La reine Élisabeth II le gracie à titre posthume en 2013.

À l'origine, le concept de machine de Turing, inventé avant l'ordinateur, était censé représenter une personne virtuelle exécutant une procédure bien définie, en changeant le contenu des cases d'un tableau infini, en choisissant ce contenu parmi un ensemble fini de symboles.

D'autre part, la personne doit mémoriser un état particulier parmi un ensemble fini d'états. La procédure est formulée en termes d'étapes très simples, du type : « si vous êtes dans l'état 42 et que le symbole contenu sur la case que vous regardez est '0', alors remplacer ce symbole par un '1', passer dans l'état 17, et regarder une case adjacente (droite ou gauche) ».

L’affirmation que tout problème de calcul fondé sur une procédure algorithmique peut être résolu par une machine de Turing s’appelle la thèse Church-Turing.

Exemple

La machine de Turing qui suit possède un alphabet {‘0’, ‘1’}, ‘0’ étant le « blanc » (case non remplie). On suppose que le ruban contient une série de ‘1’, et que la tête de lecture/écriture se trouve initialement au-dessus du ‘1’ le plus à gauche. Cette machine a pour effet de doubler le nombre de ‘1’, en intercalant un ‘0’ entre les deux séries. Par exemple, « 111 » devient « 1110111 ».

L’ensemble d’états possibles de la machine est {e1, e2, e3, e4, e5} et l’état initial est e1.
La table d’actions est la suivante :

Exemple de table de transition				
Ancien état	Symbole lu	Symbole écrit	Mouvement	Nouvel état
e1	0	(Arrêt)		
	1	0	Droite	e2
e2	1	1	Droite	e2
	0	0	Droite	e3
e3	1	1	Droite	e3
	0	1	Gauche	e4
e4	1	1	Gauche	e4
	0	0	Gauche	e5
e5	1	1	Gauche	e5
	0	1	Droite	e1

Le comportement de cette machine peut être décrit comme une boucle :

- Elle démarre son exécution dans l'état e1, remplace le premier 1 par un 0.
- Puis elle utilise l'état e2 pour se déplacer vers la droite, en sautant les 1 (un seul dans cet exemple) jusqu'à rencontrer un 0 (ou un blanc), et passer dans l'état e3.
- L'état e3 est alors utilisé pour sauter la séquence suivante de 1 (initialement aucun) et remplacer le premier 0 rencontré par un 1.
- L'état e4 permet de revenir vers la gauche jusqu'à trouver un 0, et passer dans l'état e5.
- L'état e5 permet ensuite à nouveau de se déplacer vers la gauche jusqu'à trouver un 0, écrit au départ par l'état e1.
- La machine remplace alors ce 0 par un 1, se déplace d'une case vers la droite et passe à nouveau dans l'état e1 pour une nouvelle itération de la boucle.

Ce processus se répète jusqu'à ce que e1 tombe sur un 0 (c'est le 0 du milieu entre les deux séquences de 1) ; à ce moment, la machine s'arrête.

Automates finis

Un modèle plus simple et donc plus restreint mais fort utile est celui d'un **automate fini**.

Un automate fini est une machine abstraite constituée d'**états** et de **transitions**.

Cette machine est destinée à traiter des **mots** fournis en entrée : l'automate passe d'état en état, suivant les transitions, à la lecture de chaque lettre de l'entrée.

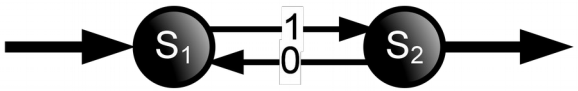
L'automate est dit « fini » car il possède un nombre fini d'états distincts : il ne dispose donc que d'une mémoire bornée, indépendamment de la taille de la donnée sur laquelle on effectue les calculs.

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

On les retrouve dans la **modélisation de processus**, le **contrôle**, les **protocoles de communication**, la vérification de programmes, la **théorie de la calculabilité**, dans **l'étude des langages formels** et en **compilation**. Ils sont utilisés dans **la recherche des motifs dans un texte.**

Un automate est constitué d'états et de transitions. Son comportement est dirigé par un **mot** fourni en entrée : l'automate passe d'état en état, suivant les transitions, à la lecture de chaque lettre de l'entrée.

Voici un exemple très simple d'un automate fini :



Dans l'exemple ci-dessus, pour l'entrée 1010101, et si l'automate démarre en s_1,

il passe successivement par les états $s_1 s_2 s_1 s_2 s_1 s_2 s_1\ s_2$, le calcul correspondant est

$$s_1 \xrightarrow{1} s_2 \xrightarrow{0} s_1 \xrightarrow{1} s_2 \xrightarrow{0} s_1 \xrightarrow{1} s_2 \xrightarrow{0} s_1 \xrightarrow{1} s_2$$

Le mot arrivant à un état terminal (état accepteur), il est accepté (lu, reconnu) par l'automate. Le mot 101010 n'arrivera pas à un état terminal, il ne sera donc pas reconnu. La lecture du mot 01 s'arrêtera au premier caractère, car il n'y a pas de transition il ne sera pas reconnu non plus.

Un tel automate est dit « **fini** » car il possède un nombre fini d'états : il ne dispose donc que d'une mémoire bornée.

On peut très bien considérer des automates sans limitation sur le nombre d'états: la théorie qui en résulte est très analogue à la théorie habituelle.

L'état initial est marqué par une flèche entrante; un état final est, selon les auteurs, soit marqué d'une flèche sortante, comme sur la figure ci-dessus, soit doublement cerclé.

Une autre façon commode de représenter un automate fini est sa *table de transition*.

Elle donne, pour chaque état et chaque lettre, l'état d'arrivée de la transition. Voici la table de transition de l'automate donné en exemple :

		0	1
Entrée	S_1	--	S_2
Sortie	S_2	S_1	--

Il existe plusieurs types d'automates finis.

Dans notre cours, nous n'allons étudier que les « **accepteurs** » (ou « **reconnaisseurs** »), qui produisent en sortie une réponse « oui » ou « non » selon qu'ils acceptent (oui) ou rejettent (non) le mot présenté en entrée.

Dans l'exemple ci-dessus, le mot 1010 n'est pas accepté, mais le mot 10101 est accepté. D'autres automates (**systèmes de reconnaissance)** classent l'entrée par catégorie: au lieu de répondre par oui ou par non, ils répondent par une classification; de tels automates se rencontrent par exemple en linguistique.

Les **capteurs** produisent un certain résultat en fonction de l'entrée.

Les **automates pondérés** associent à chaque mot une valeur numérique.

Quelques définitions

Pour définir un automate fini, on a besoin des éléments suivants :

- **Un alphabet *A*** qui est un ensemble fini d'objets qu'on appelle « lettres » ou « caractères » mais qui peuvent, selon le cas, être de n'importe quel genre (instructions machine, état civil d'une personne, type d'objet géométrique etc…)

- Des **mots** sont des séquences ordonnées de caractères de l'alphabet. Dans nos applications, on aura affaire qu'à des mots de longueur finie, mais il n'est pas interdit d'utiliser des mots de longueur infinie. La longueur d'un mot est le nombre de lettres le constituant.

Exemple : ***w = abacda*** – mot de longueur 6 sur l'alphabet latin (ou bien sur l'alphabet {*a*,*b*,*c*,*d*}).

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

- ***On note A**** ***l'ensemble*** *de tous les mots sur l'alphabet* **A, *y compris le mot vide.***

Un automate fini sur l'alphabet *A* est donné par un quadruplet (***Q***, ***I***, ***T***, ***E***) où

- ***Q*** est un ensemble fini d'états de l'automate,

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

- ***T*** $\subset$ ***Q*** est un ensemble d'états terminaux,

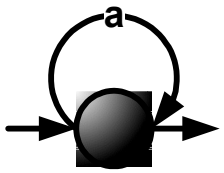
- ***E*** $\subset Q \times (A \cup \{\varepsilon\}) \times Q$ est un ensemble de triplets ***(p.a.q)*** appelés les flèches ou les transitions de l'automate,

Remarque

Nous allons temporairement oublier la notion du mot vide, jusqu'à l'introduction explicite d'automates asynchrones.

Dans tous les exemples que nous allons traiter jusque ce point-là, on pourra donc définir l'ensemble ***E*** comme faisant partie de $Q \times A \times Q$.

Il n'empêche que le mot vide pourra faire partie du langage reconnu par l'automate (par exemple, l'automate suivant reconnaît les mots ε, *a*, *aa*, *aaa* etc. :



Comparaison avec la machine de Turing :

Une machine de Turing possède une mémoire potentiellement infinie : le ruban. Il y a aussi un élément fini de la mémoire : l'ensemble d'états et de transitions, ainsi que l'alphabet. A l'arrêt, le contenu de la mémoire ruban est modifié.

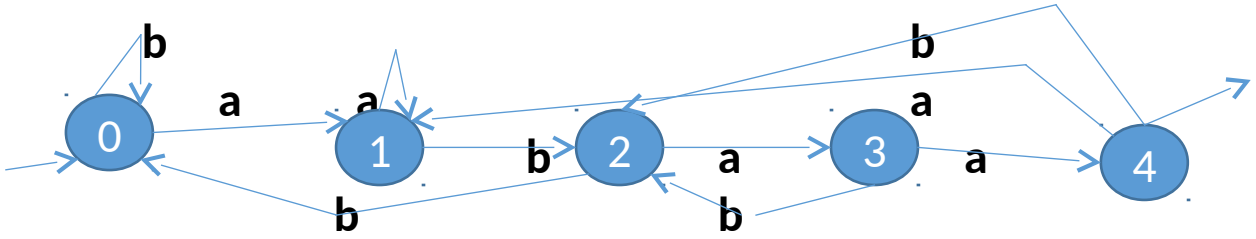
Un automate fini possède une mémoire finie : l'ensemble d'états et de transitions. Les mots fournis à l'automate, à la différence du texte sur le ruban, ne font pas partie de la mémoire, car ils ne peuvent pas être modifiés. A la fin du processus, le mot est reconnu ou non reconnu, mais non pas modifié.

L'interprétation intuitive :

A chaque instant l'automate se trouve dans l'un des états ***p*** de l'ensemble d'états ***Q***.

Il lit alors l'une des lettres ***a*** de l'alphabet ***A*** (la lettre suivant du mot qui vient à l'entrée) et passe dans un état ***q*** tel que ***(p.a.q)*** soit une flèche.

Exemple : La reconnaissance des mots qui se terminent par abaa:



***A* = {*a, b*} ; *Q* = {0, 1, 2, 3, 4} ; *I* = {0} ; *T* = {4}**

La lecture du mot ***w*** se termine dans l'état 4 ssi ***w*** se termine par ***abaa***.

Le même automate peut être représenté par une **table de transitions** :

	Etat	état résultant	
		en lisant a	en lisant b
Entrée	0	1	0
	1	1	2
	2	3	0
	3	4	2
Sortie	4	1	2

Remarque

Cet automate 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***.

On peut montrer que cet automate réalise de façon optimale l'algorithme de recherche de la séquence ***abaa*** dans un texte : il résout pour ce mot l'algorithme du « string matching » utilisé par exemple dans les éditeurs de textes.

La lecture :

L'automate prend un mot (constitué de symboles de son alphabet) en entrée et démarre dans son ou ses état(s) initial(aux).

Il parcourt ensuite le mot de gauche à droite: si l'automate se trouve dans un état ***p*** et le symbole à lire est ***a,*** alors s'il existe(nt) un ou des état(s) q_i tels que la transition ($p.a.q_i$) existe, il passe aux états q_i, et de suite.

(S'il y a plus d'un état q_i, on considère chacun d'eux séparément).

La lecture se termine soit si la lettre suivante ne peut pas être lue (il n'y a pas de transition y correspondant), soit si on a atteint la fin du mot.

Si, à la fin de la lecture, l'automate est dans un état final (accepteur), on dit qu'il accepte l'entrée ou qu'il la reconnaît. Autrement, on dit qu'il la rejette.

Notation :

Pour un mot $w \in A^*$ on note $p \xrightarrow{w} q$ 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 $p \xrightarrow{a} q$ ssi il existe une flèche $(p.a.q) \in E$.

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 $r \xrightarrow{v} q$.

Un mot $w \in A^*$ est donc reconnu par l'automate fini s'il existe un chemin $i \xrightarrow{w} t$ où $i \in I$ et $t \in T$ – c.à.d. qu'en partant d'un état initial et en lisant le mot w, on atteint un état terminal à la fin de la lecture.

Définition :

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

Exercice :
Construire un automate fini reconnaissant les entiers écrits en base 2 divisibles par 3.

Solution :

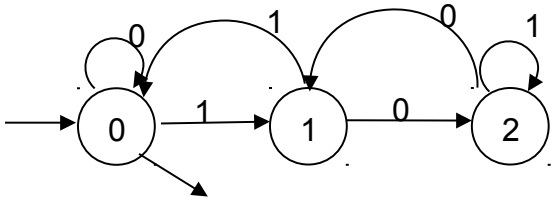
Ajout d'un 0 à la fin d'un nombre binaire le multiplie par 2.
Ajout d'un 1 à la fin d'un nombre binaire le multiplie par 2 et lui ajoute 1.
Marquant les états par le reste de la division entière par 3 :

N	N mod 3	ajout d'un 0 à la fin	reste	ajout d'un 1 à la fin	reste
3n	0	6n	0	$6n+1=3\times 2n+1$	1
3n + 1	1	$6n+2=3\times 2n+2$	2	$6n+3=3\times(2n+1)$	0
3n +2	2	$6n+4=3\times(2n+1)+1$	1	$6n+5=3\times(2n+1)+2$	2

Notons les états par le reste de la division entière par 3. On obtient

Etat	**0**	**1**
0	0	1
1	2	0
2	1	2

L'entrée se fait en 0, et la sortie se fait également en 0 :



Standardisation des automates

Définition :
L'automate $C=(Q, I, T, E)$ sur le langage A est standard si $I=\{i\}$ et, pour tout $a \in A$ et tout $q \in Q$, $(q.a.i) \notin E$ (cela peut se formuler en mots ainsi : l'automate fini est standard s'il possède un unique état initial et si aucune transition n'aboutit dans cet état).

Propriété importante :
Un automate standard possédant plus d'un état et dont l'état initial n'est pas terminal, ne reconnaît pas le mot vide. Cette propriété est évidente : pour arriver de l'état initial à un état terminal, il faut au moins passer par une des transitions partant de l'état initial, donc lire au moins un caractère.

Algorithme de standardisation

On peut rendre tout automate fini non standard ***C*** standard. En ce faisant, on ne modifie pas le langage qu'il reconnaît.

Pour standardiser, on ajoute à l'automate ***C*** un nouvel état qui devient l'unique état initial et duquel, pour chacune des transitions qui partaient des anciens états initiaux, part une transition de même étiquette vers les mêmes états.

Formellement, on construit un nouvel automate $C' = (Q \cup \{i\}, \{i\}, T', E')$ (le **standardisé** du C) avec :

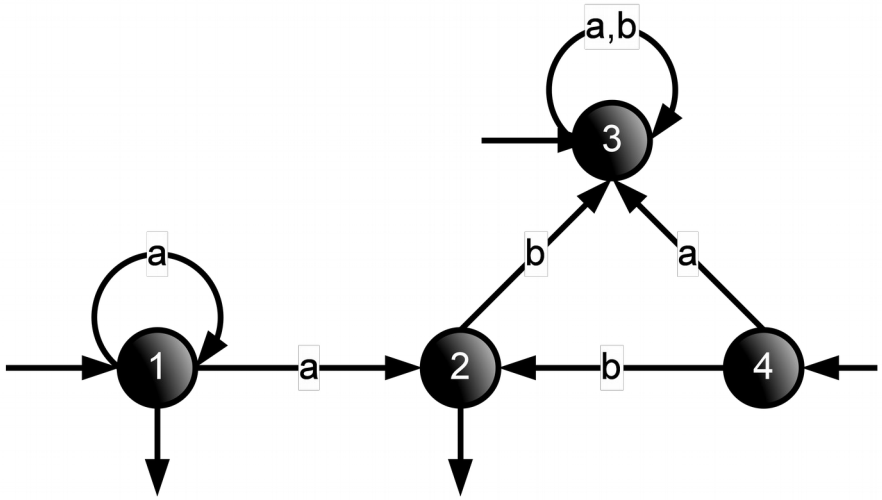
$\{i\} \notin Q$;

$$T' = \begin{cases} T \text{ si } I \cap T = \emptyset \\ T \cup \{i\} \text{ si } I \cap T \neq \emptyset \end{cases}$$

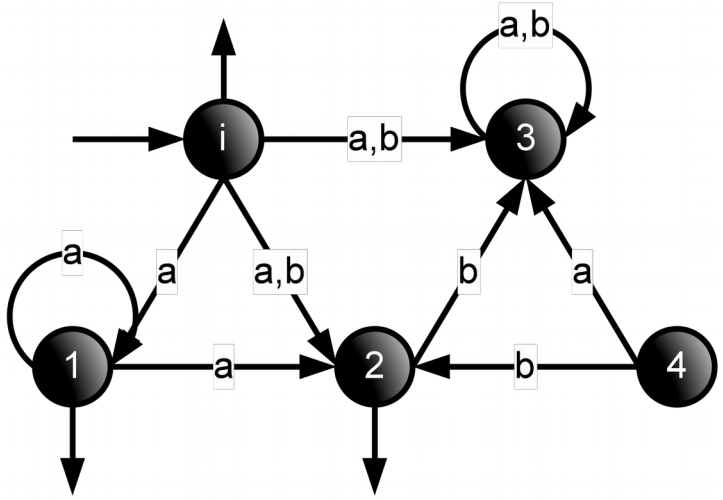
$E' = E \cup \{(i.a.q) \mid \exists q_i \in I \text{ tel que } (q_i.a.q) \in E\}$

Exemple :

L'automate suivant n'est pas standard : il possède trois états initiaux et des transitions aboutissant dans chacun d'entre eux.



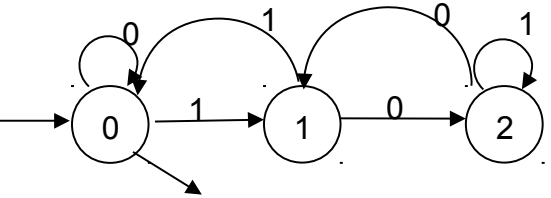
	état	a	b
E,S	1	1,2	--
S	2	--	3
E	3	3	3
E	4	3	2

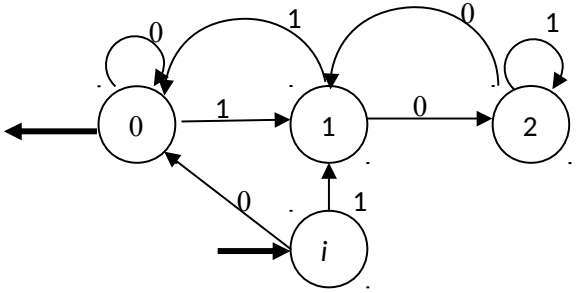


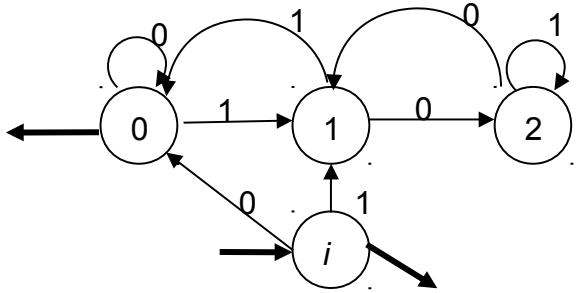
La seule façon de laquelle l'automate standardisé peut reconnaître le mot vide, c'est si l'état ***i*** est un état de sortie.

Donc on obtient un moyen très simple de modifier un automate de telle façon qu'il cesse de reconnaître le mot vide s'il le fait : il faut le standardiser et enlever la sortie sur l'état initial.

Appliquons cette recette à l'automate



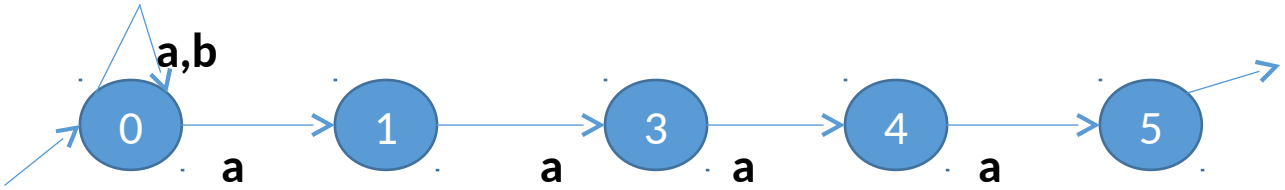




Automates déterministes

On distingue des automates finis déterministes et non déterministes. L'automate que nous avons montré précédemment est déterministe.

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



(Cet automate reconnaît tous les mots qui se terminent par ***abaa***. Il est très facile à construire.)

Le fait qu'il n'est pas déterministe se manifeste en ce que, en lisant ***a*** à partir de l'état 0, on peut aller et à l'état 0, et à l'état 1, ce qui ne serait pas possible pour un automate déterministe.

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

Automate déterministe, définition

L'automate $C = (Q, I, T, E)$ est déterministe ssi pour $\forall p \in Q$ et $\forall a \in A$ $\exists$ au plus un état $q \in Q$ tq $(p.a.q)$, et qu'il y ait un seul état initial : $I = \{ i \}$.

On peut donc caractériser un automate déterministe C par un quadruplet (Q, i, T, E) où i est l'état initial.

Si la transition $(p.a.q)$ existe, on notera **$p.a=q$.** Si elle n'existe pas, on conviendra que **$p.a$** n'a pas de valeur.

On peut facilement vérifier par récurrence sur la longueur des mots que pour un automate déterministe **C,**

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

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

Déterminisation

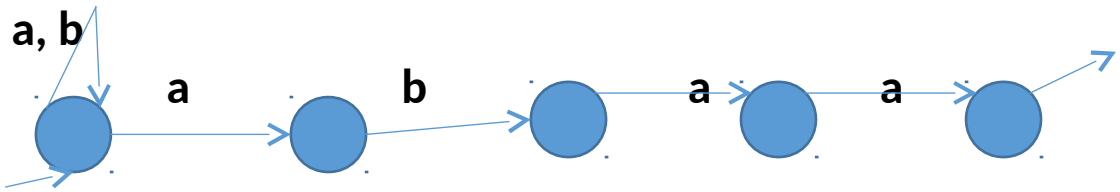
Soit ***C = (Q, I, T, E)*** un automate fini.

Notons Π **l'ensemble des parties de *Q***. C'est un ensemble fini puisque si ***Q*** a ***n*** éléments, Π en a 2^n**.**

Un automate déterministe *D* correspondant à ***C*** prend

- **comme ensemble d'états un sous-ensemble *P*** de l'ensemble Π des parties de ***Q***.
- **comme unique état initial l'ensemble *I*** des états initiaux de ***C***
- **comme ensemble d'états terminaux l'ensemble $\zeta=\{u \in P \mid u \cap T \neq \varnothing\}$** des parties de ***Q*** qui contiennent au moins un état terminal de ***C***.
- **comme flèches l'ensemble des *(u.a.v)*** ou $u \in P$ et ***v*** est l'ensemble de tous les états $q \in Q$ tels qu'il existe un état ***p*** dans ***u*** avec $(p.a.q) \in E$.

Cet algorithme formel s'explique le plus facilement sur un exemple :



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 regroupés 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 qu'on appelle (0,1). En ***b***, on va vers l'état (0) de l'automate initial.

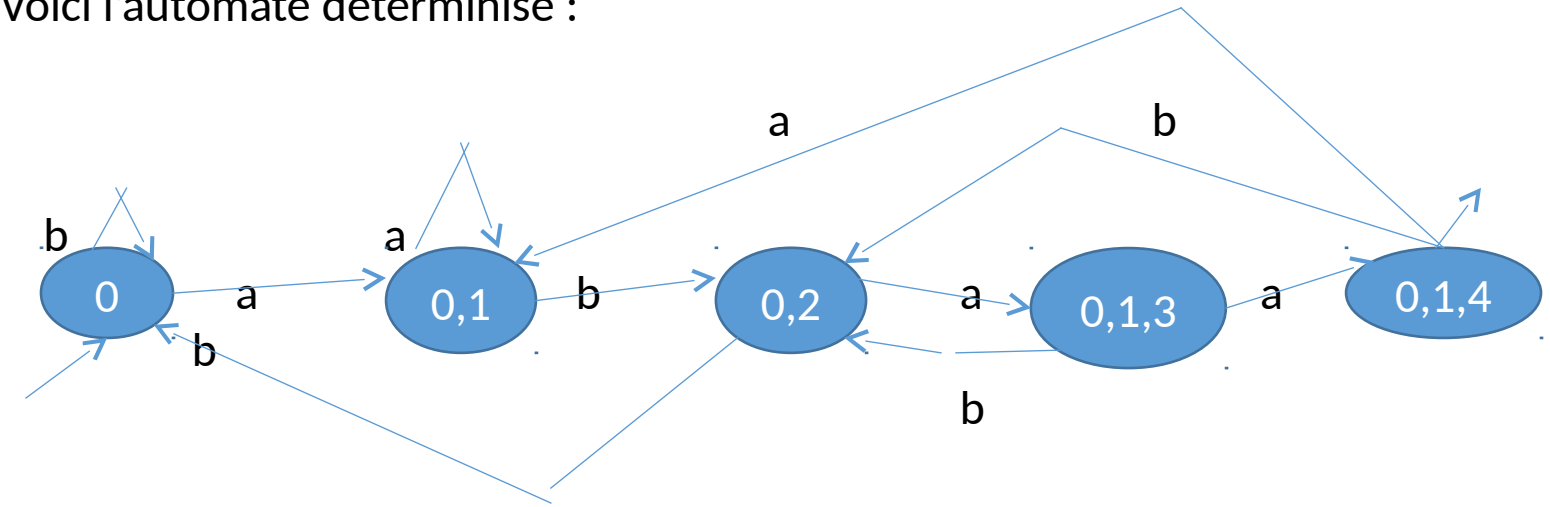
Donc, (0,1) est lui aussi un état de l'automate déterminisé.

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 ***C*** reconnaissent le même langage est fait par récurrence.

<u>Automate déterministe complet</u> (définition) :

Pour $\forall u \in P$ et $\forall a \in A \; \exists v \in P$ tel que $u.a=v$.

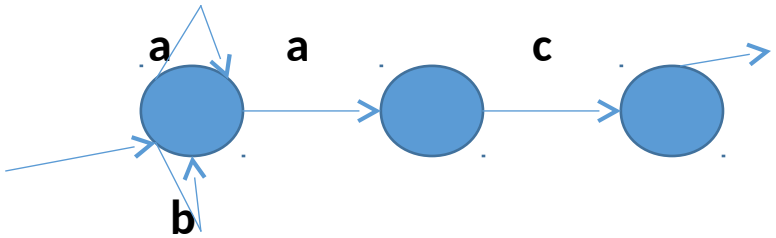
C'est-à-dire que de chaque état sortent des flèches avec toutes les étiquettes possibles.

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 s'il n'est pas déjà complet.

<u>Exemple :</u>

-



Déterminisation :

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

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

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

Remarque

Il n’y a pas trop de sens de parler d’un automate non déterministe complet ou pas complet.

De même, même si introduire un « état poubelle » dans un automate non déterministe est possible et ne nuit pas à son fonctionnement, ceci n’est pas utile pour la déterminisation de cet automate.

Normalement, on n’introduit un état poubelle si besoin est qu’une fois l’automate est devenu déterministe.

Encore quelques définitions :

- Un automate est ***accessible*** si n'importe quel état est accessible à partir de l'état initial.

- Un automate est ***coaccessible*** si à partir de n'importe quel état on peut accéder à un état terminal.

- Accessible + coaccessible = ***émondé***.

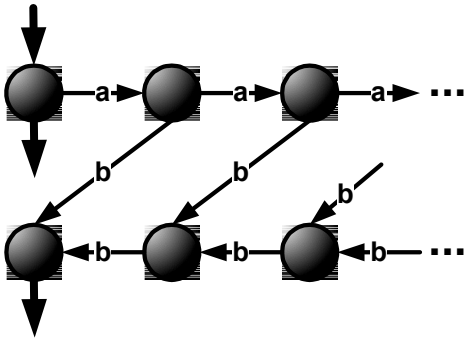
- Un ensemble de mots ***X*** est un langage ***reconnaissable*** ssi il existe un automate fini ***D*** qui le reconnaît.

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 toujours équivalent à un automate déterministe et complet ***F***.

Ecrivant ***F=(Q, i, T, E)***, on a alors $w \in X$ ssi $i.w \in T$ et donc $w \notin X$ ssi $i.w \notin T$.

Remarque

Un langage non reconnaissable peut parfois être reconnu par un automate ***non fini***. Exemple : le langage consistant en tous les mots du type $a^n b^n$ (les mots où il y a autant de *a* que de *b* (sans fixer le nombre) et où tous les *a* précèdent à tous les *b*) n'est pas reconnaissable. Or, il est évidemment reconnu par l'automate infini suivant :

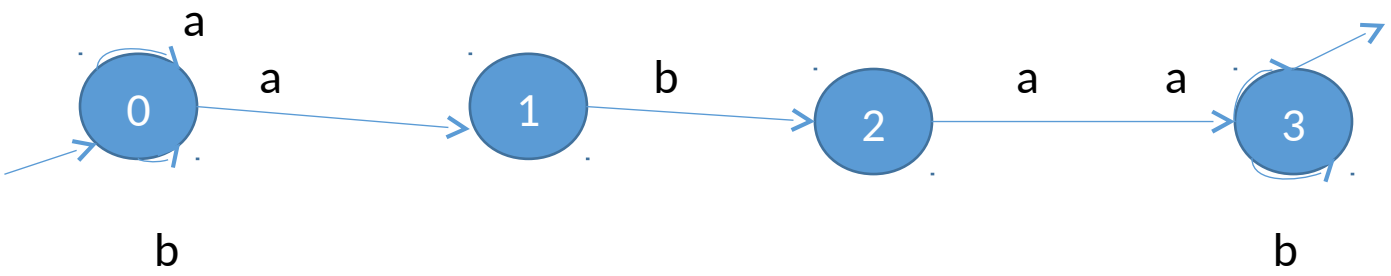


Le complément d'un langage reconnaissable

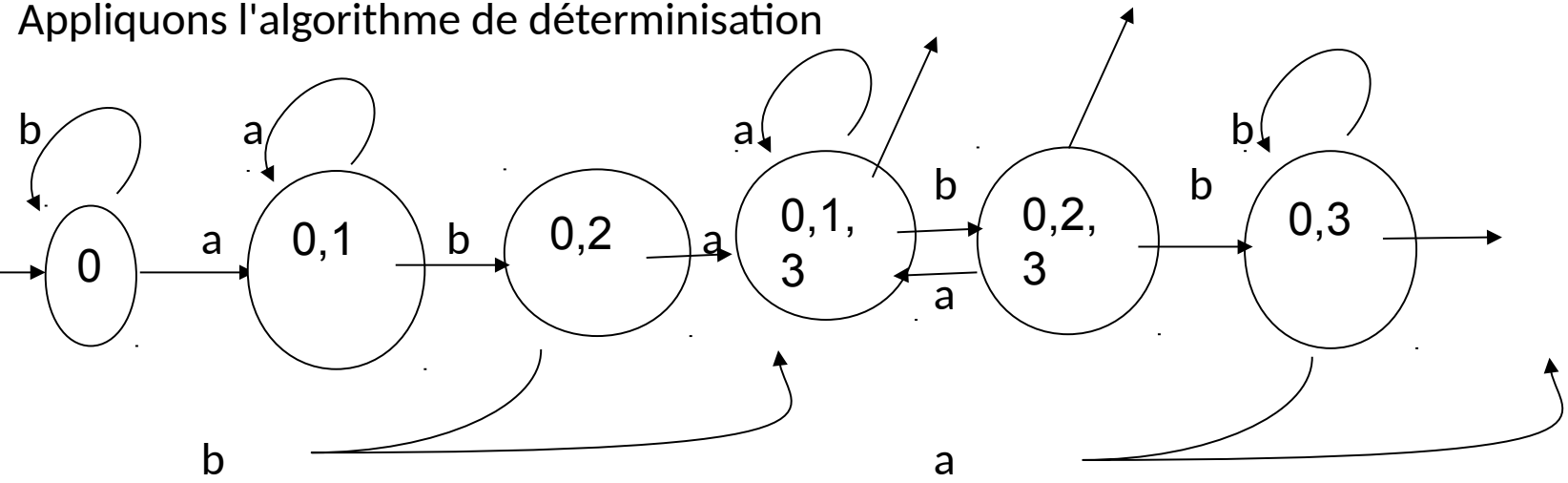
Le complément d'un langage reconnaissable (l'ensemble de mots sur le même alphabet n'appartenant pas au langage en question) est encore reconnaissable.
Soit ***D*** un automate déterministe et complet reconnaissant le langage ***X***.
Pour construire un automate reconnaissant le complément de ***X***, il suffit de prendre comme ensemble d'états terminaux le complément de l'ensemble d'états terminaux de ***D***.

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

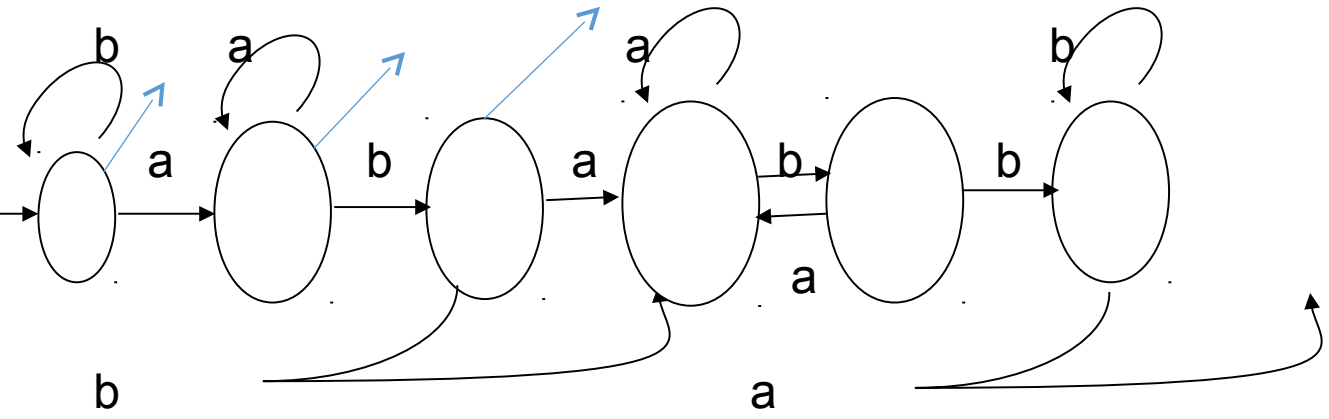
On commence par la construction d'un automate (non déterministe) reconnaissant tous les mots ayant ***aba*** en facteur.



Appliquons l'algorithme de déterminisation



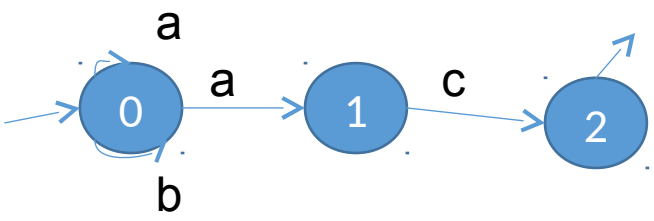
	état	a	b
initial	(0)	(0,1)	(0)
	(0,1)	(0,1)	(0,2)
	(0,2)	(0,1,3)	(0)
terminal	(0,1,3)	(0,1,3)	(0,2,3)
terminal	(0,2,3)	(0,1,3)	(0,3)
terminal	(0,3)	(0,1,3)	(0,3)



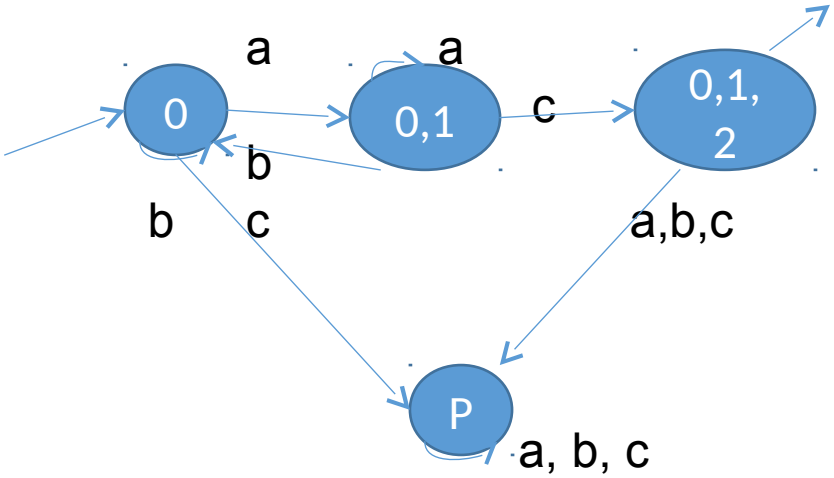
Remarque très importante :

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 terminaux par des états non terminaux et vice versa : il faut d'abord le compléter.** Alors l'état poubelle (qui ne peut pas être terminal) devient un état terminal de l'automate reconnaissant le complément du langage.

Exemple:



Il devient



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.

On sépare tous les états de l'automate déterministe initial en deux groupes : terminaux et non-terminaux.

Puis, on analyse la table des transitions en marquant vers quel groupe va chaque transition.

On repartitionne les groupes selon les patterns des transitions en terme de groupes.

On répète ce processus en utilisant cette nouvelle partition, et on réitère jusqu'à ce qu'on arrive à ne plus pouvoir partitionner.

Les groupes restants forment l'automate minimal.

On appelle souvent les itérations les étapes.

Description du processus de partitionnement itératif

Soit

un automate fini déterministe complet $D=(Q, i, T, E)$

Résultat à obtenir: 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_i, à commencer par Θ_0 :

POUR *chaque groupe* ***G*** de Θ_i ***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_i ***;***

/* 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_{i+1} ***;***

fin

3. Si $\Theta_{i+1} = \Theta_i$ ***alors*** $\Theta_{final} = \Theta_i$ **et continuer à l'étape 4.**

Sinon, répéter l'étape (2) avec Θ_{i+1} ***comme partition courante.***

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'***.

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.

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*** ($e \xrightarrow{a} r$).

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 : $\Theta_0 = \{(0, 1, 2, 3), (4)\}$

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 $\Theta_1 = \{(0, 1, 2\}, (3), (4)\}$.

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 Θ_2= {(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_3 reste identique à la séparation de l'étape précédent : $\Theta_3 = \Theta_2$ ={(0, 2}, (1), (3), (4)}.

Fin de la procédure itérative de séparation.

Voici les itérations qu'on a effectuées :

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

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

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

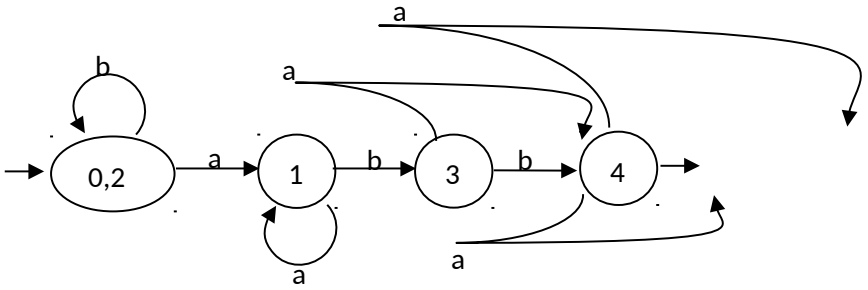
$\Theta_{new} = \{(0, 2\}, (1), (3), (4)\}$

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

$\Theta_{new} = \{(0, 2\}, (1), (3), (4)\} = \boldsymbol{\Theta_{final}}$

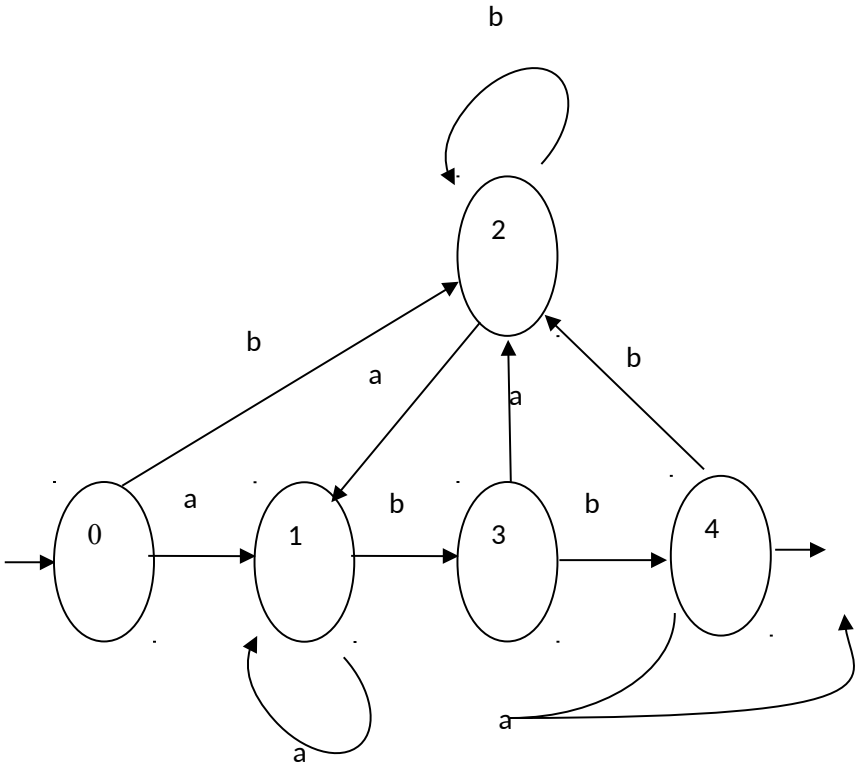
Passons aux représentants :

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).

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 :



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.

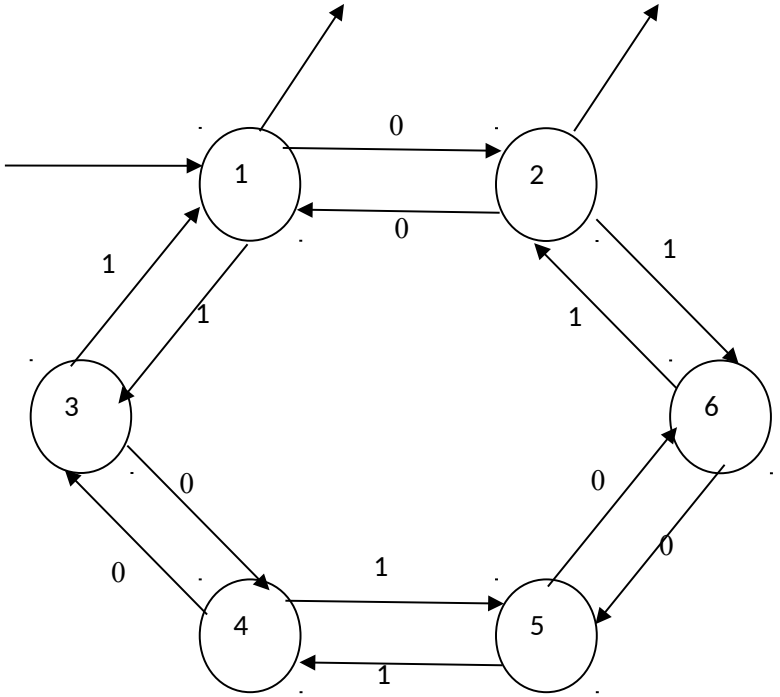
Ici, tous les états peuvent être distingués par une chaîne ou une autre, sauf les états 0 et 2.

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 (ici, il y en a un seul).

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 :