www.momirandum.com

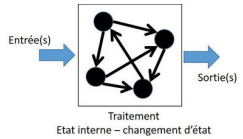
Automates finis et Reconnaissance de mots

Ce document est généré en support au cours "Automates Finis" dispensé pour l'année 2018/2019.
Aucune autre utilisation n'est autorisée.

Méthodes et Outils Mathématiques et Informatiques

Automate fini

Un automate est une machine abstraite permettant de traiter de l'information. L'information en entrée (généralement une suite d'éléments de base) guide le traitement effectué par l'automate. Ce traitement est défini de façon globale en terme d' "états internes" possibles pour l'automate, et de règles de passage d'un état à un autre en fonction des éléments de base en entrée : en fonction de l' "état courant" et de l'élément reçu en entrée, la machine passe dans un autre état (nommé ci-dessous "état atteint"). Ces règles de passage d'un état à un autre sont appelées "transitions".
Au cours et/ou à l'issue de cette suite de transitions, qui dépend donc de l'information en entrée, des traitements peuvent être effectués et un ou des résultats sont fournis par l'automate à son utlisateur.



Exemple : un téléphone en marche (version simplifiée ici, bien entendu...).
Dans une description générale, en dehors de toute conversation, un téléphone est considéré dans un état "en attente". Deux événements peuvent alors être transmis à l'automate : l'émission d'un appel "sortant", ou la réception d'un appel "entrant". Dans le second cas, le téléphone se met "en attente sur appel entrant", et émet une sonnerie. Lorsque l'utilisateur décroche, le téléphone de met dans l'état "en conversation". Que ce soit à la suite de cette série de transitions, ou de celles pouvant correspondre à l'émission d'un appel sortant, le téléphone est alors dans le même état. Reste à attendre un événement signalant la fin de la communication (événement local ou distant) pour que le téléphone retrouve son état "en attente".

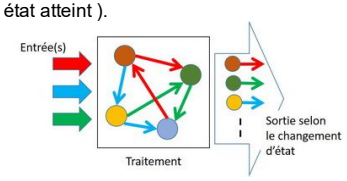
Supposons donc pour l'instant un automate défini par :
- un ensemble d'états, que l'on notera Q ;
- un ensemble d'éléments pouvant être reçus en entrée, que l'on notera A ;
- un ensemble de transitions, que l'on notera E.
L'ensemble E est donc un sous-ensemble de QxAxQ et contient des triplets (état courant , élément reçu , état atteint).

REMARQUE : Nous ne considérons dans ce cours que le cas d'automates pour lesquels tant l'ensemble des éléments pouvant être reçus ('A', l' "alphabet") que l'ensemble des états internes ('Q') sont des ensembles finis.

Les résultats du traitement de l'information reçue par l'automate peuvent être de nature différente :

- *Résultats en fonction du changement d'état de l'automate*

Chaque résultat dépend à la fois de l'état courant de l'automate lors de la réception d'un élément en entrée, et de l'entrée elle-même, et de l'état atteint. Il est donc nécessaire de définir un traitement particulier pour chaque triplet (état courant , élément en entrée , état atteint).

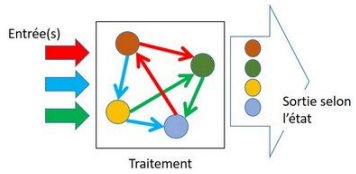


Bien entendu, à plusieurs triplets différents on peut associer le même traitement. De même un traitement peut être vide, c'est-à-dire que dans la situation associée, aucune action n'est effectuée.
On peut déjà avoir en mémoire le problème lié à une ambiguïté que peut contenir la définition de l'automate. Si deux états q_1 et q_2 peuvent être atteints depuis un même état courant p sur réception d'un même élément x en entrée, quel doit être le prochain état courant (i.e. que l état atteint doit-on choisir) ? Quelle action effectuer ? D'un point de vue opérationnel, est-ce une situation légitime, ou y a-t-il une erreur dans la définition de l'automate ?

Variante : on peut imaginer devoir créer un automate dont le traitement ne dépend pas des éléments reçus, mais uniquement des passages d'un état à un autre. On associe donc un traitement à chaque couple (état courant , état atteint).

- *Résultats en fonction de l'état courant de l'automate*

Le résultat dépend uniquement de l'état atteint après réception d'un élément en entrée, quel que soit l'état courant avant réception de cette entrée et l'élément en entrée.

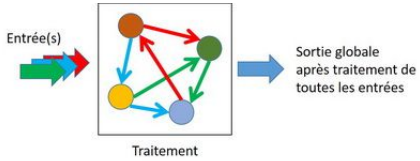


Il suffit pour cela d'associer une action à chaque état : celle-ci est effectuée par l'automate à chaque fois que celui-ci devient l'état courant.

Variante : on n'effectue pas un traitement à chaque réception d'une entrée, en fonction de l'état atteint, mais on effectue périodiquement une action, en fonction de l'état à cet instant (certains éléments en entrée n'ont donc pas d'effet direct en terme d'action ou de résultat délivré par la machine, mais seul l'état atteint après une suite d'entrées sera pertinent).

- *Résultat global après le traitement complet de l'ensemble de l'information fournie en entrée*

Un ensemble complet d'éléments en entrée est traité en bloc. Seul l'état atteint à la fin du traitement de toutes ces entrées compte.



On peut tout à fait envisager des traitements distincts selon l'état atteint en fin d'analyse de tous les éléments en entrée.

C'est dans ce cadre que nous nous intéressons ici à un type particulier de ces automates, dits "**automate accepteur**" : le résultat reflète la "conformité" ou "non conformité" de la suite des éléments en entrée, selon les règles établies par la définition de l'automate lui-même.
Plus spécifiquement, nous nous intéressons aux "**automates finis**" qui sont une catégorie de ces "automates accepteurs".

NOTIONS DE BASE - AUTOMATE, ALPHABET, MOT, LANGAGE

Un automate fini est une machine abstraite qui permet d'effectuer un traitement particulier en fonction d'une suite d'éléments qui lui sont fournis en entrée.
Ces éléments d'entrée, pris l'un distinctement des autres, sont les éléments de base d'un langage et en constituent l'alphabet. L'automate va donc prendre en entrée une suite d'éléments de cet alphabet.
Au lancement du traitement de l'automate, l'état courant de celui-ci est positionné sur un état particulier appelé "état initial". Au fur et à mesure de l'analyse de ces éléments d'entrée, le système modélisé par l'automate va changer d'état (l'état courant va changer). En fin de traitement des éléments en entrée, on déterminera si l'état courant atteint est un état dit "terminal" ou non.

Alphabet - Mot

L'**alphabet** est l'ensemble des **symboles** utilisés en entrée de l'automate. Une suite de ces symboles forme un **mot**.

Par exemple, l'ensemble des lettres 'a', 'b', ..., 'A', ... ainsi que de certains symboles particuliers tels que le tiret '-', les lettres accentuées 'é', ... permet de former les mots de la langue française.
Ou bien, l'ensemble des chiffres, plus la virgule décimale et les opérateurs '+', '-', '/' et 'x' permettent d'écrire des expressions arithmétiques.

On note habituellement $\mathcal{A}$ l'alphabet.
Dans nos exemples précédents, on a
$\mathcal{A}$ = { 'a' , 'b' , ... , 'A' , ... , '-' , 'é' , ... , 'ç' , ... },
$\mathcal{A}$ = { '0' , '1' , ... , '9' , ',' , '+' , '-' , 'x' , '/' }.

L'alphabet n'est bien sûr pas limité à un ensemble de "caractères" tels que ceux pris dans les exemples précédent. Un alphabet est un ensemble d'éléments "de base" permettant de construire un langage. Ainsi, si l'on veut modéliser une machine sous forme d'un automate, on pourra par exemple utiliser les éléments suivants dans la définition de l'alphabet :
$\mathcal{A}$ = { 'marche', 'arrêt', 'envoi message', 'envoi message avec accusé de réception', 'réception message', ... }

L'ensemble $\mathcal{A}^+$ dénote l'ensemble de tous les mots non vides (i.e. contenant au moins 1 symbole de l'alphabet) qu'il est possible de former en combinant les symboles contenus dans $\mathcal{A}$.
Si u et v sont deux éléments de $\mathcal{A}^+$, alors la concaténation des deux mots, notée uv, est également un élément de $\mathcal{A}^+$.

On note **ε** le **mot vide** (i.e. ne contenant aucun symbole).
On utilise également la notation ε pour désigner explicitement une absence de symbole (comme étiquette d'une transition ou dans une expression rationnelle : voir les chapitres correspondant).
On a alors, pour tout mot u de $\mathcal{A}^+$, uε=εu=u.
"uε" signifie "u suivi de rien"
"εu" signifie "rien suivi de u".
On a également, pour tous mots u et v de $\mathcal{A}^+$, uεv=uv.
"uεv" signifie "u, puis rien, puis v".

On note $\mathcal{A}^*$ = $\mathcal{A}^+ \cup \{ \varepsilon \}$.
C'est à dire $\mathcal{A}^*$ que dénote tous les mots pouvant être écrits en utilisant les symbole de l'alphabet $\mathcal{A}$, y compris le mot vide (ne contenant donc aucun symbole).

Langage

Soit $\mathcal{A}$ un alphabet.

On appelle **langage sur $\mathcal{A}$** tout sous-ensemble de $\mathcal{A}^*$.

Un langage est donc une partie des mots pouvant être formés à partir des symboles de l'alphabet.
Un langage peut ou non contenir le mot vide.

On parle plus loin de langage reconnu par un automate, c'est-à-dire de l'ensemble des mots qui sont "acceptés" par l'automate.

Remarque :
On a indiqué en début de ce cours que l'on ne s'intéresse qu'aux cas où l'alphabet est un ensemble fini de symboles. Par contre, un langage défini sur un tel alphabet peut être infini.

Exemple simple :

Prenons l'alphabet constitué de deux symboles : { x , y }.
Si on considère le langage contenant tous les mots contenant un et un seul symbole 'x'. Les mots de ce langage sont donc de la forme :
- un nombre quelconque (éventuellement 0) de 'y' ;
- puis un symbole 'x' ;
- puis à nouveau un nombre quelconque (éventuellement 0) de symboles 'y'.

Ce langage contient donc de toute évidence un nombre infini de mots.

DEFINITIONS ET PROPRIETES DES AUTOMATES

Nous donnons dans cette section les définitions et propriétés principales relatives aux automates "accepteurs" traités dans ce cours. Les traitements associés, par exemple comment rendre un automate "standard" ou "déterministe", sont abordés dans les sections suivantes.

Automate Fini : définition, représentation et langage reconnu

Alphabet - États - Transitions

Un **automate fini** est caractérisé par la définition de cinq éléments :

$\mathcal{A}$: l'**alphabet** utilisé pour la construction de mots
$\mathcal{A}$ est l'ensemble des éléments d'entrée de l'automate

$\mathcal{Q}$: un ensemble fini d' "**états**" de l'automate

$\mathcal{I}$: un sous-ensemble de $\mathcal{Q}$ formant les états dits "**initiaux**", ou états "de départ"
$\mathcal{I} \subseteq \mathcal{Q}$
Ces états initiaux sont les états "courants" possibles de l'automate au début de la reconnaissance d'un mot.

$\mathcal{T}$: un sous-ensemble de $\mathcal{Q}$ formant les états dits "**terminaux**", ou états "finals", ou états "d'acceptation"
$\mathcal{T} \subseteq \mathcal{Q}$
Ces états terminaux sont les états courants possibles de l'automate à la fin de l'analyse d'un mot pour que celui-ci soit dit "reconnu", ou "accepté" par l'automate.

$\mathcal{E}$: un ensemble de triplets appelés "**transitions**"
$\mathcal{E} \subseteq \mathcal{Q} \times \mathcal{A} \times \mathcal{Q}$
Les transitions sont donc des triplet donnant les règles de passage d'un état à un autre sous la forme :
(état courant , symbole de l'alphabet , état cible)
Le fonctionnement classique d'un automate est : si l'état courant est 'p', que le symbole reçu en entrée est 'x', et qu'il existe une transition (p,x,q), alors l'état courant devient l'état 'q' et on passe à l'analyse du symbole suivant.

On notera l'automate ainsi désigné par le quintuplet
Aut = < $\mathcal{A}$, $\mathcal{Q}$, $\mathcal{I}$, $\mathcal{T}$, $\mathcal{E}$ >

Exemple

< { a , b , c , d , e } ,	*alphabet*
{ 1 , 2 , 3 , 4 , 5 , 6 } ,	*états*
{ 1 , 3 , 6 } ,	*états initiaux*
{ 6 } ,	*états terminaux*
{ (1,a,2) , (1,a,4) , (2,a,2) , (2,c,5) , (2,d,5) , (3,b,2) , (3,b,4) , (4,b,4) , (4,c,5) , (4,d,5) , (5,e,6) }	*transitions*
>	

Remarque concernant le mot / symbole 'ε'

Mot vide 'ε'

L'acceptation du mot vide dans un langage se satisfait de la définition précédente d'un automate. Il suffit pour cela qu'il y ait au moins un état initial qui soit également un état terminal.

En effet :
Soit 'p' cet état. Au début de la reconnaissance d'un mot, l'état courant de l'automate peut être positionné à 'p'. Si le mot à analyser est le mot vide, alors il n'y a aucune transition à appliquer, puisque la chaîne de symboles en entrée est déjà finie. L'état courant 'p' étant lui aussi un état terminal, le mot (vide) est donc reconnu.

Symbole vide 'ε' et transitions

La définition précédente d'un automate de mentionne pas le symbole 'ε'. Celui-ci ne peut pas formellement faire partie de l'alphabet, puisque ce n'est pas un symbole.
Il est cependant fort utile de pouvoir utiliser ce symbole dans la construction d'automate afin de le construire plus aisément. L'ensemble des transitions est alors défini de la façon suivante :

$E \subseteq \mathcal{Q} \times (\mathcal{A} \cup \{\varepsilon\}) \times \mathcal{Q}$

Cette facilité offerte par les transitions utilisant le symbole 'ε' peut bien entendu avantageusement être utilisée lors de la construction "intuitive" d'un automate correspondant à la définition d'un langage particulier.
Elle est systématiquement utilisée par les techniques de construction automatiques d'un automate fini à partir d'une expression rationnelle (voir sections associées).

Dans ce cours, lorsque aucune mention explicite n'est faite à ces transitions particulières, on suppose que l'automate n'en contient pas. Ceci est vrai dans les définitions comme dans les algorithmes de manipulation d'automates.

Table de transitions

Soit un automate $< \mathcal{A} , \mathcal{Q} , \mathcal{I} , \mathcal{T} , \mathcal{E} >$.

Une **table de transitions** est un moyen de représenter l'ensemble $\mathcal{E}$ des transitions constituant l'automate.

Il s'agit d'une table à deux dimensions :
- chaque ligne correspond à un état,
- chaque colonne correspond à un symbole de l'alphabet.

Remarque : ces correspondances ligne/état et colonne/symbole ne sont bien sûr que des conventions utilisées ici. Leur utilisation est cependant assez courante.

Bien que $\mathcal{E}$ soit défini précédemment comme un ensemble de triplets, et non pas précisément une matrice, notons quand même $\mathcal{E}[p,x]$ la case de la table de transitions correspondant à l'état p (ligne) et au symbole x (colonne).
$\mathcal{E}[p,x]$ est un ensemble d'états tel que :

$q \in \mathcal{E}[p,x]$ si et seulement si $(p,x,q) \in \mathcal{E}$

i.e. :

$\mathcal{E}[p,x] = \{ q \in \mathcal{Q} \mid (p,x,q) \in \mathcal{E} \}$

Une table de transitions donne explicitement la valeur de l'ensemble $\mathcal{E}$ (transitions), et implicitement celles de l'ensemble $\mathcal{A}$ (symboles = colonnes de la table) et celles de l'ensemble $\mathcal{Q}$ (états = lignes de la table).
Pour qu'un automate soit entièrement spécifié, il faut également indiquer quels sont les états initiaux ($\mathcal{I}$) et les états terminaux ($\mathcal{T}$).
Il existe plusieurs façons de représenter les états initiaux (respectivement terminaux) : '→', 'I'(nitial), 'E'(ntrée), ... (respectivement '←', 'T'(erminal), 'S'(ortie), ...). Nous utiliserons ici les flèches '→' et '←'.

Exemple

Définition

< { a , b , c , d , e } ,
{ 1 , 2 , 3 , 4 , 5 , 6 } ,
{ 6 } ,
{ (1,a,2) , (1,a,4) ,
(2,a,2) , (2,c,5) , (2,d,5) ,
(3,b,2) , (3,b,4) ,
(4,b,4) , (4,c,5) , (4,d,5) ,
(5,e,6) }
>

Représentation sous forme de table des transitions

		a	b	c	d	e
→	1	2,4				
	2	2		5	5	
→	3		2,4			
	4		4	5	5	
	5					6
→ ←	6					

Remarques
- S'il existe plusieurs transitions partant d'un même état et concernant un même symbole, par exemple ci-dessus (1,a,2) et (1,a,4), la case $\mathcal{E}[1,a]$ contient l'ensemble "{2,4}". Nous utilisons cependant une **notation simplifiée** et les états sont plus simplement identifiés sous forme d'une liste "1,2".
- Afin de ne pas charger inutilement une table de transitions, une case vide, par exemple $\mathcal{E}[3,c]$, signifie l'absence de transition ($\mathcal{E}[3,c]=\varnothing$).

Graphes d'états

On peut également représenter "graphiquement", ou schématiquement, un automate par son **graphe d'états**.

Les états sont représentés par des "cercles".
Bien que cela ne soit pas toujours nécessaire pour les explications données dans ce cours, on s'efforcera de donner un nom distinct à chaque état : c'est le libellé qui est inscrit dans ces cercles.
En "théorie des graphes", les états sont des "sommets".

Les transitions sont représentées par des "flèches" reliant deux états. A chacune de ces flèches on associe une étiquette pour identifier un symbole de l'alphabet. Dans l'exemple ci-dessous, la flèche de '3' vers '4' étiquetée 'b' indique que l'automate contient une transition permettant de passer de l'état '3' à l'état '4' su le symbole courant en entrée est un 'b' : transition (3,b,4).
Important : le sens des flèches est essentiel : la transition (3,b,4) est différente d'une transition éventuelle (4,b,3) !
En "théorie des graphes", les transitions sont représentées par des "arcs".

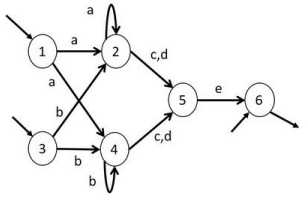
On distingue également au moyen d'un graphisme particulier les états initiaux et/ou terminaux.
Une flèche "entrante" est associée aux états initiaux ; une flèche "sortante" est associée aux états terminaux. Ces flèches ne relient pas deux états, ce ne sont donc pas des transitions.

Exemple

Table des transitions

		a	b	c	d	e
→	1	2,4				
	2	2		5	5	
→	3		2,4			
	4		4	5	5	
	5					6
→ ←	6					

Graphe des états (et des transitions)



Remarques :
- Dans la convention utilisée ci-dessus, les états initiaux et terminaux sont repérés par des flèches sans étiquette, puisque ce ne sont pas des transitions. On trouve parfois d'autres conventions graphiques, comme par exemple des carrés, triangles ou doubles ronds.
- L'automate représenté contient notamment deux transitions de l'état 2 vers l'état 5. On aurait pu représenter deux flèches avec chacune son étiquette. Pour simplifier le graphe des états et faciliter sa lecture, on ne trace qu'une seule flèche et on liste les deux symboles correspondant à chacune des deux transitions. Mais il y a bien 2 transitions allant de 2 à 5. De même de 4 à 5.
- L'automate en exemple contient une transition (4,b,4), c'est à dire que si l'état courant est 4 et que le symbole en entrée est b, on reste dans l'état 4. La transition est donc représentée par une "boucle" : une flèche qui part de

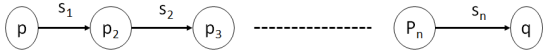
et arrive sur l'état 4.

Mot et langage reconnu

Mot

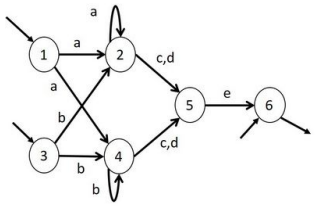
Un **mot** écrit sur l'alphabet $\mathcal{A}$ est une suite de symboles de l'alphabet :
$X = s_1 s_2 s_3 ... s_n$
On a donc $X \in \mathcal{A}^*$ (X peut éventuellement être vide).

Notons ce mot $X_{p,q}$ s'il existe dans $\mathcal{T}$ un ensemble de transitions telles que :
$p = p_1$
$\forall\ 1 \le i \le n, \exists\ (p_i, s_i, p_{i+1}) \in \mathcal{T}$
$q = p_{n+1}$
Nous dirons alors que le mot $X_{p,q}$ "fait passer l'automate de l'état p à l'état q" : il existe une suite de transitions qui, l'une après l'autre (étiquettes successives dans l'ordre des symboles du mot), permettent de passer de l'état p à l'état q.



En termes de théorie des graphes, nous dirons qu'il existe un chemin (ici une suite d'arcs) dont les étiquettes successives forment le mot X, et menant du sommet p au sommet q.

Exemple :



Le mot " aaac " est de la forme $X_{1,5}$ avec les transitions successives :
(1,a,2) (2,a,2) (2,a,2) (2,c,5)

Le mot " bbce " est de la forme $X_{3,6}$ avec les transitions successives :
(3,b,4) (4,b,4) (4,c,5) (5,e,6)

Le même mot " bbce " est également de la forme $X_{4,6}$ avec les transitions successives :
(4,b,4) (4,b,4) (4,c,5) (5,e,6)

Un **mot** $X_{p,q}$ est **reconnu par l'automate** s'il le fait passer d'un état initial à un état terminal, c'est-à-dire :
Un mot de la forme $X_{p,q}$ est reconnu par l'automate si $p \in \mathcal{I}$ et $q \in \mathcal{T}$

Pour montrer qu'un mot est reconnu par un automate, il suffit de trouver une paire " état initial / état terminal " telle qu'indiquée ci-dessus.
Il peut y en avoir plusieurs, cela ne change rien au résultat (nous ne dirons pas pas qu'un mot est, par exemple, "deux fois reconnu").

Si le mot $X_{p,q}$ pris en exemple est également de la forme $X_{p,r}$ avec 'r' n'étant pas un état terminal, le mot est néanmoins considéré comme reconnu par l'automate. C'est-à-dire que toutes les suites de transitions partant de l'état p et correspondant à la suite de symboles ne doivent pas nécessairement mener à un état terminal : il suffit qu'il y en ait au moins une.

ATTENTION :
Pour une même suite de symboles X, les valeurs de p et q ne sont pas nécessairement uniques. C'est à dire que pour un X donné :
- il peut y avoir plusieurs couples d'états (p,q) tels que X soit de la forme $X_{p,q}$;
- pour un couple (p,q) d'états, il peut y avoir plusieurs suites de transitions dont les étiquette successives forment le mot X ;
- ceci est également vrai pour un mot reconnu, à savoir qu'il peut exister plusieurs couples (p,q) $\in \mathcal{I} \times \mathcal{T}$.
Nous verrons que ceci n'est pas le cas si l'automate est dit "déterministe".

Dans le cas d'un automate dit asynchrone, la suite des transitions peut comporter des transitions étiquetées 'ε'. Il

s'agit en fait d'un symbole spécial dont la signification est "aucun symbole de l'alphabet".

Mot vide

Le mot vide, qui ne contient donc aucun caractère, est reconnu par l'automate s'il permet "de passer d'un état initial à un état terminal", comme tous les autres mots reconnus.
Puisqu'il n'est composé d'aucun symbole, sa reconnaissance laissera l'automate dans son état initial (ou un de ses états initiaux, s'il en contient plusieurs).
Pour que le mot vide soit reconnu par l'automate, il faut donc que celui-ci dispose d'un état qui soit à la fois initial et terminal. Si 'i' est un tel état, alors le mot vide est de la forme $X_{i,i}$.
On a dans ce cas $i \in \mathcal{I} \cap \mathcal{T}$.

Cas d'un automate contenant au moins une transition "ε" :
Se reporter aux chapitres "automate asynchrone" et "fermeture epsilon".

Langage reconnu

Le **langage reconnu** par un automate est l'ensemble des mots reconnus par celui-ci :
Automate : $< \mathcal{A}, \mathcal{Q}, \mathcal{I}, \mathcal{T}, \mathcal{E} >$
Langage : $L = \{ X_{p,q} \mid p \in \mathcal{I}, q \in \mathcal{T} \}$

Des algorithmes de reconnaissance de mots à l'aide d'automate fini sont décrits plus loin dans ce cours, en fonction de certaines caractéristiques de l'automate.

Expression rationnelle

De même qu'un automate fini donne la définition d'un langage (ensemble des mots reconnus par l'automate), une expression rationnelle permet également de définir un ensemble de mots sous une forme synthétique.

Une expression rationnelle peut donc être utilisée tout simplement pour définir un langage. Cela peut être fait pour traduire tout simplement une description textuelle qui peut être plus ou moins ambigüe, ou tout simplement pour éviter d'avoir recours à de telles descriptions textuelles. Il est également possible de définir l'expression rationnelle correspondant à un automate fini existant.
A l'inverse, il existe aussi une technique de construction automatique d'automate à partir de son expression rationnelle.

Définition

Une **expression rationnelle** est une formule permettant de décrire un langage au moyen des symboles de l'alphabet et d'opérateurs.
Précisément, elle peut contenir :
- les symboles de l'alphabet,
- le mot vide noté ε,
- des opérateurs, notamment de concaténation, de choix, de puissance.

Une expression rationnelle représente un ensemble de mots.
Un langage peut donc être défini par une expression rationnelle.

Exemples d'expressions rationnelles simples :

a b c	un 'a', suivi d'un 'b' puis d'un 'c' (la concaténation est représentée simplement en notant les éléments les uns après les autres)
a + b	un 'a' ou un 'b' (l'opérateur '+' représente un choix entre ses deux opérandes)
a (b + c) d	un 'a', suivi d'un 'b' ou d'un 'c', le tout suivi d'un 'd' c'est-à-dire soit "abd", soit "acd"
a b* c	un 'a', suivi d'un nombre quelconque (éventuellement 0) de 'b', le tout suivi d'un 'c' (le '*' est un opérateur à un seul opérande, signifiant la répétition un nombre quelconque de fois, éventuellement 0 fois)

	exemples : "ac" "abbbbbc" "abc"
a (b c)* d	un 'a', suivi d'un nombre quelconque (éventuellement 0) de séquences 'bc', le tout suivi d'un 'd' exemples : "ac" "abcd" "abcbcbcbcd"
a	un 'a', et rien d'autre
ε	le mot vide

Selon les outils utilisés, d'autres opérateurs peuvent être utilisés pour la constitution d'expressions rationnelles.

Une expression rationnelle peut bien entendue être écrite en utilisant d'autres expressions rationnelles. Par exemple, si X, Y et Z sont elles-mêmes des expressions rationnelles, on peut par exemple définir l'expression
a X* b (Y + Z) c
qui correspond à tous les mots
- qui débutent par un 'a'
- suivi d'un certain nombre de fois (éventuellement 0) une suite de symboles conforme à l'expression X
- suivi d'un 'b'
- puis d'une suite de symboles correspondant soit à l'expression Y, soit à l'expression Z
- et se terminant par un 'c'.

Attention :
Lors de l'interprétation d'une expression rationnelle, il est important de bien maîtriser la portée des différents opérateurs. Au besoin, utiliser des parenthèses. Par exemple :
ab*c n'est pas la même chose que (ab)*c
a+b* n'est pas la même chose que (a+b)*

Opérateurs

Opérateurs utilisés pour la combinaison d'éléments de l'alphabet ou d'expressions rationnelles.

Notation:
Dans les définitions qui suivent, l'écriture de " $u \in X$ " signifie que 'u' correspond à une suite de symboles conforme à l'expression rationnelle X.

Concaténation

Si X et Y sont deux expressions rationnelles, la concaténation de l'une après l'autre se note simplement XY.
$\mathbf{XY} = \{ uv \mid u \in X, v \in Y \}$

Choix (alternative)

Si X et Y sont deux expressions rationnelles, le choix entre l'une ou l'autre se note X+Y.
$\mathbf{X+Y} = \{ u \mid u \in X \cup Y \}$

Puissance

Si X est une expression rationnelle, la concaténation de n occurrences de X se note $\mathbf{X^n}$.
$X^1 = X$
$\forall\ n \geq 2,\ X^n = \{ u'u \mid u' \in X^{n-1}, u \in X \}$

Pour représenter un nombre quelconque de X, éventuellement 0, on note $\mathbf{X^*}$.
$X^* = \cup_{n \geq 0} (X^n)$

Pour représenter un nombre quelconque de X, mais au moins une fois, on note $\mathbf{X^+}$.
$X^+ = XX^* = X^*X$

> Attention : ne pas confondre le '+' ici noté en exposant, qui signifie "au moins une fois", avec le '+' qui indique l'alternative.

Exemples

Expression rationnelle	Langage

a*b	les mots formés d'un b précédé d'un nombre quelconque de a
(a+b)c	un c précédé d'un a ou d'un b
a(b+c)a	deux a séparés par un b ou un c
(a+ε)b	un b précédé d'un a ou de rien
C^+(.CC*+ε) avec C=0+1+2+3+4+5+6+7+8+9	des nombres composés de : - au moins un chiffre ("C^+") - suivi éventuellement ("+ε") d'un point décimal suivi d'au moins un chiffre (".CC*").

Lemme d'Arden

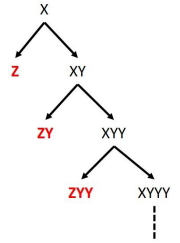
Une propriété importante de ré-écriture d'une expression rationnelle permet d'enlever sa définition récursive.
Nous verrons son utilisation dans le calcul de l'expression rationnelle correspondant à un automate fini.

Règles :

Soit X, Y et Z des expression rationnelles.
- $X = XY+Z \Leftrightarrow X = ZY^*$
- $X = YY+Z \Leftrightarrow X = Y^*Z$

à la condition que le mot vide ε ne fasse pas partie des mots décrits par l'expression rationnelle Y : $\varepsilon \notin Y$.

Démonstration intuitive de la 1ère règle:
X=XY+Z signifie que X est la même chose que XY ou Z.
Si l'on utilise la formule X=XY+Z comme une règle de dérivation, ou règle de transformation de X, essayons d'abord d'appliquer n fois la première transformation possible. On a alors $X = XY = XYY = ... = XY^n$.
Puis, pour finir, utilisons la seconde transformation possible. On a alors $X = ZY^n$.
Dans le schéma ci-dessous, le passage d'un niveau à un niveau immédiatement inférieur représente l'application des deux transformations possibles de X.



Puisque n peut avoir une valeur quelconque, éventuellement nulle, X peut prendre n'importe quelle forme parmi Z, ZY, ZYY, ZYYY, ... Ce sont toutes des réécritures de X qui n'utilisent pas 'X' lui-même.
On a alors de façon générale $X = ZY^*$.

Facteurs - Préfixes et suffixes

Soit $< \mathcal{A}, \mathcal{Q}, \mathcal{I}, \mathcal{T}, \mathcal{E} >$ un automate,
et X un mot reconnu par cet automate.

La suite de symboles S est un **facteur** de X s'il existe deux suites de symboles U et V telles que X peut s'écrire par concaténation de U, puis S, puis V, c'est-à-dire : X= USV.

Si U est vide, S est dit **facteur gauche** de X (X=SV, X "commence" par S).

Si V est vide, S est dit **facteur droite** de X (X = US, X "termine par S).

Soit L le langage reconnu par l'automate.
L'ensemble des facteurs gauche de L est le langage FG(L) reconnu par l'automate $<\mathcal{A},\mathcal{Q},\mathcal{I},\mathcal{Q},\mathcal{E}>$,
à la condition que tous les états de l'automate $<\mathcal{A},\mathcal{Q},\mathcal{I},\mathcal{T},\mathcal{E}>$ soient co-accessibles.
On considère les mêmes "débuts" de mots en conservant les mêmes états initiaux, mais on peut s'arrêter n'importe où en considérant que tous les états sont terminaux.
L'ensemble des facteurs droite de L est le langage FD(L) reconnu par l'automate $<\mathcal{A},\mathcal{Q},\mathcal{Q},\mathcal{T},\mathcal{E}>$,
à la condition que tous les états de l'automate $<\mathcal{A},\mathcal{Q},\mathcal{I},\mathcal{T},\mathcal{E}>$ soient accessibles.
On ne peut reconnaître que les fins de mots possibles de L en gardant les mêmes états terminaux, mais on démarre la reconnaissance d'un facteur droite n'importe où en considérant tous les états comme des états initiaux.

Partant de cette définition générale d'un facteur, intéressons-nous plus particulièrement aux facteurs gauche et droite.

Un **facteur droite** est une suite de symboles qui peut "terminer" un mot. Il s'agit donc d'un **suffixe**.
Un **facteur gauche** est une suite de symboles qui peut "commencer" un mot. Il s'agit donc d'un **préfixe**.
Écrivons X=UV, avec U facteur gauche de X, et V facteur droite de X. Si X est un mot reconnu par un automate donné, alors on peut affirmer qu'il existe un état p tel que :
U est de la forme $U_{i,p}$ avec e un état initial de l'automate
S est de la forme $S_{p,t}$ avec t un état terminal de l'automate.
Par la suite, en englobant tous les états initiaux ou tous les états terminaux, nous pourrons être amenés à utiliser les notations suivantes :
$X_{.,p}$: tous les préfixes de mots menant d'un état initial quelconque à l'état 'p' ;
$X_{p,.}$: tous les suffixes de mots menant de l'état 'p' à un état terminal quelconque.

Ces notions de facteurs gauche / droite, de préfixe / suffixe, trouvent différentes utilisations.
En particulier, la méthode de minimisation d'un automate (trouver l'automate minimal correspondant à un langage) se base sur la notion de suffixe, et de "séparation" des états en fonction des suffixes de mots qu'ils permettent de reconnaître.
Les concepts de préfixes et suffixes sont également sous-jacents aux deux systèmes d'équations pour passer d'un automate fini à son expression rationnelle.

Accessibilité / Co-accessibilité - Automate émondé

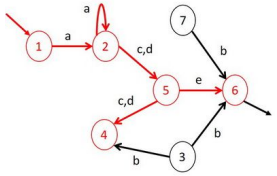
Ces deux notions peuvent servir à déterminer quels sont les états potentiellement utiles dans, par exemple, la reconnaissance de mots. Seuls les états accessibles et co-accessibles sont pertinents.

Accessibilité

"Un état q est accessible s'il existe au moins un mot faisant passer l'automate d'un état initial (quelconque) à l'état q".

Soit un automate $<\mathcal{A},\mathcal{Q},\mathcal{I},\mathcal{T},\mathcal{E}>$
Soit $q \in \mathcal{Q}$
L'état q est dit "**accessible**" si :
$\exists \{ p_1, p_2, \dots , p_{n+1} \} \subseteq \mathcal{Q}$, $p_1 \in \mathcal{I}$, $q = p_{n+1}$
Il existe une suite d'états p_i telle que p_1 soit un état initial, et p_{n+1} l'état considéré
$\exists X = s_1s_2s_3...s_n \in \mathcal{A}^*$
et il existe un mot écrit sur l'alphabet $\mathcal{A}$, éventuellement vide
tels que
$\forall\ 1 \le i \le n, (p_i,s_i,p_{i+1}) \in \mathcal{E}$
tels que : on peut passer de l'état initial p_1 à l'état q par une suite de transitions étiquetées avec la suite de symboles composant X.
C'est-à-dire que le mot X s'écrit sous la forme $X_{p,q}$.

Exemple :



1, 2, 4, 5 et 6 sont accessibles
3, 7 ne sont pas accessibles

En termes de "théorie des graphes" : il existe au moins un chemin d'un état initial à q.

En reprenant une notation utilisée précédemment, on peut également énoncer que q est "accessible" si :
$\exists$ p $\in \mathcal{I}$, $\exists X_{p,q}$ un mot du langage défini sur $\mathcal{A}$ permettant de passer de l'état p à l'état q.

Par extension, on dira qu'un automate est "accessible" si tous ses états sont accessibles.

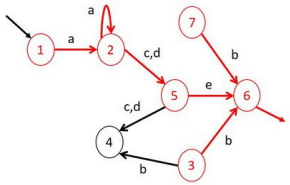
Co-accessibilité

"Un état p est co-accessible s'il existe un mot faisant passer l'automate de l'état p à un état terminal (quelconque)"

La définition ressemble à celle de l'accessibilité, en considérant les transitions en sens inverse.

Soit un automate $<\mathcal{A},\mathcal{Q},\mathcal{I},\mathcal{T},\mathcal{E}>$
Soit $p \in \mathcal{Q}$
L'état p est dit "**co-accessible**" si :
$\exists \{ p_1, p_2, \dots , p_{n+1} \} \subseteq \mathcal{Q}$, $p_{n+1} \in \mathcal{T}$, $p = p_1$
$\exists X = s_1s_2s_3...s_n \in \mathcal{A}^*$
$\forall\ 1 \le i \le n, (p_i,s_i,p_{i+1}) \in \mathcal{E}$

Exemple :



1, 2, 3, 5, 6, 7 sont co-accessibles
4 n'est pas co-accessible

En reprenant la terminologie liée aux graphes : il existe au moins un chemin de p à un état terminal.

On peut également énoncer que p est "co-accessible" si :
$\exists$ q $\in \mathcal{T}$, $\exists X_{p,q}$ un mot du langage défini sur $\mathcal{A}$

Par extension, un automate est dit "co-accessible" si tous ses états sont co-accessibles.

Automate émondé

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

Attention : la notion d' "automate émondé" a une importance toute relative.
En effet, on peut y voir un moyen d'avoir un automate "le plus simple possible", au sens on on élimine ainsi les états qui ne servent à rien dans la reconnaissance des mots, soit parce qu'ils sont inaccessibles depuis un état initial (accessibilité) ou parce que, depuis un état il est impossible d'accéder à un état terminal, donc impossible de reconnaître un mot (co-accessibilité).
Cependant, dans les traitements abordés dans ce cours, notamment dans les mécanismes de passage au langage

complémentaire ou de recherche de l'automate minimal, il est nécessaire de créer d'abord l'automate dit complet, qui dans ce cas perdra la propriété de co-accessibilité.

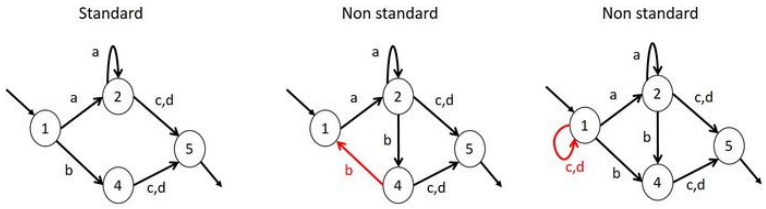
Automate standard

Soit <$\mathcal{A},\mathcal{Q},\mathcal{I},\mathcal{T},\mathcal{E}$> un automate. Il est dit "**standard**" si :

1) il contient un et un seul état initial
   card($\mathcal{I}$) = 1

2) il n'y a aucune transition "entrante" sur cet état initial
   $\mathcal{I}$ = { i }
   $\forall$ p $\in \mathcal{Q}$, $\forall$ x $\in \mathcal{A}$, (p,x,i) $\notin \mathcal{E}$

Exemples



Algorithme et utilisation

L'algorithme de standardisation d'un automate est excessivement simple.
Une fois l'automate rendu standard, il est possible notamment d'utiliser une technique extrêmement simple pour ajouter ou supprimer le mot vide à un langage reconnu par un automate.

Remarque :
La standardisation est souvent (par mes étudiants tout au moins) utilisée en préambule à la déterminisation d'un automate. L'idée est de dire que puisqu'un automate déterministe ne doit contenir qu'un seul état initial (voir la définition d'un automate déterministe), utilisons cette technique de standardisation pour respecter cette première propriété.
Ceci n'a aucun intérêt puisque la technique de déterminisation vue dans ce cours s'occupe de cela. De plus, avoir standardisé l'automate ne simplifie en rien le reste (i.e. la très grande partie) du processus de déterminisation.
Oublions donc cette fausse bonne idée !

Automate déterministe

Avantages / utilisation : Le déterminisme d'un automate a un intérêt dans la mise en œuvre de la reconnaissance de mot.
Il est également nécessaire de disposer d'un automate déterministe pour effectuer d'autres traitements (passage au langage complémentaire, minimisation).

Un automate <$\mathcal{A},\mathcal{Q},\mathcal{I},\mathcal{T},\mathcal{E}$> est dit **déterministe** si :

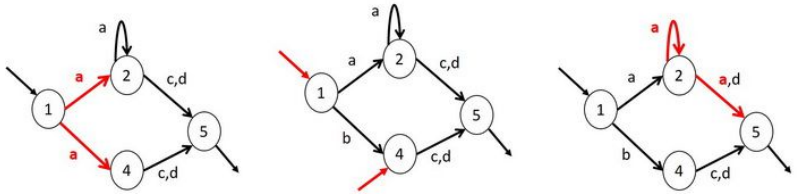
1) il a un et un seul état initial
   card($\mathcal{I}$) = 1

2) pour tout état p et tout symbole x, il existe au plus une transition (p,x,q) dans $\mathcal{E}$
   $\forall$ p $\in \mathcal{Q}$

   $\forall$ x $\in \mathcal{A}$
   card({ (p,x,q) $\in \mathcal{E}$ }) $\leq$ 1

Attention : Certains ouvrages ou sites internet nomme "automate déterministe " un automate qui est "déterministe" et "complet" au sens des définitions utilisées sur MOMIrandom. Notons que certains traitements peuvent exiger que l'automate soit déterministe, sans nécessairement exiger qu'il soit aussi complet. Il est donc préféré ici de séparer les deux propriétés.

Exemples d'automates *non déterministes*



Automate complet

Un automate fini est dit complet si, pour tout état p et tout symbole x, il existe au moins un état q tel que (p,x,q) soit une transition de l'automate.

<$\mathcal{A},\mathcal{Q},\mathcal{I},\mathcal{T},\mathcal{E}$> est **complet** si et seulement si
   $\forall$ p $\in \mathcal{Q}$, $\forall$ x $\in \mathcal{A}$, $\exists$ q $\in \mathcal{Q}$ | (p,x,q) $\in \mathcal{E}$

C'est-à-dire que pour tout état, il existe au moins une transition "sortante" pour chaque symbole de l'alphabet.

Algorithme et utilisation

La technique pour rendre un automate complet est extrêmement simple.

Disposer d'un automate complet a un intérêt particulier dans différents traitements :
- reconnaissance de mots : quelque soit l'état courant et le symbole analysé, il y a toujours une transition ;
- passage au langage complémentaire : l'automate doit impérativement être complet pour effectuer les transformations requises ;
- calcul de l'automate minimal reconnaissant le même langage.

Dans ces deux derniers cas, l'automate doit d'abord avoir été rendu déterministe. Il n'y a pas sur MOMIrandum de cas où il est intéressant de rendre un automate complet sans qu'il soit également déterministe.

Automate minimal

Un automate est dit **minimal** si tout automate reconnaissant le même langage contient au moins autant d'états :

   "minimal" = "le moins d'états possible"

Propriété :
   Pour un alphabet donné et un langage reconnaissable donné, il existe un et un seul automate minimal.

Cette propriété peut notamment être utilisée pour déterminer si deux automates, ou deux expressions rationnelles, ou un automate et une expression rationnelle correspondent au même langage. Il suffit en effet de produire l'automate minimal correspondant aux deux, et de les comparer.

Algorithme

Ce cours présente une technique de minimisation d'un automate. Celui-ci doit d'abord avoir été rendu déterministe et complet.

Langage complémentaire

Soit un langage L défini sur un alphabet $\mathcal{A}$.

$\mathcal{A}^*\backslash L$, le **langage complémentaire** de L sur $\mathcal{A}$, est l'ensemble de tous les mots appartenant à $\mathcal{A}^*$ qui n'appartiennent pas à L.

> Rappel: $\mathcal{A}^*$ est l'ensemble de tous les mots, incluant le mot vide, pouvant être formés au moyen des symboles de $\mathcal{A}$.

Il peut bien entendu être intéressant, dans un processus global de manipulation d'un automate existant, de passer d'un langage à son complémentaire.
La connaissance de ce mécanisme a également son intérêt lors de la création d'un automate, lorsque la définition textuelle du langage contient une négation. Par exemple, pour construire un automate qui reconnaisse tous les mots qui ne contiennent ni la séquence "abc", ni "cba", il peut être plus simple de construire d'abord un automate reconnaissant tous les mots qui contiennent les deux, puis de passer au langage complémentaire.

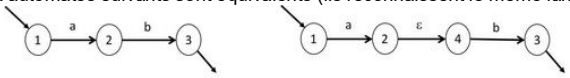
Automate asynchrone (i.e. ayant au moins une transition étiquetée 'ε')

Introduction / Définition

Dans les définitions précédentes, concernant les automates ne contenant aucune transition 'ε', toute transition est définie comme une relation entre 2 états associée à un symbole de l'alphabet (l'étiquette de la transition). Un mot est reconnu par l'automate s'il existe une suite de telles transitions permettant de passer d'un état initial à un état terminal de telle sorte que la suite des étiquettes forme le mot en question.

Les automates dits "asynchrones" permettent d'utiliser un symbole additionnel (à ceux logiquement définis dans l'alphabet de l'automate), noté 'ε', et ne correspondant à aucun symbole effectif. Il est ainsi possible, au moyen d'une transition portant cette étiquette, de passer d'un état à un autre sans représenter de symbole formant le mot.

Par exemple, les deux automates suivants sont équivalents (ils reconnaissent le même langage) :



Le langage ne contient qu'un seul mot : "ab".
Celui-ci peut être reconnu :
- par l'automate de gauche à l'aide des transitions (1,a,2) et (2,b,3) ;
- par l'automate de droite à l'aide des transitions (1,a,2), (2,ε,4) et (4,b,3).
En effet,
à gauche : $X_{1,3} = ab$
à droite : $X_{1,3} = a\varepsilon b = ab$ puisque ε représente l'absence de symbole !

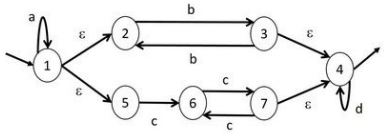
Un automate sera dit **asynchrone** s'il contient au moins une transition spéciale dont l'étiquette est ε.

> Attention : cette terminologie n'est pas fondamentale, en ce sens qu'elle n'est pas utilisée dans tous les ouvrages traitant des automates finis.

> *Remarque : Sur le site MOMIrandum, les automates dits "asynchrones" ne sont manipulés que dans les relations entre les automates finis et les expressions rationnelles. Voir ici le chapitre correspondant. Leur utilisation peut bien entendu être plus large.*

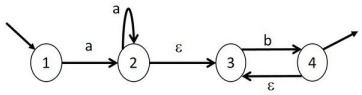
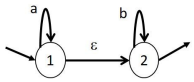
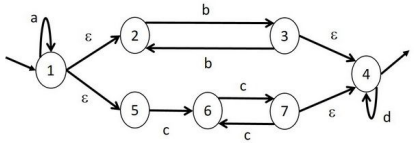
Etat terminal

Supposons l'automate suivant :



L'état terminal identifié comme tel est l'état 4.
Cependant, les états 3 et 7 doivent également être considérés comme des états terminaux car tout mot menant le l'état initial 1 à l'un de ces états peut être complétés par un 'ε' (donc sans modifier le mot) pour utiliser une transition supplémentaire et atteindre 4.
Tout mot pouvant s'écrire sous la forme $X_{1,3}$, par exemple, peut être complété par la chaîne vide, c'est-à-dire qu'on ne lui ajoute rien, et devenir un mot de la forme $X_{1,4}$ et donc être accepté par l'automate.
Le mot "aaabbb" est de la forme $X_{1,3}$. Mais "aaabbb" est égal à "aaabbbε", qui est lui de la forme $X_{1,4}$. Le mot "aaabbb" est donc reconnu par l'automate, donc l'état 3 est considéré lui aussi comme un état terminal.
Les états 3 et 7 sont considérés comme terminaux car leurs "fermetures epsilon" (voir ici) contient un état terminal.

Exemples

Automate asynchrone	*mots reconnus*
	-au moins un 'a' -suivi d'au moins un 'b'
	-un nombre quelconque de 'a' (éventuellement aucun) -suivis d'un nombre quelconque de 'b' (éventuellement aucun) Peut ne contenir ni 'a', ni 'b', donc reconnaît le mot vide. Ne permet pas de mélanger les 'a' et les 'b'.
	-un nombre quelconque de 'a' (éventuellement aucun) - suivis soit d'un nombre impair de 'b' (au moins un), soit d'un nombre pair de 'c' (au moins 2) -puis un nombre quelconque de 'd' (éventuellement aucun).

Retour sur quelques propriétés

Des précisions (définition, interprétation) doivent être apportées sur les propriétés vues précédemment.

Accessibilité / Coaccessibilité
Standard
Déterministe
Complet
Minimal

> ... *bientôt documenté*...

Nous ne parlons (dans ce cours) d'automate minimal que dans le cas d'automate ne contenant aucune transition 'ε'.

Fermeture epsilon

La "fermeture epsilon" d'un état peut se calculer quel que soit l'automate. Elle n'a cependant une utilité effective que lorsque l'automate contient au moins une transition étiquetée 'ε' (automate dit "asynchrone").
Elle est utilisée lors du processus de déterminisation d'un automate asynchrone.
Elle peut aussi être utilisée dans la procédure de reconnaissance de mot, si celle-ci est effectuée à partir d'un automate asynchrone.

Fermeture epsilon d'un état

Soit un automate $<\mathcal{A},\mathcal{Q},\mathcal{I},\mathcal{T},\mathcal{E}>$ asynchrone.
Soit un état p de cet automate.

La "**fermeture epsilon**" de p, que l'on peut noter $\mathbf{p^{\varepsilon}}$, est l'ensemble des états accessibles depuis p par une suite éventuellement vide de transitions étiquetées 'ε' :

Soit $<\mathcal{A},\mathcal{Q},\mathcal{I},\mathcal{T},\mathcal{E}>$ un automate
$p \in \mathcal{Q}$
$p^{\varepsilon} = \{ p \} \cup \{ q \in \mathcal{Q} \mid \exists\, e_i \in \mathcal{Q},\ 1 \le i \le n,\ (e_i,\varepsilon,e_{i+1}) \in \mathcal{E},\ e_1=p,\ e_{n+1}=q \}$

Si il existe une suite de transitions "epsilon" : $p \to e_2 \to e_3 \to \dots e_n \to q$
alors l'état q fait partie de la fermeture epsilon de p.
Lors de la reconnaissance de mot, on peut donc passer de l'état p à l'état q sans progresser dans la lecture du mot.

Attention : un état appartient à sa propre fermeture epsilon.
En conséquence, s'il n'y a aucune transition "epsilon" sortante associée à un état donné, sa fermeture epsilon est réduite à l'état lui-même.
Une fermeture epsilon n'est donc jamais un ensemble vide !

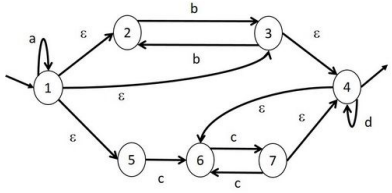
Fermeture epsilon d'un ensemble d'états

On définit la fermeture epsilon sur un ensemble d'états comme l'union des fermetures epsilon des éléments de cet ensemble :

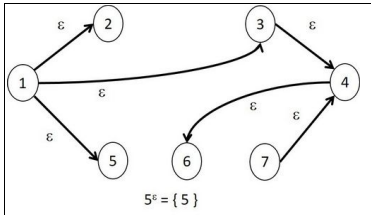
$P=\{p_1,p_2,\dots\} \subseteq \mathcal{Q}$
$P^{\varepsilon} = p_1{}^{\varepsilon} \cup p_2{}^{\varepsilon} \cup p_3{}^{\varepsilon} \cup \dots$

Exemples

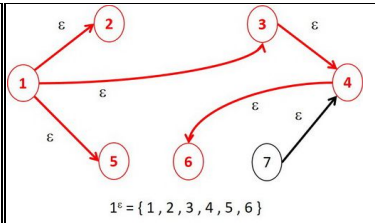
Automate asynchrone



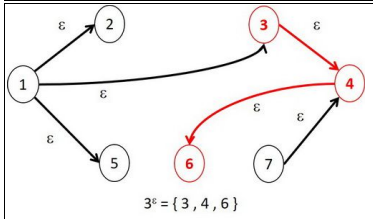
et quelques fermetures epsilon déduites des transitions epsilon de l'automate.
Pour illustrer le calcul, on trace un sous-ensemble du graphe des états dans lequel seules les transitions epsilon sont indiquées.



Il n'y a aucune transition epsilon sortant de l'état 5



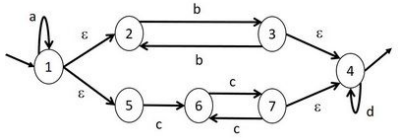
Partant de l'état 1, on identifie les transitions epsilon allant les les états 2, 3, 5.
On poursuit le calcul en partant de l'état 3 (aucune transition epsilon sortant des états 2 et 5). On ajoute donc l'état 4 à la fermeture epsilon de 1.
Depuis l'état 4, on détecte la transition epsilon allant à l'état 6, duquel aucune transition epsilon ne sort.



De l'état 3, on ajoute l'état 4, puis l'état 6.

État terminal et fermeture epsilon

Comme indiqué lors de la définition d'un automate asynchrone, l'état identifié explicitement comme état terminal n'est pas nécessairement le seul état d'acceptation désigné comme tel.
Sur cet exemple :



nous avons indiqué que 3 et 7 sont des états d'acceptation. On remarque que leurs fermetures epsilon contiennent l'état 4.
Cela mène à la définition suivante :

Dans un automate asynchrone, est considéré comme état d'acceptation, ou état terminal, tout état p dont la fermeture epsilon contient au moins un état terminal.
$p \in \mathcal{Q}$ est un état d'acceptation si et seulement si $p^{\varepsilon} \cap \mathcal{T} \neq \varnothing$

MANIPULATIONS D'AUTOMATE(S)

Nous abordons ici différentes techniques qui permettent de transformer un automate, tout en conservant le même langage reconnu (à l'exception de la transformation pour reconnaître le langage complémentaire, bien évidemment).

Automate et reconnaissance de mot

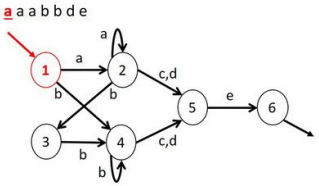
En reprenant les définitions de mot et de mot reconnu, il suffit donc de "progresser" dans l'automate, en partant d'un état initial, en parcourant une à une les transitions relatives à l'état courant et au symbole lu, et vérifier qu'à la fin du traitement, l'état courant est bien un état terminal.
Reconnaître un mot X est donc déterminer s'il est de la forme $X_{p,q}$, avec p un état initial et q un état terminal.

Cas d'un automate déterministe

Considérons tout d'abord le cas simple d'un automate déterministe (voir définition associée), à savoir qu'à chaque étape il y a au plus une possibilité de passage d'un état à un autre.

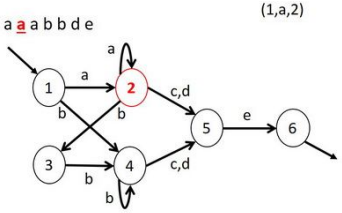
Exemple

Soit un automate et un mot à analyser :

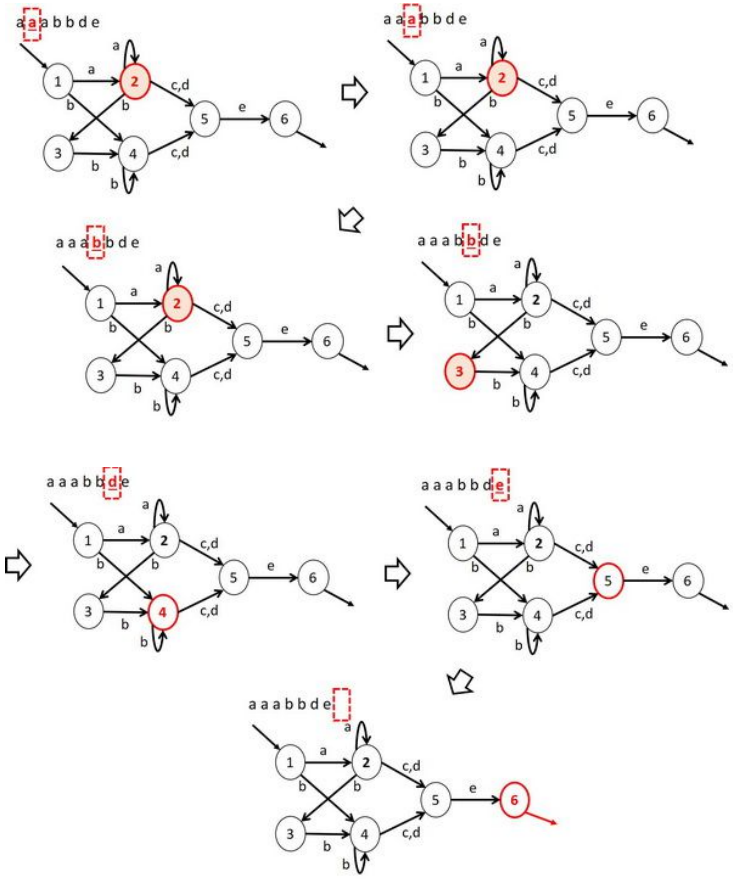


Le "symbole courant" est initialé au premier symbole du mot (en rouge et souligné).
L' "état courant " de l'automate est initialisé avec son état initial "1".

La transition (1,a,2) peut alors être utilisée, puisque son état de départ est l'état courant, et son symbole est le symbole courant.
L'automate "passe" donc de l'état 1 à l'état 2, et le symbole courant (à analyser à l'étape suivant) est décalé d'une position vers la fin du mot.
On obtient ainsi la situation suivante :

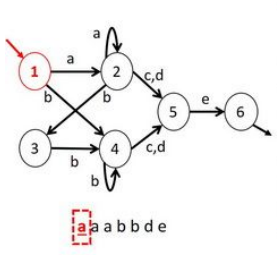


L'état courant est '2' et le symbole courant est 'a'. La transition (2,a,2) peut alors être utilisée pour passer de l'état '2' à l'état '2' (c'est le principe d'une "boucle" sur l'état 2). C'est-à-dire que l'état courant ne change pas, mais le symbole courant est bien entendu décalé.
Les étapes suivantes sont illustrées sans plus explication, tellement elles paraissent évidentes...

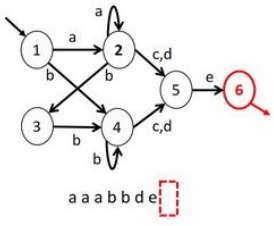


A la dernière étape, le symbole courant est "vide", ou "fin de mot" (le mot en entrée a été entièrement analysé), et l' "état courant" est un état terminal de l'automate. Le mot est donc reconnu.

Le tableau ci-dessous illustre les différentes situations tout au long de l'analyse :



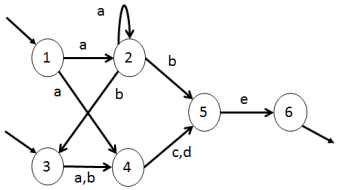
Etat courant	Reste du mot à analyser	Transition applicable
1	aaabbde	(1,a,2)
2	aabbde	(2,a,2)
2	abbde	(2,a,2)
2	bbde	(2,b,3)
3	bde	(3,b,4)
4	de	(4,d,5)
5	e	(5,e,6)
6	∅	



Lien : algorithme (pseudo code) de reconnaissance de mot pour un automate déterministe

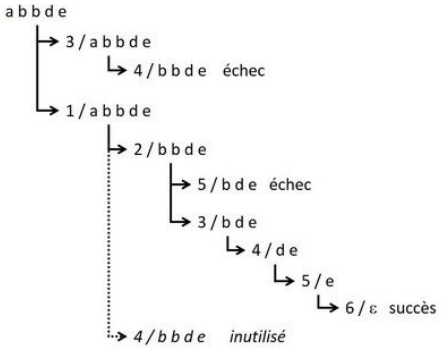
Cas d'un automate non déterministe

L'exemple précédent est simple car à chaque étape de la reconnaissance du mot, il n'y a qu'un seul choix possible, au plus. Y compris au tout début, puisqu'il n'y a qu'un seul état initial.
Ce n'est pas toujours le cas, lorsque l'automate est "non déterministe", c'est-à-dire si au cours de la reconnaissance, il existe une étape lors de laquelle il y a un choix entre plusieurs transitions, ou si dès le départ on a le choix entre plusieurs états initiaux.
Par exemple, essayons de reconnaître le mot " abbde " avec l'automate suivant :



La première question à se poser est : quel doit être l'état courant initial pour pouvoir reconnaître (accepter) le mot ?
Si l'on tente une reconnaissance à partir de l'état 3, on arrive à l'état courant 4 et un "reste de mot" à analyser "bbde". Aucune transition partant de l'état 4 avec le symbole b, donc échec.
Il faut alors se souvenir que 1 est également un état initial, et recommencer.

Soit donc le second départ avec le couple état courant / [reste de] mot à analyser : 1 / abbde.
Après utilisation de la transition (1,a,2), ce couple devient 2 / bbde. Doit-on ensuite aller dans l'état 3 ou l'état 5 ?
Si l'on va en 5, on obtient alors la situation 5 / bde, et une situation de blocage.
Il faut donc revenir en arrière (2 / bbde) et passer à l'état 3 : 3 / bde. On peut ensuite progresser jusqu'à l'état 6 et reconnaître le mot à analyser.
Si cela n'était pas le cas, nous reviendrions au début pour tenter de passer de l'état 1 à l'état 4 en analysant le 1er 'a', et ainsi de suite.
Le schéma ci-dessous illustre les différentes étapes avec les retour en arrière effectués :



L'algorithme de reconnaissance est donc ici un peu plus compliqué, puisqu'il faut sauvegarder quelque part les alternatives rencontrées, et être capable d'y revenir. Ce n'est qu'après avoir essayé toutes les alternatives possibles que l'on peut décider qu'un mot n'est pas reconnaissable.

On peut envisager un algorithme itératif utilisant une pile pour les différentes alternatives. On empile un "contexte" représentant l'état courant et le reste de mot à analysé, comme illustré précédemment. A chaque itération, on prend l'état en haut de pile. Si une ou plusieurs transitions sont applicables, les contextes correspondants sont mis en haut de pile. Si aucune transition n'est applicable, le sommet de pile est supprimé.
Lorsqu'on arrive en fin de mot, si l'état contenu dans le contexte en haut de pile est un état terminal, on accepte le mot. Sinon il est refusé.
Si la pile est vide, le mot est refusé.
Le schéma ci-dessous illustre les transformations de la pile pour l'exemple ci-dessus.

	Départ I={1,3}								6∈T Succès
Pile	3/abbde 1/abbde	4/bbde 1/abbde	1/abbde	2/bbde 4/bbde	5/bde 3/bde 4/bde	3/bde 4/bde	4/de 4/bde	5/e 4/bde	6/ε 4/bde
Transitions	(3,a,4)	(4,b,) échec	(1,a,2) (1,a,4)	(2,b,5) (2,b,3)	(5,b,) échec	(3,b,4)	(4,d,5)	(5,e,6)	

... *pseudo code bientôt disponible* ...

On peut également utiliser un algorithme récursif. La "pile" évoquée ci-dessus est alors gérée par les appels récursifs sur les différentes options disponibles à un moment donné.

... *pseudo code bientôt disponible* ...

Cas d'un automate avec des transitions epsilon

La complexité vient ici de la nécessité de pouvoir "passer au travers" de ces transitions epsilon. Différentes techniques sont envisageables, dont le calcul des fermetures epsilon.

... *pseudo code et exemple bientôt disponibles* ...

Complétion - Etat poubelle

Rappel : Un automate fini est dit complet si, pour tout état p et tout symbole x, il existe au moins un état q tel que (p,x,q) soit une transition de l'automate.

$<\mathcal{A},\mathcal{Q},\mathcal{I},\mathcal{T},\mathcal{E}>$ est **complet** si et seulement si
$\forall p \in \mathcal{Q}, \forall x \in \mathcal{A}, \exists q \in \mathcal{Q} \mid (p,x,q) \in \mathcal{E}$

Technique de complétion

Un façon simple d'obtenir un automate complet équivalent (i.e. reconnaissant le même langage) à un automate non complet est d'ajouter un état dit "poubelle", et d'y associer les transitions manquante :

Soit $<\mathcal{A},\mathcal{Q},\mathcal{I},\mathcal{T},\mathcal{E}>$ un automate non complet.
L'automate complet correspondant peut être défini par $<\mathcal{A},\mathcal{Q}_c,\mathcal{I},\mathcal{T},\mathcal{E}_c>$ où :

- $\mathcal{Q}_c = \mathcal{Q} \cup \{ P \}$ — *ajout de l'état poubelle P*
- $\mathcal{E}_c = \mathcal{E}$ — *toutes les transitions de l'automate non complet sont gardées dans l'automate*
  - $\cup \{ (q,x,P) \mid \neg (\exists q' \in \mathcal{Q} \mid (q,x,q') \in \mathcal{E}) \}$ — *ajout de (q,x,P) pour chaque transition (q,x,q') absente de $\mathcal{E}$*
  - $\cup \{ (P,x,P) \mid x \in \mathcal{A} \}$ — *ajout des transitions (P,x,P) pour tout l'alphabet*

L'ajout de l'état poubelle ne peut en aucun cas ajouter des mots au langage, puisque la poubelle est un état qui n'est pas co-accessible : lorsque la reconnaissance d'un mot mène à l'état poubelle, aucune transition à venir ne peut atteindre un état terminal (l'état courant de la reconnaissance de mot reste cet état poubelle jusqu'au dernier symbole à analyser).

Attention : l'oubli des transitions (P,x,P) laisse l'automate non complet !

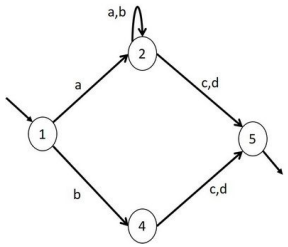
Une fois l'automate rendu complet, l'état poubelle est alors traité comme les autres.

Remarque : L'état poubelle est ici nommé 'P', mais ce n'est bien entendu qu'une convention utilisée à des fins didactiques.
Note à l'attention des étudiants pour leurs projets : si les états de l'automate sont représentés par des nombres, par exemple de 0 à 15, l'état poubelle peut tout simplement être nommé "16". Inutile de mettre en œuvre une

gymnastique algorithmique dans vos programmes !

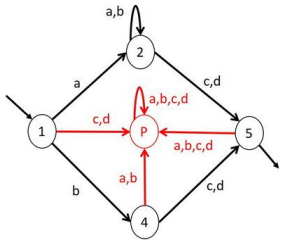
Exemples

Non complet



		a	b	c	d
→	1	2	4		
	2	2	2	5	5
	4			5	5
←	5				

Complet équivalent



		a	b	c	d
→	1	2	4	P	P
	2	2	2	5	5
	4	P	P	5	5
←	5	P	P	P	P
	P	P	P	P	P

Remarques :
* Il n'y a pas beaucoup d'intérêt à vouloir rendre complet un automate quelconque. Cette propriété est cependant nécessaire dans certains cas, par exemple :
- pour calculer un automate minimal ;
- pour obtenir aisément un automate reconnaissant le langage complémentaire de celui reconnu par un automate donné.
La complétion se fera donc dans la chaîne de traitement déterminisation-complétion-minimisation, ou la chaîne déterminisation-complétion-passage au langage complémentaire. Rendre un automate complet juste avant de le rendre déterministe n'a par vraiment d'utilité : il est nettement préférable d'effectuer les traitements dans l'ordre inverse.
* La programmation de la reconnaissance d'un mot est également plus simple si l'automate est complet. Cette amélioration n'est toutefois intéressante que si l'automate est déterministe (la complexité de programmation liée à un automate non déterministe est bien supérieure à celle d'un automate non complet).

Standardisation

Soit $<\mathcal{A},\mathcal{Q},\mathcal{I},\mathcal{T},\mathcal{E}>$ un automate non standard.
Soit $<\mathcal{A},\mathcal{Q}',\mathcal{I}',\mathcal{T}',\mathcal{E}'>$ l'automate standard équivalent.

Méthode de standardisation

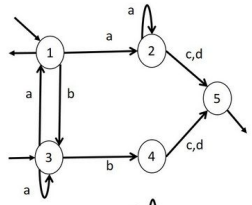
La standardisation d'un automate passe par 3 étapes :
1. Ajout d'un état initial, noté ici 'i'
2. Ajout de cet état initial à la liste des états terminaux si nécessaire (si l'automate non standard dispose d'un état qui est à la fois initial et terminal)
3. Duplication au départ de 'i' de toutes les transitions partant d'un état initial de l'automate à standardiser.

1.
$\mathcal{Q}' = \mathcal{Q} \cup \{ i \}$
Ajout d'un nouvel état
$\mathcal{I}' = \{ i \}$
Le nouvel état devient le seul état initial

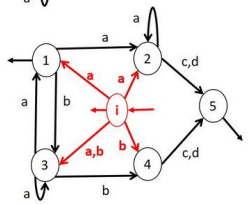
2.
$\mathcal{I} \cap \mathcal{T} = \varnothing \Rightarrow \mathcal{T}' = \mathcal{T}$
$\mathcal{I} \cap \mathcal{T} \neq \varnothing \Rightarrow \mathcal{T}' = \mathcal{T} \cup \{ i \}$
Le nouvel état initial devient un état terminal si et seulement si au moins un état initial de l'automate à standardiser est également un état terminal.
cela permet à l'automate d'accepter le mot vide si et seulement si l'automate initial l'accepte.

3.
$\mathcal{E}' = \mathcal{E} \cup \{ (i,x,q) \mid \exists\, p \in \mathcal{I}, (p,x,q) \in \mathcal{E} \}$
On garde toutes les transitions de l'automate à standardiser, auxquelles on ajoute la reproduction (duplication) depuis l'état 'i' de toute transition sortante d'un état initial ('p') de l'automate à standardiser

Exemples

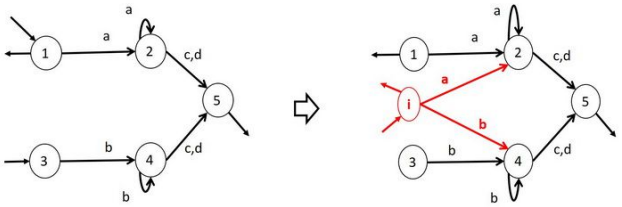


'1' est état initial et terminal
Transitions issues d'un état initial :
(1,a,2)
(1,b,3)
(3,a,1)
(3,a,3)
(3,b,4)



'i' devient état terminal
Transitions ajoutées :
(i,a,2)
(i,b,3)
(i,a,1)
(i,a,3)
(i,b,4)

Remarque :
La standardisation d'un automate peut rendre certains états non-accessibles. La "simplification" de l'automate n'est cependant pas partie intégrante de la standardisation. Dans l'exemple ci-dessous, les états 1 et 3 sont inutiles (non accessibles) après standardisation.



La mise en œuvre automatique et systématique de la simplification peut être une erreur. Dans un système opérationnel plus complexe que la seule standardisation, on peut tout à fait imaginer une chaîne de traitements dans laquelle rendre accessible un automate juste après l'avoir standardisé serait une erreur du point de vue de l'objectif final visé.

Ajout / élimination du mot vide

Supposons vouloir éliminer le mot vide du langage L reconnu par un automate A.
Puisque le mot vide est reconnu, A dispose donc d'un état initial qui est également état terminal. Notons 'i' cet état.
Si l'automate est standard, alors le mot vide 'ε' est de la forme $X_{i,i}$.
Supposons que l'état 'i' ne soit plus un état terminal. Pour que seul le mot vide soit alors enlevé du langage, il faut que $X_{i,i} = \{ \varepsilon \}$, c'est-à-dire qu'aucune autre suite de symbole autre que le mot vide ne mène à cet 'i'. Ceci est assurément le cas si l'état 'i' n'accepte aucune transition entrante. C'est à dire s'il est standard.

Un raisonnement similaire peut être tenu pour l'ajout du seul mot vide à un langage reconnu par un automate existant.

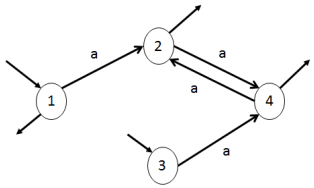
Passage au langage complémentaire

Soit <$\mathcal{A},\mathcal{Q},\mathcal{I},\mathcal{T},\mathcal{E}$> un automate déterministe complet.
L'automate reconnaissant le langage complémentaire est tout simplement : <$\mathcal{A},\mathcal{Q},\mathcal{I},\mathcal{Q}\setminus\mathcal{T},\mathcal{E}$>.

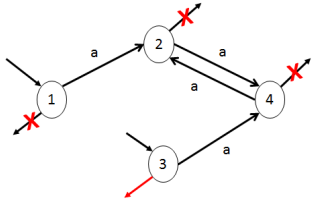
Les états terminaux deviennent non terminaux, et vice-versa.

Méthode excessivement simple, mais encore faut-il s'assurer que l'automate <$\mathcal{A},\mathcal{Q},\mathcal{I},\mathcal{T},\mathcal{E}$> soit :

- Déterministe



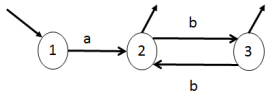
Le mot vide est accepté



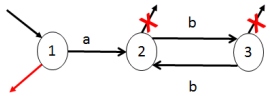
Le mot vide est encore accepté

L'automate est non déterministe (2 états initiaux). L'état 3, état initial, devient un état terminal et permet donc à nouveau de reconnaître le mot vide.

- Complet



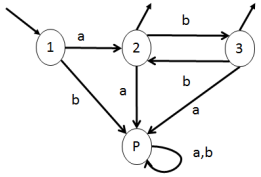
Mots reconnus :
Un 'a' suivi d'un nombre quelconque de 'b'

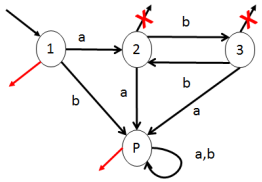


Ne reconnait que le mot vide !

Il n'y a pas de transition sur 'b' partant de l'état 1 (automate non complet). Donc, tout mot commençant par 'b', qui n'est pas reconnu par l'automate initial, ne pourra pas être reconnu après transformation puisque aucune transition (1,b,?) ne pourra mener à un état terminal.

Ce problème, et tous les autres sont résolus en ayant rendu l'automate de départ complet :





Déterminisation d'un automate

Le processus de déterminisation peut être appliqué à un automate fini quelconque. Il n'y a pas de phase préparatoire requise.

En particulier, comme nous l'avons évoqué dans les chapitres consacrés aux automates standards, il n'est pas nécessaire en préambule de standardiser l'automate afin d'avoir un état initial unique.
A ce propos, notons que le résultat d'une déterminisation directe (avec les mécanismes décrits dans ce chapitre) d'un automate donnera un résultat différent de celui résultant de l'enchaînement standardisation + déterminisation.
A noter également, comme cela a déjà été indiqué, que la déterminisation d'un automate ne nécessite pas que l'automate initial soit complet. Compléter un automate non déterministe n'a d'ailleurs pas grand intérêt.

La déterminisation s'effectue sur un automate contenant ou pas des transitions epsilon. Pour une raison didactique, nous commencerons par décrire le processus de déterminisation pour des automates ne contenant pas de transition epsilon. Nous étendons ensuite ce mécanisme pour prendre en compte de telles transitions.
On notera ensuite que le second algorithme, plus général, peut s'appliquer tel quel à un automate ne contenant pas de transition epsilon.
Important : la déterminisation d'un automate contenant des transitions epsilon est un automate déterministe n'en contenant pas.

En résultat de la technique présentée ici, on obtient un automate déterministe qui n'est pas nécessairement complet. La complétion de cet automate est considéré dans ce cours comme un traitement supplémentaire différent.

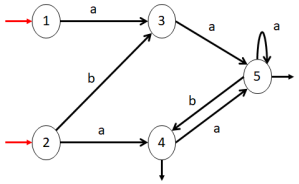
Propriété notable : les états qui sont créés au long du processus sont tous accessibles depuis l'état initial. On ne créé pas d'état "inutile".

Principe général :

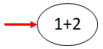
On construit un automate déterministe dans lequel chaque état p correspond à un ensemble d'états (p_1,p_2,...} de l'automate non déterministe : être dans l'état 'p' de l'automate déterministe est équivalent à être dans l'un des états p_i de l'automate non déterministe.
A partir de la définition de l'état initial de l'automate déterministe (ensemble des états initiaux de l'automate non déterministe), on construit itérativement les états et transitions de l'automate déterministe de la façon suivante :
si p = {p_1,p_2,...}, la transition (p,x,q) est créée, ainsi que l'état q, avec q={q_1,q_2,...} l'ensemble des états de l'automate non déterministe accessibles par une transition étiquetée 'x' depuis l'un des états p_i. C'est à dire qu'il existe au moins un i et un j tel que la transition (p_i,x,q_j) existe dans l'automate non déterministe.

Exemple introductif - Automate Fini sans transition epsilon

Prenons un automate simple :

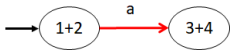


Il dispose de 2 états initiaux.
Afin de satisfaire la condition (propriété) d'un automate déterministe, il suffit de construire un automate dont le seul état initial consistera à être soit dans l'état 1, soit dans l'état 2. Nommons cet état "1+2".
On obtient l'état initial de l'automate déterministe :

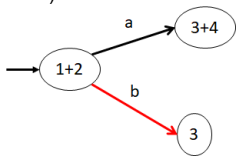


Puisque être dans l'état "1+2" revient à être soit dans l'état "1" soit dans l'état "2" de l'automate non déterministe, il nous faut considérer l'équivalence entre les deux situations.
Considérons le symbole 'a'.
L'automate non déterministe dispose de deux transitions (1,a,3) et (2,a,4).
Donc, en étant soit dans "1" soit dans "2", on peut aller, en lisant un 'a', soit dans "3" soit dans "4". Représentons cette possibilité par la création d'un état "3+4". On ajoute ainsi un état et une transition dans l'automate

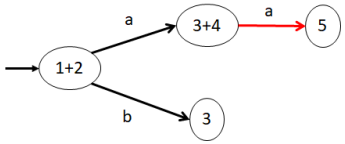
déterministe :



En faisant de même pour le symbole 'b', on est amené à créer dans l'automate déterministe l'état "3" (issue de la transition (2,b,3) de l'automate non déterministe) et la transition associée :



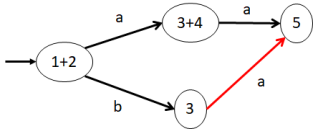
On procède de même en considérant l'état "3+4" (soit 3, soit 4 dans l'automate non déterministe). Aucune transition n'est créé pour le symbole 'b' puisqu'il n'en existe aucune dans l'automate non déterministe depuis les états 3 et 4. En considérant l'ensemble des transitions étiquetées 'a' issues des états 3 et 4 (transitions (3,a,5) et (4,a,5)), on obtient alors :



On remarque ici que, étant dans l'état 3 ou l'état 4 de l'automate initial, si le symbole courant est 'a', on passe dans l'état 5 ou dans l'état 5. On ne va bien entendu pas créer l'état "5+5" car être dans 5 ou dans 5 est équivalent à être dans 5 ! Il n'y a pas de doublon dans la composition des états de l'automate déterministe.

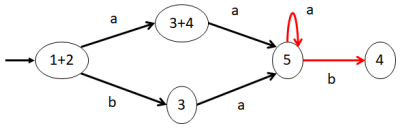
Procédons de même pour tous les états qui sont itérativement créés dans l'automate déterministe :

En considérant ensuite l'état "3" de l'automate déterministe, on obtient :

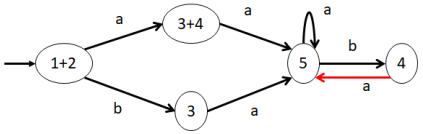


L'état "5" ayant déjà été créé dans l'automate déterministe, il n'est bien évidemment pas dupliqué.

Puis l'état "5" :



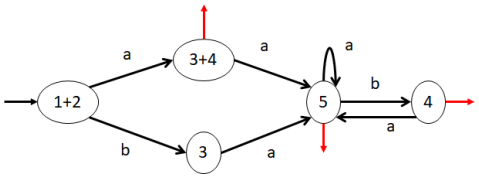
Et finalement l'état "4" dernièrement créé :



Tous les états et les transitions de l'automate déterministe ont été créés.
Reste à déterminer les états terminaux de l'automate déterministe.

Dans l'automate non déterministe, les états 4 et 5 sont des états terminaux.

Dans l'automate déterministe, tout état qui représente, entre autres, la possibilité d'être dans l'état 4 ou dans l'état 5 est considéré comme un état terminal. C'est donc le cas pour l'état "3+4" (soit 3, soit 4), et plus simplement pour les états "4" et "5" eux-même. On obtient ainsi l'automate déterministe :



Algorithme de déterminisation - Automate sans transition epsilon

Soit <$\mathcal{A}$,$\mathcal{Q}$,$\mathcal{I}$,$\mathcal{T}$,$\mathcal{E}$> un automate non déterministe,
et <$\mathcal{A}$,$\mathcal{Q}$',$\mathcal{I}$',$\mathcal{T}$',$\mathcal{E}$'> l'automate déterministe équivalent.

Lors du processus de déterminisation, nous définissons progressivement les états et les transitions de l'automate déterministe.
Durant ce processus, les états de l'automate déterministe sont identifiées par des sous-ensembles de $\mathcal{Q}$. Tous les sous-ensembles de $\mathcal{Q}$ ne sont pas cependant pas nécessairement dans $\mathcal{Q}$'.

On commence par l'état initial de l'automate déterministe, qui est égal à $\mathcal{I}$, partie de $\mathcal{Q}$.
Les autres états sont créés au fur et à mesure du traitement, et sont traités les uns après les autres.
On utilise un mécanisme de marquage pour identifier les états créés qui n'ont pas encore été pris en compte pour ajouter les transitions sortantes associées.

Méthode de déterminisation

3 étapes :
- création de l'état initial
- boucle de création des autres états et des transitions
- identification des états terminaux

Ci-dessous l'algorithme seul. Voir juste après pour quelques explications.
Notation : on nomme ici p, q, ... les états de l'automate non déterministe, et p', q', ... ceux de l'automate déterministe.

- $\mathcal{Q}' \leftarrow \{ \mathcal{I} \}$
  $\mathcal{I}' \leftarrow \{ \mathcal{I} \}$
  marque($\mathcal{I}$) = à_traiter
- TANT QUE { e' $\in \mathcal{Q}'$ | marque(e') = à_traiter } $\neq \varnothing$
  Choisir p' $\in \mathcal{Q}'$ | marque(p') = à_traiter
  - $\forall$ x $\in \mathcal{A}$,
    Soit q' = { q $\in \mathcal{Q}$ | $\exists$ p $\in$ p', (p,x,q) $\in \mathcal{E}$ }
    q' $\notin \mathcal{Q}' \Rightarrow$ ($\mathcal{Q}' \leftarrow \mathcal{Q}' \cup$ { q' } , marque(q') = à_traiter)
    $\mathcal{E}' \leftarrow \mathcal{E}' \cup$ { (p',x,q') }
  - marque(p') = déjà_traité
- $\mathcal{T}'$ = { p' $\in \mathcal{Q}'$ | $\exists$ p $\in$ p', p $\in \mathcal{T}$ }

Explications

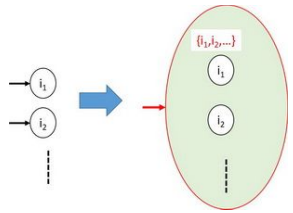
- $\mathcal{Q}' \leftarrow \{ \mathcal{I} \}$
  $\mathcal{I}' \leftarrow \{ \mathcal{I} \}$

L'automate déterministe est initialisé avec son état initial seul.
$\mathcal{I}$ est l'ensemble des états initiaux de l'automate non déterministe. C'est donc bien un sous-ensemble de Q.

marque($\mathcal{I}$) = à_traiter

Initialisation de l'ensemble des états (de l'automate déterministe) à traiter. "Traiter" un état, c'est déterminer ses transitions "sortantes" sur les différents symboles de l'alphabet, ainsi que les états cible associés.
Au départ, seul l'état initial est connu. C'est donc pour l'instant le seul "à traiter".
Tout état qui sera créé durant le processus de déterminisation sera marqué de la sorte afin de calculer ultérieurement ses transitions sortantes pour chaque symbole de l'alphabet.

Illustration :
i_1, i_2, ... sont les états initiaux de l'automate non déterministe ; l'état I={i_1,i_2,...} est d'abord défini comme l'état initial de l'automate déterministe :



Pour reprendre la notation utilisée dans l'exemple introductif, cet état initial serait noté " i_1+i_2+... "

- TANT QUE { e' ∈ $\mathcal{Q}$' | marque(e') = à_traiter } ≠ ∅

  "tant que l'ensemble des états à traiter n'est pas vide"
  Lorsque tous les états de l'automate déterministe, créés itérativement à partir de son état initial, auront eux-même été traités, le processus de création des états et des transitions prend fin.

  Soit p' ∈ $\mathcal{Q}$' | marque(p') = à_traiter

  On choisit un état qui n'a pas encore été traité.
  Le choix peut être quelconque. A la première itération, il n'y a bien entendu que l'état initial.
  Rappel : p' est une partie de Q, c'est-à-dire un ensemble d'états de l'automate non déterministe.
  On peut donc écrire p' = { p_1 , p_2, ... } ou, selon la notation de l'exemple introductif, p' = p_1 + p_2 + ...

  - ∀ x ∈ $\mathcal{A}$,

    On considère tous les éléments de l'alphabet, les uns après les autres, et on calcule les transitions sortantes, si elle existent. Les états cibles sont créés s'ils n'existent pas déjà dans l'automate déterministe.

    Soit q' = { q ∈ $\mathcal{Q}$ | ∃ p ∈ p', (p,x,q) ∈ $\mathcal{E}$ }

    p' est un ensemble d'états de l'automate non déterministe.
    On considère tous les éléments p de p' pour lesquels il existe, dans l'automate non déterministe, une transition sortante (p,x,q).
    On calcule ainsi q' qui est l'ensemble des états de l'automate non déterministe accessibles (dans l'automate non déterministe) par une transition étiquetée 'x' partant d'un état (de l'automate non déterministe) appartenant à p'.

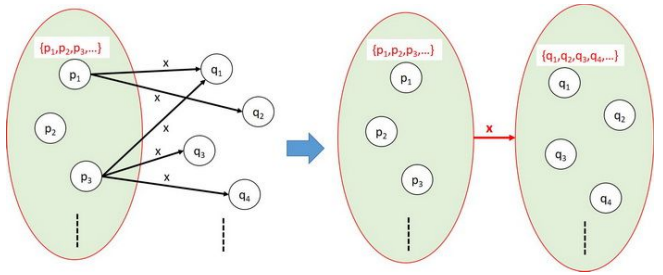
    $\mathcal{E}$' ← $\mathcal{E}$' ∪ { (p',x,q') }

    La transition (p',x,q') est ajoutée à l'automate déterministe

    q' ∉ $\mathcal{Q}$' ⇒ ($\mathcal{Q}$' ← $\mathcal{Q}$' ∪ { q' } , marque(q') = à_traiter)

    Si l'état q' n'est pas déjà présent dans l'automate déterministe, il y est ajouté. Il est alors marqué "à traiter".

    Illustration :



p' = { p_1,p_2,p_3,... } et q' = { q_1,q_2,q_3,q_4,...}
Il existe au moins une transition étiquetée x de l'un des éléments de p' vers chacun des états q_1, q_2, q_3, q_4, ...
On créé donc dans l'automate déterministe un état q' qui les regroupe et on ajoute une transition (p',x,q')

  - marque(p') = déjà_traité

    On indique que p' a été traité

Remarque :
Concrètement, lorsqu'on exécute cet algorithme, on construit progressivement la table de transitions. La boucle "tant que" portant sur les états de l'automate déterministe consiste à passer d'une ligne à la suivante, jusqu'à traitement de toutes les lignes. L'ajout d'un nouvel état consiste en l'ajout d'une nouvelle ligne en fin de table.
La séquence "∀ x ∈ $\mathcal{A}$" qui consiste à prendre en compte tous les symboles de l'alphabet, revient à prendre en compte les différentes colonnes de la table, les unes après les autres également.
La table de transitions de l'automate déterministe est ainsi "construite et parcourue" en même temps.
Voir illustrations.

- $\mathcal{T}$' = { p' ∈ $\mathcal{Q}$' | ∃ p ∈ p', p ∈ $\mathcal{T}$ }

  Les états terminaux de l'automate déterministes sont tous les états contenant au moins un état terminal dans l'automate non déterministe.

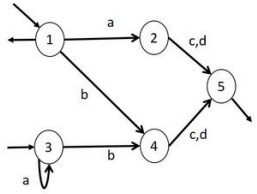
Illustrations

Pour chacun des exemples, nous donnons :
- l'automate non déterministe sous forme de schéma et sous forme de table des transitions,
- les différentes étapes de la table des transitions de l'automate déterministe, en indiquant à chaque étape les états restant à traiter,
- l'automate déterministe obtenu sous forme de table de transitions et de schéma.

Pour l'automate déterministe, nous utiliserons une écriture simplifiée des états : celle utilisée dans l'exemple introductif avec le signe "+".

Exemple #1

Automate non déterministe (plusieurs états initiaux)



		a	b	c	d
→	1	2	4		
←					
	2			5	5
→	3	3	4		
	4			5	5
←	5				

Déterminisation

Etape 1 :
→ **création de l'état initial**
boucle de création des autres états et des transitions
identification des états terminaux

On initialise la table des transitions avec la ligne de l'état initial

	a	b	c	d
→ {1,3}				

Etape 2 :
création de l'état initial
→ **boucle de création des autres états et des transitions**
identification des états terminaux

Lorsqu'on traite un état particulier, on remplit sa ligne dans la table des transitions, et on ajoute une ligne pour chaque nouvel état créé.
Lorsqu'un état est traité, on passe à la ligne suivante.
Lorsqu'on a traité l'état de la dernière ligne sans ajouter de nouvel état, on a terminé le processus.

L'étape 2 démarra donc avec l'état initial {1,3}, le seul créé lors de l'étape 1.
Pour chaque symbole de l'alphabet, on considère les transitions dans l'automate non déterministe partant des états 1 et 3:
pour 'a' : (1,a,2) (3,a,3)
pour 'b' : (1,b,4) (3,b,4)
pour 'c' : aucune
pour 'd' : aucune
Considérant être, dans l'automate non déterministe soit dans '1', soit dans '3', selon le symbole analysé, on passe :
pour 'a' : dans '2' ou dans '3'
pour 'b' : dans '4'
pour 'c' et 'd' : nulle part
D'où la création des états {2,3} et {4} ainsi que des transitions associées.
Ces états étant nouveaux, ils sont ajoutés à Q' et marqués "à traiter".

	a	b	c	d
→ {1,3}	{2,3}	{4}		
{2,3}				
{4}				

On continue ensuite la boucle de création des états et des transitions :

	a	b	c	d
→ {1,3}	{2,3}	{4}		
{2,3}	{3}	{4}	{5}	{5}
{4}				
{3}				
{5}				

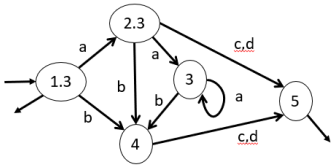
	a	b	c	d
→ {1,3}	{2,3}	{4}		
{2,3}	{3}	{4}	{5}	{5}
{4}			{5}	{5}
{3}	{3}	{4}		
{5}				

Etape 3 :
création de l'état initial
boucle de création des autres états et des transitions
→ **identification des états terminaux**

Tout état de l'automate déterministe contenant au moins un état terminal de l'automate non déterministe devient un état terminal.

	a	b	c	d
→ {1,3}	{2,3}	{4}		
{2,3}	{3}	{4}	{5}	{5}
{4}			{5}	{5}
{3}	{3}	{4}		
← {5}				

Automate déterministe obtenu (notation simplifiée)

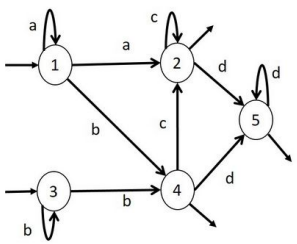


	a	b	c	d
→ 1.3	2.3	4		
2.3	3	4	5	5
4			5	5
3	3	4		
← 5				

Plutôt que de noter les états sous forme ensembliste, on simplifie la notation en indiquant tout simplement la liste des états de l'automate non déterministe. Afin de ne pas confondre avec la notation simplifiée utilisée pour les automates non déterministes, les éléments sont ici séparés par un point.
Par exemple :
pour un automate non déterministe :
E[p,x] = 1,2
indique qu'il existe deux transitions (p,x,1) et (p,x,2) ; 'p', '1' et '2' sont trois états de l'automate, 'x' est un symbole
pour un automate déterministe :
E[p,x] = 1.2
indique qu'il existe une transition (p,x,1.2) ; 'p' et '1.2' sont deux états de l'automate, 'x' est un symbole.

Exemple #2

Automate non déterministe



AFND

	a	b	c	d
→ 1	1,2	4		
← 2			2	5
→ 3		3,4		
← 4			2	5
← 5				5

Attention : Dans ce tableau, les notations sous forme de liste ne correspondent pas à la notation simplifiée utilisée pour les automates déterministes.
Ainsi, la mention "1,2" dans la case [1,a] signifie qu'il y a 2 transitions partant de l'état 1 et étiquetées 'a' : l'une allant vers l'état 1 et l'autre vers l'état 2.

Déterminisation

1) création de l'état initial
boucle de création des autres états et des transitions

identification des états terminaux

	a	b	c	d
→ 1.3				

1) création de l'état initial
2) boucle de création des autres états et des transitions
identification des états terminaux

	a	b	c	d
→ 1.3	1.2	3.4		
1.2				
3.4				

	a	b	c	d
→ 1.3	1.2	3.4		
1.2	1.2	4	2	5
3.4				
4				
2				
5				

	a	b	c	d
→ 1.3	1.2	3.4		
1.2	1.2	4	2	5
3.4		3.4	2	5
4				
2				
5				

	a	b	c	d
→ 1.3	1.2	3.4		
1.2	1.2	4	2	5
3.4		3.4	2	5
4			2	5
2				
5				

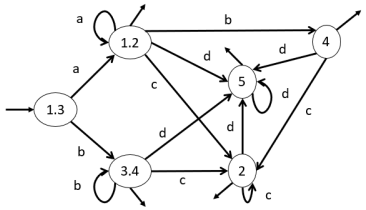
	a	b	c	d
→ 1.3	1.2	3.4		
1.2	1.2	4	2	5
3.4		3.4	2	5
4			2	5
2			2	5
5				

	a	b	c	d
→ 1.3	1.2	3.4		
1.2	1.2	4	2	5
3.4		3.4	2	5
4			2	5
2			2	5
5				5

1) création de l'état initial
2) boucle de création des autres états et des transitions
3) identification des états terminaux

	a	b	c	d
→ 1.3	1.2	3.4		
← 1.2	1.2	4	2	5
← 3.4		3.4	2	5
← 4			2	5
← 2			2	5
← 5				5

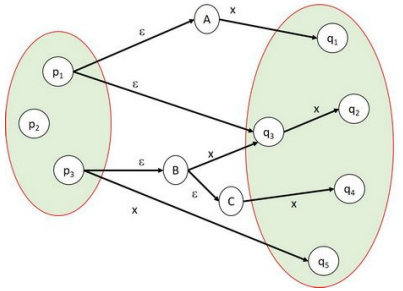
Automate déterministe obtenu



Cas des transitions epsilon - Principe

Le principe de la déterminisation d'un automate contenant des transitions epsilon est similaire à celui de la déterminisation d'un automate n'en contenant pas, auquel on ajoute le traitement particulier des fermetures epsilon.

La différence réside donc dans la recherche des états "cible" depuis un état de l'automate déterministe. Plutôt que de prendre en compte uniquement les transitions étiquetées par un symbole de l'alphabet (différent de ε), on autorise que ces transitions soient précédées par une suite de transitions epsilon. Dans l'exemple ci-dessous, à supposer que l'état $\{p_1,p_2,p_3\}$ ait été créé, l'automate déterministe disposera d'une transition ($\{p_1,p_2,p_3\}$, x , $\{q_1,q_2,q_3,q_4,q_5\}$) :



Suites de transitions utilisées :

$(p_1,\varepsilon,A)\ (A,x,\mathbf{q_1})$
$(p_1,\varepsilon,q_3)\ (q_3,x,\mathbf{q_2})$
$(p_3,x,\mathbf{q_5})$
$(p_3,\varepsilon,B)\ (B,x,\mathbf{q_3})$
$(p_3,\varepsilon,B)\ (B,\varepsilon,C)\ (C,x,\mathbf{q_4})$

Ceci revient donc, avant de prendre en compte les transitions étiquetées avec le symbole x, de rechercher les états accessibles avec des transitions epsilon.
Dans l'exemple ci-dessus, partant de $\{p_1,p_2,p_3\}$, on passe par la recherche de $\{p_1,p_2,p_3,A,q_3,B,C\}$ avant de prendre en compte les transitions sur x. C'est-à-dire pas la fermeture epsilon de $\{p_1,p_2,p_3\}$.

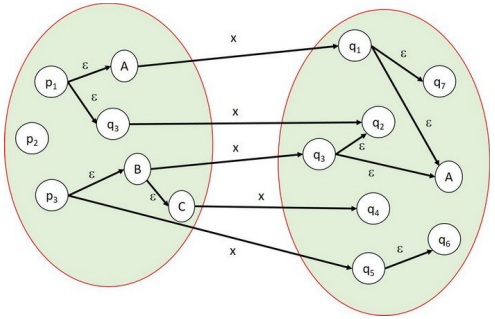
Afin de rendre le traitement le plus simple possible par rapport à l'algorithme vu précédemment (i.e. lorsqu'il n'y a aucune transition epsilon), on procède comme suit :
lors de la création d'un état par l'algorithme précédent, $\{p_1,p_2,p_3\}$ dans notre exemple ci-dessus, celui-ci est immédiatement remplacé par sa fermeture epsilon $\{p_1,p_2,p_3\}^{\varepsilon}$.
Ce principe est appliqué une première fois lors de la création de l'état initial de l'automate déterministe.
C'est répété tout au long du processus de calcul des transitions sortantes, avec création éventuelle d'un nouvel état.

Algorithme de déterminisation - Automate avec transitions epsilon

Par rapport à l'algorithme de déterminisation d'un automate ne contenant pas de transition epsilon, la transformation consiste simplement à remplacer chaque état créé par sa fermeture epsilon. On ajoute ainsi à un ensemble d'états tous ceux qui sont accessibles "en ne lisant rien", c'est-à-dire sans déplacer le symbole courant en entrée d'analyse.

Introduction

Considérons la création, avec la technique décrite précédemment, d'un état cible $\{p_1,p_2,p_3\}$.
Dans l'illustration ci-dessous, on remarque que, partant de l'un de ces états, des transitions epsilon permettent d'atteindre d'autres états.
La fermeture epsilon de $\{p_1,p_2,p_3\}$ est :
$\{p_1,p_2,p_3\}^{\varepsilon}$ = { p_1 , p_2 , p_3 , A , q_3 , B , C }
C'est cet état qui aura été créé durant le processus de déterminisation.



Attention : dans un graphe des états, 'q_3' n'apparaît bien évidemment qu'une seule fois ! De même pour 'A'.

Ensuite, considérant le symbole 'x', on identifie les transitions sortantes de l'un des états de cette fermeture epsilon : q_1, q_2, q_3, q_4, q_5.
L'état cible qui va être créé n'est pas l'ensemble de ces 5 états, mais sa fermeture epsilon :
$\{q_1,q_2,q_3,q_4,q_5\}^{\varepsilon}$ = { q_1 , q_2 , q_3 , q_4 , q_5 , q_6 , q_7 , A }

Algorithme de déterminisation avec transitions epsilon

Soit $<\mathcal{A},\mathcal{Q},\mathcal{I},\mathcal{T},\mathcal{E}>$ un automate non déterministe contenant des transitions epsilon,
et $<\mathcal{A},\mathcal{Q}',\mathcal{I}',\mathcal{T}',\mathcal{E}'>$ l'automate déterministe correspondant.

- $\mathcal{Q}' \leftarrow \{ \mathcal{I}^{\varepsilon} \}$
  $\mathcal{I}' \leftarrow \{ \mathcal{I}^{\varepsilon} \}$
  marque($\mathcal{I}^{\varepsilon}$) = à_traiter
  L'état initial de l'automate déterministe est la fermeture epsilon de l'ensemble des états initiaux de l'automate non déterministe.

- TANT QUE { p' ∈ $\mathcal{Q}'$ | marque(p') = à_traiter } ≠ ∅
  Choisir p' ∈ $\mathcal{Q}'$ | marque(p') = à_traiter
  p' est la fermeture epsilon d'un ensemble d'états de l'automate non déterministe.
  Par exemple p'=$\{p_1,p_2,p_3\}^{\varepsilon}$=$\{p_1,p_2,p_3,q_3,A,B,C\}$ dans l'exemple ci-dessus.
  - ∀ x ∈ $\mathcal{A}$,
    Soit q' = { q ∈ $\mathcal{Q}$ | ∃ p ∈ p', (p,x,q) ∈ $\mathcal{E}$ }
    On calcule ici l'ensemble q' comme dans le cas de la déterminisation d'un automate ne contenant pas de transition epsilon.
    Il suffit ici de prendre en compte les transitions étiquetées epsilon issues des éléments de p'.
    $q'^{\varepsilon} \notin \mathcal{Q}' \Rightarrow$ ($\mathcal{Q}' \leftarrow \mathcal{Q}' \cup \{ q'^{\varepsilon} \}$, marque(q'^{ε}) = à_traiter)
    $\mathcal{E}' \leftarrow \mathcal{E}' \cup \{ (p',x,q'^{\varepsilon}) \}$
    Mais on n'ajoute pas l'état q' dans l'automate déterministe : on ajoute sa fermeture epsilon !
  - marque(p') = déjà_traité

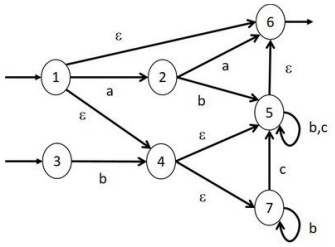
- $\mathcal{T}'$ = { p' ∈ $\mathcal{Q}'$ | ∃ p ∈ p', p ∈ $\mathcal{T}$ }
  Les états terminaux de l'automate déterministe sont les états (fermetures epsilon) contenant au moins un état terminal de l'automate asynchrone non déterministe.

La différence par rapport à la déterminisation d'un automate ne contenant pas de transition epsilon n'est pas "majeure" :

Déterminisation automate sans transition epsilon	*Déterminisation automate avec transitions epsilon*
• $\mathcal{Q}' \leftarrow \{ \mathcal{I} \}$ $\mathcal{I}' \leftarrow \{ \mathcal{I} \}$ marque($\mathcal{I}$) = à_traiter • TANT QUE { p' ∈ $\mathcal{Q}'$ | marque(p') = à_traiter } ≠ ∅ Choisir p' ∈ $\mathcal{Q}'$ | marque(p') = à_traiter ○ ∀ x ∈ $\mathcal{A}$, Soit q' = { q ∈ $\mathcal{Q}$ | ∃ p ∈ p', (p,x,q) ∈ $\mathcal{E}$ } q' ∉ $\mathcal{Q}' \Rightarrow$ ($\mathcal{Q}' \leftarrow \mathcal{Q}' \cup \{ q' \}$, marque(q') = à_traiter) $\mathcal{E}' \leftarrow \mathcal{E}' \cup \{ (p',x,q') \}$ ○ marque(p') = déjà_traité • $\mathcal{T}'$ = { p' ∈ $\mathcal{Q}'$ | ∃ p ∈ p', p ∈ $\mathcal{T}$ }	• $\mathcal{Q}' \leftarrow \{ \mathcal{I}^{\varepsilon} \}$ $\mathcal{I}' \leftarrow \{ \mathcal{I}^{\varepsilon} \}$ marque($\mathcal{I}^{\varepsilon}$) = à_traiter • TANT QUE { p' ∈ $\mathcal{Q}'$ | marque(p') = à_traiter } ≠ ∅ Choisir p' ∈ $\mathcal{Q}'$ | marque(p') = à_traiter ○ ∀ x ∈ $\mathcal{A}$, Soit q' = { q ∈ $\mathcal{Q}$ | ∃ p ∈ p', (p,x,q) ∈ $\mathcal{E}$ } q'^{ε} ∉ $\mathcal{Q}' \Rightarrow$ ($\mathcal{Q}' \leftarrow \mathcal{Q}' \cup \{ q'^{\varepsilon} \}$, marque(q'^{ε}) = à_traiter) $\mathcal{E}' \leftarrow \mathcal{E}' \cup \{ (p',x,q'^{\varepsilon}) \}$ ○ marque(p') = déjà_traité • $\mathcal{T}'$ = { p' ∈ $\mathcal{Q}'$ | ∃ p ∈ p', p ∈ $\mathcal{T}$ }

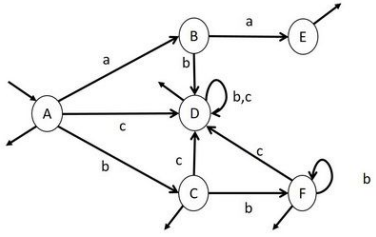
Exemple

Automate asynchrone non déterministe :

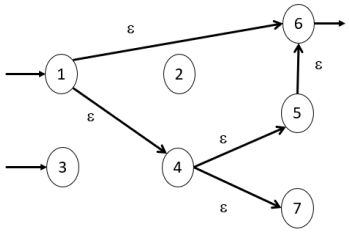


	a	b	c	ε
1	2			4,6
2	6	5		
3		4		
4				5,7
5		5	5	6
6				
7		7	5	

Automate déterministe synchrone correspondant :



Calcul des fermetures epsilon :
Il peut être judicieux, mais pas absolument nécessaire, de commencer par le calcul des fermetures epsilon de chaque état.
C'est assez simple à faire "graphiquement" si l'automate n'est pas trop compliqué. Voir la section précédente pour un algorithme plus systématique.
Commençons par ne considérer que les transitions epsilon du graphe des états. Il est ensuite aisé de remplir un tableau des fermetures epsilon de chaque état.



p	p^{ε}
1	1 , 4 , 5 , 7, 6
2	2
3	3
4	4 , 5 , 6 , 7
5	5 , 6
6	6
7	7

Si l'on doit ensuite calculer la fermeture epsilon d'un ensemble d'états, il suffit de faire l'union de plusieurs lignes du tableau. Par exemple, la fermeture epsilon de {3,5} est l'union de {3} et de {5,6}.

Procédons maintenant à la déterminisation proprement dite.

Initialisation :

- $\mathcal{Q}' \leftarrow \{ I^{\varepsilon} \}$
  $\mathcal{I}' \leftarrow \{ I^{\varepsilon} \}$

$\mathcal{I} = \{ 1 , 3 \}$
$I^{\varepsilon} = \{ 1 , 3 , 4 , 5 , 6 , 7 \}$

L'état initial créé pour l'automate déterministe est la fermeture epsilon de l'ensemble des états initiaux de l'automate non déterministe, soit {1,3,4,5,6,7}.

	a	b	c
→ {1,3}ε={1,3,4,5,6,7}			

Boucle de création des états et transitions :

- TANT QUE { p' ∈ $\mathcal{Q}'$ | marque(p') = à_traiter } ≠ ∅
  Choisir p' ∈ $\mathcal{Q}'$ | marque(p') = à_traiter
  - ∀ x ∈ $\mathcal{A}$,
    Soit q' = { q ∈ $\mathcal{Q}$ | ∃ p ∈ p', (p,x,q) ∈ $\mathcal{E}$ }
    $q'^{\varepsilon} \notin \mathcal{Q}' \Rightarrow$ ($\mathcal{Q}' \leftarrow \mathcal{Q}' \cup \{ q'^{\varepsilon} \}$, marque(q'^{ε}) = à_traiter)
    $\mathcal{E}' \leftarrow \mathcal{E}' \cup \{ (p',x,q'^{\varepsilon}) \}$
  - marque(p') = déjà_traité

p' = { 1,3,4,5,6,7 }

Il n'est pas nécessaire dans cette méthode de calculer la fermeture epsilon de l'état p', puisqu'il est lui-même le résultat de la fermeture epsilon d'un autre état. On aurait de toute façon $p'^{\varepsilon} = p'$.

Pour chaque symbole x, il suffit de chercher l'ensemble de tous les états cible des transitions partant des éléments contenus dans p'.
Par exemple, pour x=b, on a les transitions (3,b,4), (5,b,5) et (7,b,7). Il n'y a pas de transition sortante pour les états 1, 4 et 6.
L'ensemble des états cible est dont {4,5,7}. On créé l'état qui est sa fermeture epsilon : $\{4,5,7\}^{\varepsilon}$. La transition sortant de l'état p' est ajoutée.

Calculs pour les 3 symboles de l'alphabet :

x=a → (1,a,2) → 2 $\{2\}^{\varepsilon}=\{2\}$
x=b → (3,b,4) (5,b,5) (7,b,7) → 4,5,7 $\{4,5,7\}^{\varepsilon}=\{4,5,6,7\}$
x=c → (5,c,5) (7,c,5) → 5 $\{5\}^{\varepsilon}=\{5,6\}$

On obtient la table suivante :

	a	b	c
→ {1,3}ε={1,3,4,5,6,7}	{2}ε	{4,5,7}ε	{5}ε
{2}ε={2}			
{4,5,7}ε={4,5,6,7}			
{5}ε={5,6}			

Suite du déroulement de l'algorithme :

p' = { 2 }
x=a → (2,a,6) → 6 $\{6\}^{\varepsilon}=\{6\}$
x=b → (2,b,5) → 5
x=c → /

	a	b	c
→ {1,3}ε={1,3,4,5,6,7}	{2}ε	{4,5,7}ε	{5}ε
{2}ε={2}	{6}ε	{5}ε	
{4,5,7}ε={4,5,6,7}			
{5}ε={5,6}			
{6}ε={6}			

p' = { 4,5,6,7 }

	a	b	c
→ {1,3}ε={1,3,4,5,6,7}	{2}ε	{4,5,7}ε	{5}ε
{2}ε={2}	{6}ε	{5}ε	
{4,5,7}ε={4,5,6,7}		{5,7}ε	{5}ε
{5}ε={5,6}			
{6}ε={6}			
{5,7}ε={5,6,7}			

p' = { 5,6 }

	a	b	c
→ {1,3}ε={1,3,4,5,6,7}	{2}ε	{4,5,7}ε	{5}ε
{2}ε={2}	{6}ε	{5}ε	
{4,5,7}ε={4,5,6,7}		{5,7}ε	{5}ε
{5}ε={5,6}		{5}ε	{5}ε
{6}ε={6}			
{5,7}ε={5,6,7}			

p' = { 6 }

	a	b	c
→ {1,3}ε={1,3,4,5,6,7}	{2}ε	{4,5,7}ε	{5}ε
{2}ε={2}	{6}ε	{5}ε	
{4,5,7}ε={4,5,6,7}		{5,7}ε	{5}ε
{5}ε={5,6}		{5}ε	{5}ε
{6}ε={6}			
{5,7}ε={5,6,7}			

p' = { 5,6,7 }

	a	b	c
→ {1,3}ε={1,3,4,5,6,7}	{2}ε	{4,5,7}ε	{5}ε
{2}ε={2}	{6}ε	{5}ε	
{4,5,7}ε={4,5,6,7}		{5,7}ε	{5}ε
{5}ε={5,6}		{5}ε	{5}ε
{6}ε={6}			
{5,7}ε={5,6,7}		{5,7}ε	{5}ε

Identification des états terminaux :

- $\mathcal{T}' = \{ p' \in \mathcal{Q}' \mid \exists p \in p', p \in \mathcal{T} \}$

$\mathcal{T} = \{ 6 \}$
Donc, dans l'automate déterministe, tous les états qui contiennent '6' sont des états terminaux.

		a	b	c
→ ←	$\{1,3\}^\varepsilon=\{1,3,4,5,6,7\}$	{2}	{4,5,6,7}	{5,6}
	$\{2\}^\varepsilon=\{2\}$	{6}	{5,6}	
←	$\{4,5,7\}^\varepsilon=\{4,5,6,7\}$		{5,6,7}	{5,6}
←	$\{5\}^\varepsilon=\{5,6\}$		{5,6}	{5,6}
←	$\{6\}^\varepsilon=\{6\}$			
←	$\{5,7\}^\varepsilon=\{5,6,7\}$		{5,6,7}	{5,6}

Comme pour la déterminisation d'un automate ne contenant aucune transition epsilon, on peut bien entendu utiliser la notation simplifiée. L'état initial sera noté 1.3.4.5.6.7 par exemple.

Calcul des fermetures epsilon

Soit $<\mathcal{A},\mathcal{Q},\mathcal{I},\mathcal{T},\mathcal{E}>$ un automate asynchrone.

La fermeture epsilon d'un état, ou d'un ensemble d'états, peut se calculer par un parcours de l'automate dans lequel on ne considère que les transitions epsilon.
Lors de ce parcours, il faut bien entendu prendre garde de ne pas traiter plusieurs fois le même état. On utilise un mécanisme de marquage.

Fermeture epsilon d'un état 'e'

Il s'agit d'un traitement itératif dans lequel les états sont ajoutés les uns après les autres.
On démarre avec l'état 'e' lui-même.
Puis on recherche tous les états p de l'automate tels qu'une transition (e,ε,p) existe. Ils sont ajoutés à la fermeture epsilon de e.
On les considère alors les uns après les autres, en faisant la même chose.
Le traitement s'arrête lorsque aucun nouvel état n'est ajouté à la fermeture epsilon calculée.

- e^ε = { e }
  marque(e) = à_traiter
  La fermeture epsilon est initialisée par l'état lui-même.
  Un état marqué "à traiter" est un état pour lequel on doit chercher les transitions epsilon sortantes, pour ajouter les états cibles dans la fermeture epsilon calculée.
- TANT QUE $\exists$ p $\in$ e^ε | marque(p) = à_traiter
  Soit p $\in$ e^ε | marque(p) = à_traiter
  On sélectionne un état qui est dans la fermeture epsilon, et qui n'a pas déjà été traité (i.e. on n'a pas encore considéré les transitions epsilon qui en partent).
  L'ordre de choix n'a aucune importance.
  - $\forall$ q $\in$ $\mathcal{Q}$ \ e^ε | (p,ε,q) $\in$ $\mathcal{E}$, ($e^\varepsilon \leftarrow e^\varepsilon \cup$ { q } , marque(q) = à_traiter)
    Ayant choisi un état p, on recherche tous les états q accessibles depuis p par une transition epsilon (p,ε,q).
    On ne considère que les états qui n'ont pas déjà été insérés dans la fermeture epsilon calculée.
    Les états q obtenus sont alors ajoutés à la fermeture epsilon, et marqués comme devant être traités
  - marque(p) = déjà_traité
    L'état sélectionné pour l'itération en cours est marqué comme ayant déjà été considéré

Fermeture epsilon d'un ensemble P d'états

On peut considérer une mise en œuvre dans laquelle on calcule la fermeture epsilon de chacun des états contenus dans l'ensemble P, et on fait l'union de toutes ces fermetures epsilon (la fermeture epsilon d'un ensemble d'états est l'union des fermetures epsilon des éléments de l'ensemble).

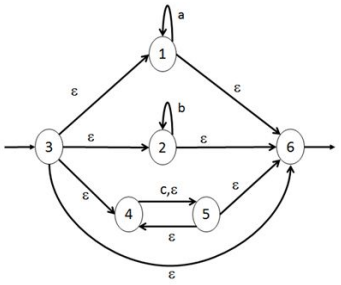
On peut toutefois reprendre l'algorithme ci-dessus en ne modifiant que la phase d'initialisation : tous les états de l'ensemble sont mis dans la fermeture à calculer, et ils sont tous considérés "à traiter".

- P^ε = P
  $\forall$ e $\in$ P, marque(e) = à_traiter
  La fermeture epsilon est initialisée par l'ensemble des états P lui-même
- TANT QUE $\exists$ p $\in$ P^ε | marque(p) = à_traiter
  Soit p $\in$ P^ε | marque(p) = à_traiter
  - $\forall$ q $\in$ $\mathcal{Q}$ \ P^ε | (p,ε,q) $\in$ $\mathcal{E}$, ($P^\varepsilon \leftarrow P^\varepsilon \cup$ { q } , marque(q) = à_traiter)
  - marque(p) = déjà_traité

Il existe bien évidemment d'autres mises en œuvre possible.

Exemple

Considérons l'automate suivant :



Calcul de 1^ε :

Etat déjà traité	*Etat à traiter*	
	1	Initialisation avec l'état considéré
1	6	transition (1,ε,6)
1 6		pas de transition epsilon partant de 6

Calcul de 3^ε :

Etat déjà traité	*Etat à traiter*	
	3	
3	1 2 4 6	transitions epsilon partant de 3 vers 1, 2, 4 et 6
3 1	2 4 6	transition de 1 vers 6, mais 6 déjà présent dans la fermeture epsilon
3 1 2	4 6	
3 1 2 4	6 5	
3 1 2 4 6	5	
3 1 2 4 6 5		

Calcul de $\{1,2,4\}^\varepsilon$:

Etat déjà traité	*Etat à traiter*	
	1 2 4	Initialisation avec l'ensemble dont on calcule la fermeture epsilon
1	2 4 6	
1 2	4 6	
1 2 4	6 5	
1 2 4 6	5	

12465

Minimisation

Le processus de minimisation vise à réduire le nombre d'états.

Il ne peut s'appliquer que sur un automate déterministe et complet.
Il doit également être "accessible" (tout état est "accessible" par une suite de transitions partant de l'état initial).
Remarque : l'automate n'est pas nécessairement coaccessible. En particulier s'il a été rendu complet avec la méthode de complétion d'un automate, l'état "poubelle" ajouté" n'est pas coaccessible (aucune suite de transitions ne mène de l'état poubelle à un état terminal).

Equivalence de Nérode

Soit $< \mathcal{A}, \mathcal{Q}, \{i\}, \mathcal{T}, \mathcal{E} >$ un automate déterministe complet.

On dit que deux états **p et q sont séparés par le mot X** de $\mathcal{A}^*$ si l'une ou l'autre des deux conditions suivantes est satisfaite :
$X = X_{p,t}$, $t \in \mathcal{T} \Rightarrow \neg \exists\, t' \in \mathcal{T}$, $X = X_{q,t'}$
Si le mot X permet de passer de l'état p à un état terminal t, alors ce n'est pas le cas en partant de l'état q.
et vice versa

L'**équivalence de Nérode** permet de diviser $\mathcal{Q}$ en un ensemble de parties telles que 2 états appartiennent à la même partie si et seulement s'ils ne sont séparés par aucun mot sur A*. En d'autres termes, 2 états appartiennent à la même partie si et seulement si en partant de ces deux états on peut reconnaître les mêmes suffixes de mots.

Ce principe de séparation selon les suffixes des mots est à la base du principe de minimisation décrit dans ce cours.

Principe de minimisation

Le principe est basé sur la différenciation des états à partir desquels les suffixes des mots reconnaissables sont différents (voir équivalence de Nérode).

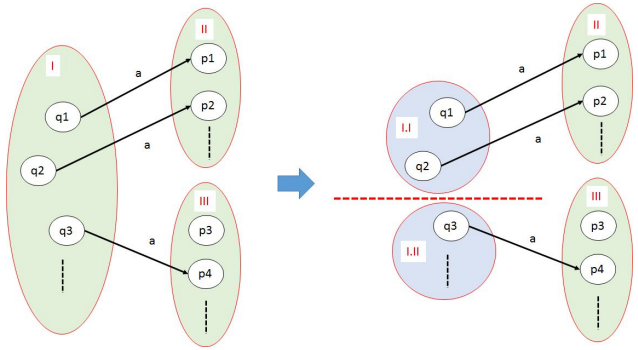
Intuitivement, le principe de base permettant de réduire le nombre d'états serait de fusionner deux états p et q si l'ensemble des suffixes reconnus à partir de ces deux états est le même. C'est-à-dire, en reprenant une notation déjà utilisée, si on a $X_{p,-} = X_{p,-}$

En fait, on ne sait pas comment procéder à cette fusion. La technique utilisée est l'inverse : on sépare deux états lorsque l'on est certain que l'égalité ci-dessus n'est pas respectée. Ainsi :
- Au départ, on sépare les états terminaux des autres :
si p est un état non terminal alors que t en est un, on sait que le suffixe vide peut être reconnu à partir de t, alors qu'il ne l'est pas à partir de p :
$\varepsilon \in X_{t,-}$ et $\varepsilon \notin X_{p,-}$
donc
$X_{t,-} \neq X_{p,-}$
et il sont donc séparés dans deux ensembles (parties) différentes.
- De façon itérative, on regarde si des états dans une même partie doivent être séparés : c'est le cas si on détecte au moins un suffixe reconnu à partir d'un état mais pas de l'autre.
- On arrête le processus lorsque aucune partie ne peut plus être divisée (deux itérations donnent le même résultat, ou toutes les parties sont réduites à un seul élément).

Dans l'exemple ci-dessous, on suppose avoir séparé l'ensemble des états en 3 parties distinctes nommées I, II et III.

Les transitions sur le symbole 'a' à partir des états de la partie I vont vers des états des parties II ou III.
Si les parties II et III ont été créées, alors on peut affirmer que les suffixes reconnus à partir des états p1 ou p2 (notons $X_{II,-}$ cet ensemble de suffixes) diffèrent de ceux reconnus à partir de p3 ou p4 (notons l'ensemble de suffixes $X_{III,-}$). On a $X_{II,-} \neq X_{III,-}$.
Si l'on compare q1 et q3, et que l'on ne considère que le symbole courant 'a'; on a :
$X_{q1,-} = aX_{II,-}$ et $X_{q3,-} = aX_{III,-}$
Puisque $X_{II,-} \neq X_{III,-}$, on en déduit que :
$X_{q1,-} \neq X_{q3,-}$
Les deux états ne permettant pas de reconnaître les mêmes suffixes, ils doivent être séparés.
De même pour q2 et q3.
Par contre, rien ne nous prouve pour l'instant que les suffixes reconnus à partir de q1 et q2 sont différents, puisqu'ils sont de la forme $aX_{II,-}$. Ils vont donc rester dans la même partie.

La partie I est alors divisée en deux.



Ce mécanisme de séparation se poursuit de façon itérative, jusqu'à ce qu'aucune des parties ne doive plus être divisée.
A noter qu'un singleton, partie contenant un seul état, ne peut plus être séparée, et peut donc être "oubliée" jusqu'à la fin du processus, moment où l'on construit la table de transitions.

Algorithme

Nous décrivons ici "formellement" la procédure de minimisation.
La mise en œuvre plus "pragmatique" de cette procédure sous forme de tableaux est illustrée ici.

Soit $<\mathcal{A},\mathcal{Q},\mathcal{I},\mathcal{T},\mathcal{E}>$ un automate déterministe, complet, émondé,
et $<\mathcal{A},\mathcal{Q}',\mathcal{I}',\mathcal{T}',\mathcal{E}'>$ l'automate minimal équivalent.

- $\Theta = \{ \mathcal{T}, \mathcal{Q} \setminus \mathcal{T} \}$
Partition initiale : les terminaux et les autres.
Remarque : bien que l'état "poubelle", s'il a été ajouté, peut très souvent se retrouver seul dans une partie en fin de minimisation, il ne doit pas être séparé des autres non terminaux dans la partition initiale.

- FAIRE
Soit $\Theta = \{ \theta_j , 1 \leq j \leq n \}$
Le nombre de partitions varie d'une itération à une autre. Il ne peut bien entendu pas être supérieur au nombre d'états de l'automate non minimal (au maximum, chaque partie contient un et un seul état, et il y a donc une partie par état).
Il n'y a pas de nécessité d'avoir une relation entre la numérotation des parties pour deux partitions différentes.
$\forall\ 1 \leq j \leq n$,

On analyse chaque partie, pour éventuellement la scinder en plusieurs parties.
Le traitement peut être fait sur toutes les parties, mais il est bien entendu inutile sur celles ne contenant qu'un seul état.

Diviser θ_j en k parties $\theta_{j,k}$ telles que :

$(\forall p_1 \in \theta_{j,k} , \forall p_2 \in \theta_{j,k'} , k \neq k') \Leftrightarrow (\exists x \in \mathcal{A} \mid (p_1,x,p_2) \in \mathcal{E}, (p_2,x,q_2) \in \mathcal{E}, q_1 \in \theta_r, q_2 \in \theta_s, r \neq s)$

Deux états p_1 et p_2 de θ_j sont séparés dans $\theta_{j,k}$ et $\theta_{j,k'}$ si et seulement s'il existe au moins un symbole x tel que les états cibles q_1 et q_2 des transitions (p_1,x,q_1) et (p_2,x,q_2) sont dans des parties θ_r et θ_s différentes.
S'il n'y a pas lieu de diviser une partie, elle reste inchangée.

$\Theta \leftarrow \{ \theta_{j,k} \}$

La nouvelle partition est définie par l'ensemble des parties résultant des divisions éventuelles

TANT QUE Θ a été modifié

- $\mathcal{Q}' = \Theta$

Les états sont les parties issues de la dernière étape de division

$\mathcal{I}' = \{ \theta_i \}, \mathcal{I} \subseteq \theta_i$

L'état initial est la partie contenant l'état initial de l'automate à minimiser.

$\mathcal{T}' = \{ \theta_i \}, T \cap \theta_i \neq \varnothing$

Les états terminaux sont les parties contenant des états terminaux de l'automate à minimiser.
A noter que, suite à la première partition, ces parties ne contiennent que des états terminaux de l'automate à minimiser.

$\mathcal{E}' = \{ (\theta_i , x , \theta_j) \mid \forall p \in \theta_i, \forall q \in \theta_j, (p,x,q) \in \mathcal{E} \}$

Les transitions sont ré-écrites en terme de parties, en prenant comme "modèle" un état de chacune d'entre-elles.

Illustrations

Soit $<\mathcal{A},\mathcal{Q},\mathcal{I},\mathcal{T},\mathcal{E}>$ un automate déterministe, complet, accessible.
Soit $<\mathcal{A},\mathcal{Q}',\mathcal{I}',\mathcal{T}',\mathcal{E}'>$ l'automate minimal équivalent.

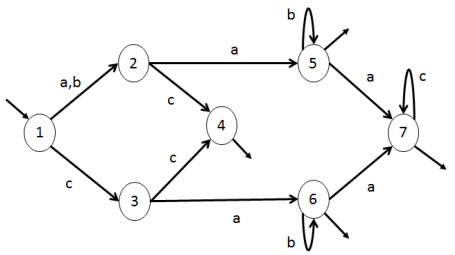
Algorithme (rappel) :

- $\Theta = \{ \mathcal{T} , \mathcal{Q} \setminus \mathcal{T} \}$
- FAIRE
  Soit $\Theta = \{ \theta_j , 1 \leq j \leq n \}$
  $\forall 1 \leq j \leq n,$
  Diviser θ_j en k parties $\theta_{j,k}$ telles que :
  $(\forall p_1 \in \theta_{j,k} , \forall p_2 \in \theta_{j,k'} , k \neq k') \Leftrightarrow (\exists x \in \mathcal{A} \mid (p_1,x,p_2) \in \mathcal{E}, (p_2,x,q_2) \in \mathcal{E}, q_1 \in \theta_r, q_2 \in \theta_s, r \neq s)$
  $\Theta \leftarrow \{ \theta_{j,k} \}$
  TANT QUE Θ a été modifié
- $\mathcal{Q}' = \Theta$
  $\mathcal{I}' = \{ \theta_i \}, \mathcal{I} \subseteq \theta_i$
  $\mathcal{T}' = \{ \theta_i \}, T \cap \theta_i \neq \varnothing$
  $\mathcal{E}' = \{ (\theta_i , x , \theta_j) \mid \forall p \in \theta_i, \forall q \in \theta_j, (p,x,q) \in \mathcal{E} \}$

Exemple #1

Automate déterministe :

Automate déterministe complet correspondant :



		a	b	c
→	1	2	2	3
	2	5	8	4
	3	6	8	4
←	4	8	8	8
←	5	7	5	8
←	6	7	6	8
←	7	8	8	7
	8	8	8	8

L'état '8' est l'état "poubelle" ajouté.

Partition initiale :

- $\Theta = \{ \mathcal{T} , \mathcal{Q} \setminus \mathcal{T} \}$

On identifie les 2 partitions, numérotées 'I' (non terminaux) et 'II' (terminaux) :

		a	b	c	Θ
→	1	2	2	3	I
	2	5	8	4	I
	3	6	8	4	I
←	4	8	8	8	II
←	5	7	5	8	II
←	6	7	6	8	II
←	7	8	8	7	II
	8	8	8	8	I

La partition initiale est donc : $\Theta = \{ I , II \} = \{ \{ 1 , 2 , 3 , 8 \} , \{ 4 , 5 , 6 , 7 \} \}$.
Pour une lecture plus facile, on peut ré-ordonner les lignes de la table de transitions :

		a	b	c	Θ
→	1	2	2	3	I
	2	5	8	4	I
	3	6	8	4	I
	8	8	8	8	I
←	4	8	8	8	II
←	5	7	5	8	II
←	6	7	6	8	II
←	7	8	8	7	II

Remarque : Un ordinateur peut faire de tri rapidement ; le faire "à la main" va être assez lourd car il est potentiellement à faire après chaque étape...

Itérations successives de division des parties :

- FAIRE
  Soit $\Theta = \{ \theta_j , 1 \leq j \leq n \}$
  $\forall 1 \leq j \leq n,$
  Diviser θ_j en k parties $\theta_{j,k}$ telles que :
  $(\forall p_1 \in \theta_{j,k} , \forall p_2 \in \theta_{j,k'} , k \neq k') \Leftrightarrow (\exists x \in \mathcal{A} \mid (p_1,x,p_2) \in \mathcal{E}, (p_2,x,q_2) \in \mathcal{E}, q_1 \in \theta_r, q_2 \in \theta_s, r \neq s)$
  $\Theta \leftarrow \{ \theta_{j,k} \}$
  TANT QUE Θ a été modifié

Itération #1

Concrètement, on duplique la table de transition (les colonnes des symboles de l'alphabet) en indiquant les parties d'appartenance à la place des états "cible" des transitions :

		a	b	c	Θ	a	b	c
→	1	2	2	3	I	I	I	I
	2	5	8	4	I	II	I	II
	3	6	8	4	I	II	I	II
	8	8	8	8	I	I	I	I
←	4	8	8	8	II	I	I	I
←	5	7	5	8	II	II	II	I
←	6	7	6	8	II	II	II	I
←	7	8	8	7	II	I	I	II

Par exemple :
- dans l'automate initial, 2 a pour états cibles les états 5 (pour le symbole 'a'), 8 ('b') et 4 ('c') qui sont respectivement dans les parties II, I et II (colonne "Θ").
- dans la ligne correspondant à l'état 2, après la colonne "Θ", on note donc II, I et II dans les dernières colonnes étiquetées avec les symboles de l'alphabet.
On fait de même pour tous les états.

Pour la partie I, on remarque que les parties cibles "I/I/I" sont communes aux états 1 et 8, alors que les parties cibles sont "II/I/II" pour les états 2 et 3.
La partie I est donc scindée en conséquence : les états correspondants sont "séparés" (voir introduction et équivalence de Nérode).
De même, la partie II est scindée en 3 parties distinctes.

Attention :
Les parties cibles des états 1, 8 et 4 sont les mêmes. Cependant, l'état 4 a déjà été dissocié des états 1 et 8 (il n'est plus dans la même partie). Il ne faut donc absolument pas recréer une partie qui contiendrait les états 1, 4 et 8 !
Il n'y a dans le processus de minimisation que des scissions de parties, jamais de regroupement !!

On obtient ainsi une nouvelle partition, indiquée dans la dernière colonne du tableau suivant :

		a	b	c	Θ0	a	b	c	Θ1
→	1	2	2	3	I	I	I	I	I.1
	2	5	8	4	I	II	I	II	I.2
	3	6	8	4	I	II	I	II	I.2
	8	8	8	8	I	I	I	I	I.1
←	4	8	8	8	II	I	I	I	II.1
←	5	7	5	8	II	II	II	I	II.2
←	6	7	6	8	II	II	II	I	II.2
←	7	8	8	7	II	I	I	II	II.3

On a donc :

Partition initiale : $\Theta_0 = \{ \{ 1 , 2 , 3 , 8 \} , \{ 4 , 5 , 6 , 7 \} \}$

Partition en fin d'itération 1 : $\Theta_1 = \{ \{ 1 , 8 \} , \{ 2 , 3 \} , \{4 \} , \{ 5 , 6 \} , \{ 7 \} \}$

La table des transitions peut alors être ré-ordonnée en fonction de la nouvelle partition, et les parties peuvent également être renumérotées.
On renumérote ici les parties, ce qui n'est pas une nécessité.

		a	b	c	Θ1		
→	1	2	2	3	I.1	I	
	8	8	8	8	I.1	I	
	2	5	8	4	I.2	II	
	3	6	8	4	I.2	II	
←	4	8	8	8	II.1	III	
←	5	7	5	8	II.2	IV	
←	6	7	6	8	II.2	IV	
←	7	8	8	7	II.3	V	

Itération #2

On procède de même, en ne prenant en compte que les parties qui peuvent effectivement être scindées (i.e. on ignore les singletons III et V).

		a	b	c	Θ1	a	b	c	Θ2
→	1	2	2	3	I	II	II	II	I.1
	8	8	8	8	I	I	I	I	I.2
	2	5	8	4	II	IV	I	III	II
	3	6	8	4	II	IV	I	III	II
←	4	8	8	8	III				III
←	5	7	5	8	IV	V	IV	I	IV
←	6	7	6	8	IV	V	IV	I	IV
←	7	8	8	7	V				V

La partie I est scindée, alors que les autres restent identiques.
$\Theta_1 = \{ \{ 1 , 8 \} , \{ 2 , 3 \} , \{4 \} , \{ 5 , 6 \} , \{ 7 \} \}$
$\Theta_2 = \{ \{ 1 \} , \{ 8 \} , \{ 2 , 3 \} , \{ 4 \} , \{ 5 , 6 \} , \{ 7 \} \}$
Renumérotons les parties, et passons à l'itération suivante.

Itération #3
Seules les parties III et V ne sont pas des singletons et doivent être analysées.

		a	b	c	Θ2			
→	1	2	2	3	I			
	8	8	8	8	II			
	2	5	8	4	III	V	II	IV
	3	6	8	4	III	V	II	IV
←	4	8	8	8	IV			
←	5	7	5	8	V	VI	V	II
←	6	7	6	8	V	VI	V	II
←	7	8	8	7	VI			

A ce stade, aucune partie n'est scindée. On arrête donc le processus avec la partition courante.

Construction de l'automate minimal :

- $\mathcal{Q}' = \Theta$

  $\mathcal{I}' = \{ \theta_i \}, \mathcal{I} \subseteq \theta_i$

  $\mathcal{T}' = \{ \theta_i \}, T \cap \theta_i \neq \varnothing$

  $\mathcal{E}' = \{ (\theta_i , x , \theta_j) \mid \forall p \in \theta_i, \forall q \in \theta_j, (p,x,q) \in \mathcal{E} \}$

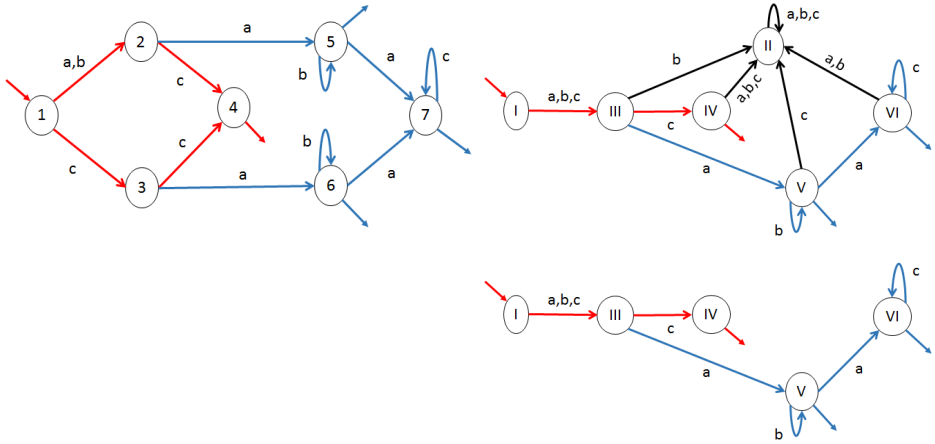
L'automate minimal est construit à partir des transitions écrites en termes de parties. Dans le tableau obtenu à la dernière itération, on ne garde qu'une ligne par partie, et on complète les lignes correspondant aux singletons qui avaient été mis de côté. Les lignes associées aux états 2 et 3 sont fusionnées, ainsi que celles associées à 5 et 6.

		a	b	c	Θ	a	b	c
→	1	2	2	3	I	III	III	III
	8	8	8	8	II	II	II	II
	2	5	8	4	III	V	II	IV
	3	6	8	4				
←	4	8	8	8	IV	II	II	II
←	5	7	5	8	V	VI	V	II
←	6	7	6	8				
←	7	8	8	7	VI	II	II	VI

En ne gardant que les colonnes écrites en terme de "parties", on obtient la table de transition finale :

		a	b	c
→	I	III	III	III
	II	II	II	II
	III	V	II	IV
←	IV	II	II	II
←	V	VI	V	II
←	VI	II	II	VI

Pour comparaison, on trace ci-dessous les graphes d'états des deux automates : le déterministe (non complet) et le minimal. A vous de vous assurer que le langage reconnu est bien le même (les couleurs peuvent vous y aider...). Pour une comparaison encore plus facile, le schéma du minimal non complet (élimination de l'état poubelle) est également donné.



Remarque sur la présentation des traitements

La présentation utilisée ici n'est bien entendu qu'un exemple. Beaucoup de "styles" sont envisageables.
On peut par exemple construire un seul tableau, dans lequel l'ordre des lignes n'est pas changé.
On peut également utiliser d'autres systèmes de numérotation des parties. Par exemple, pour la partition initiale, on peut noter 'N' l'ensemble des états non terminaux, et 'T' l'ensemble des états terminaux, puis continuer la numérotation par niveau : N.1, T.2.1, ...

Par exemple (on ajoute un niveau de numérotation à chaque itération, si nécessaire) :

		a	b	c	Θ_0	a	b	c	Θ_1	a	b	c	Θ_2	a	b	c
→	1	2	2	3	I	I	I	I	I.I	I.II	I.II	I.II	I.I.I			
	2	5	8	4	I	II	I	II	I.II	II.II	I.I	II.I	I.II	II.II	I.I.II	II.I
	3	6	8	4	I	II	I	II	I.II	II.II	I.I	II.I	I.II	II.II	I.I.II	II.I
←	4	8	8	8	II	I	I	I	II.I				II.I			
←	5	7	5	8	II	II	II	I	II.II	II.III	II.II	I.I	II.II	II.III	II.II	I.I.II
←	6	7	6	8	II	II	II	I	II.II	II.III	II.II	I.I	II.II	II.III	II.II	I.I.II
←	7	8	8	7	II	I	I	II	II.III				II.III			
	8	8	8	8	I	I	I	I	I.I	I.I	I.I	I.I	I.I.II			

Aucune séparation n'est faite à partir de Θ_2. On construit donc la table de transitions finale :

		a	b	c
→	I.I.I	I.II	I.II	I.II
	I.II	II.II	I.I.II	II.I
←	II.I	I.I.II	I.I.II	I.I.II
←	II.II	II.III	II.II	I.I.II
←	II.III	I.I.II	I.I.II	II.III
	I.I.II	I.I.II	I.I.II	I.I.II

Autres exemples (cas particuliers)

Automate déjà minimal

Si l'automate est déjà minimal, on doit aboutir à une situation où il y a autant de parties (les états de l'automate minimal calculé) que d'états dans l'automate initial.
Le traitement peut s'arrêter lorsque cette situation est reconnue :
- autant de parties que d'état dans l'automate de départ,
- ou bien toutes les parties contiennent un et un seul état de l'automate de départ.

Il est également possible de ne pas tester cette condition d'arrêt dans la mise en œuvre de la minimisation. En effet, si celle-ci est rencontrée à l'itération 'i', alors, puisque aucune partie ne peut plus être découpée (ce sont toutes des singletons), alors on aura exactement la même partition à la fin de l'itération 'i+1'.

...exemple à venir...

Automate reconnaissant le langage A*

Si l'automate déterministe complet à minimiser est tel que tous ses états sont des états terminaux, alors tous les mots sont reconnus.
Lors de la minimisation, une seule partie (contenant tous les états terminaux, donc tous les états) sera définie dans la partition initiale. Le calcul des parties cible de la première itération identifiera donc toujours cette partie unique. Aucune séparation ne sera faite.
En résultat : une seule partie. L'automate minimal contiendra donc un seul état.

...exemple à venir...

RELATION
AUTOMATE / EXPRESSION RATIONNELLE

Nous voyons ici des méthodes de passage entre les 2 représentations d'un langage : expression rationnelle et automate.

Expression Rationnelle vers Automate Asynchrone

Soit X une expression rationnelle.
Quel est l'automate permettant de reconnaître les mots représentés par X ?

On peut bien entendu essayer intuitivement, ou empiriquement, de construire cet automate. Nous nous intéressons ici à une technique systématique pouvant être mise en œuvre dans un système informatique.

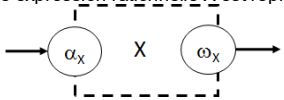
Règles de construction

Principe de base :

Afin de permettre toutes les combinaisons possibles, chaque automate construit correspondant à une expression rationnelle satisfait les propriétés suivantes :
- il dispose d'un état initial unique n'acceptant aucune transition "entrante" ;
- il dispose d'un état terminal unique n'acceptant aucune transition "sortante".

Moyennant ces propriétés, les règles décrites ci-dessous peuvent être combinées afin de construire un automate. Ceci a toute son importance : les règles de constructions marchent toujours, même si elles paraissent compliquées. En d'autres termes : une expression rationnelle peut souvent conduire à un automate plus simple que présenté ici, mais la combinaison de plusieurs automates "simplifiés" peut conduire à des effets désastreux (on ne reconnaît pas le langage correspondant à l'expression rationnelle).

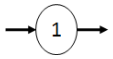
Pour les règles de construction combinant plusieurs expressions rationnelles, par exemple X=YZ, où X, Y et Z sont des expressions rationnelles, nous utiliserons la représentation suivante :
une expression rationnelle X est représentée schématiquement par :



avec α_X et ω_X ses états initial et terminal.
Rappel : Dans la partie de l'automate "à l'intérieur" de la zone marquée 'X', il n'y a aucune transition aboutissant à l'état α_X, et aucune transition partant de l'état ω_X.

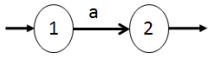
ε (mot vide)

Une construction extrêmement simple suffit. L'automate ci-dessous reconnaît bien uniquement le mot vide.



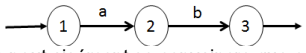
a (un symbole)

'a' est un symbole de l'alphabet. On reconnaît le mot 'a' avec l'automate :



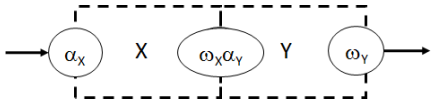
XY (concaténation)

Si X=a et Y=b, on a bien évidemment une représentation de XY sous la forme suivante :



On peut aisément se convaincre que cette construction revient à "fusionner" l'état terminal de X avec l'état initial de Y.

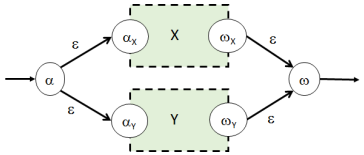
Généralisons pour X et Y deux expressions rationnelles quelconques, reconnues chacune par un automate. L'automate correspondant à la concaténation XY de ces deux expressions rationnelles se construit de la façon suivante, en fusionnant les états ω_X et α_Y :



Au final, pour reconnaître une chaîne de forme XY, on commence par reconnaître son début X en passant de l'état α_X à l'état $\omega_X\alpha_Y$. Ensuite, on reconnaît Y en passant de cet état à ω_Y.

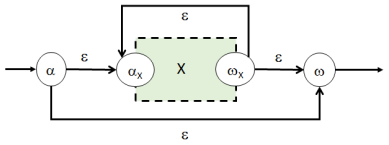
X+Y (choix)

On créé ici un "aiguillage", en utilisant des transitions epsilon, permettant d'abord de "rentrer" dans l'automate correspondant à X ou à Y, puis d'en sortir. On respecte bien évidemment les propriétés liées à l'état initial et l'état final de la construction.



X (répétition)*

La construction à appliquer est la suivante :

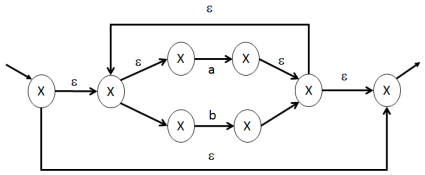


On note qu'il n'y a pas de transition entrante sur l'état α, ni de transition entrante sur l'état ω.

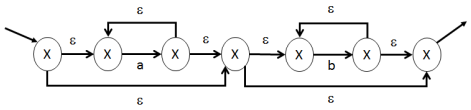
Illustrations

Dans les exemples donnés ci-dessous, les états ne sont pas numérotés distinctement. Par simplification, et parce que cela n'a pas vraiment d'importance ici.

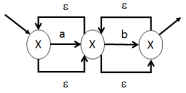
(a+b)*



a*b*

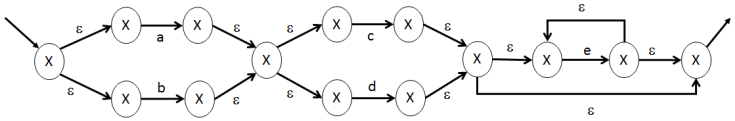


Remarque :
A vous de vous convaincre que la solution suivante ne marche pas car la mise en étoile (de 'a' et de 'b') n'est pas mise en œuvre correctement :

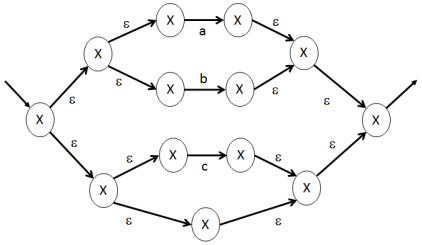


...on peut mélanger les 'a' et les 'b', alors que l'expression 'a*b*' ne le permet pas.

(a+b)(c+d)e*



(a+b)+(c+ε)



Automate vers Expression Rationnelle

A l'inverse de la section précédente : comment peut-on déduire l'expression rationnelle correspondant à un langage lorqu'on dispose de l'automate associé ?

A noter que l'automate de départ peut être asynchrone ou non.

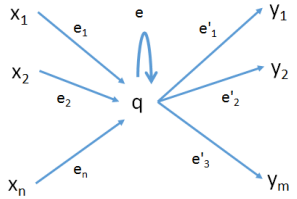
Nous présentons ici deux méthodes distinctes :
- réduction d'un automate par élimination d'états ;
- résolution d'un système d'équations utilisant les expressions rationnelles.

Par élimination d'états

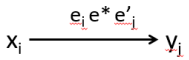
Cette technique est dérivée de l'algorithme de Mac Naughton et Yamada.

Principe d'élimination d'état

Soit un état q ayant des transitions "entrantes" et des transitions "sortantes" :



Si l'on supprime l'état q, il faut alors ajouter les transitions permettant de passer des états x_i vers les états y_j.
Pour ce faire, pour chaque couple x_i / y_j, on ajoute la transition (x_i , e_i e* e'_j , y_j) :



L'étiquette de cette transition n'est donc plus un symbole de l'alphabet, mais une expression rationnelle.

Par exemple, pour aller de x_1 à y_2 en passant par q, il faut bien d'abord lire un mot de la forme e_1 pour aller dans l'état q, puis autant de fois que l'on veut (boucle sur q) un mot de la forme e, puis un mot de la forme e'_2 pour aller de q à y_2. Soit en concaténant ces 3 parties : $e_1 e^* e'_2$.

L'état q, et toutes les transitions associées, peuvent alors être supprimés.

A noter que la technique décrite ici permet d'éliminer les états dans n'importe quel ordre. Les expressions rationnelles obtenues seront peut-être différentes (dans leur écriture), mais elle correspondront toutes au même langage.

Méthode

Soit un automate $< \mathcal{A} , \mathcal{Q} , \mathcal{I} , \mathcal{T} , \mathcal{E} >$ donc on veut déterminer l'expression rationnelle.

Soit l'automate $< \mathcal{A}' , \mathcal{Q}' , \{\alpha\} , \{\omega\} , \mathcal{E}' >$
$\mathcal{A}'$: l'ensemble des expressions rationnelles pouvant être construites à partir de l'alphabet $\mathcal{A}$

En fin de processus, cet automate n'aura plus qu'une seule transition (α,X,ω), avec X l'expression rationnelle recherchée.

- $\mathcal{Q}' = \mathcal{Q} \cup \{\alpha,\omega\}$
  $\mathcal{E}' = \mathcal{E} \cup \{ (\alpha,\varepsilon,i), i \in \mathcal{I} \} \cup \{ (t,\varepsilon,\omega), t \in \mathcal{T} \}$
  Initialisation :
  On créé un état initial α et un état terminal ω, et les transitions epsilon les reliant aux états initiaux et terminaux de l'automate de départ.
  Les états de $\mathcal{I}$ et $\mathcal{T}$ ne sont plus ni initiaux, ni terminaux.

- $\forall e \in \mathcal{Q}$,
  Traitement itératif :
  Tous les états de l'automate de départ doivent être traités (i.e. éliminés), dans n'importe quel ordre.
  - $\forall (p,x,e) \in \mathcal{E}', \forall (e,y,q) \in \mathcal{E}', p \neq e, q \neq e,$
    On traite tout couple transition "entrante" (p,x,e) / transition "sortante" (e,y,q) sur e, et on ajoute les transitions menant de p à q.
    Attention, on peut avoir p=q, ce qui entraînera la création d'une boucle sur p.
    On peut bien entendu aussi avoir x=y.
    $\exists (e,z,e) \in \mathcal{E}' \Rightarrow \mathcal{E}' \leftarrow \mathcal{E}' \cup \{ (p , xz^*y , q) \}$
    Si il existe une boucle sur l'état e, c'est à dire une transition (e,z,e) :
    *pour aller de p à q, on doit d'abord lire un 'x' pour aller dans l'état e, puis autant de 'z' que l'on veut (éventuellement aucun) tout en restant dans l'état e, puis lire un 'y' pour aller dans l'état q. Soit au total une chaîne de la forme " xz*y ".*

On ajoute donc la transition correspondant allant de p à q.

$\neg(\exists\ (e,z,e) \in \mathcal{E}') \Rightarrow \mathcal{E}' \leftarrow \mathcal{E}' \cup \{ (\ p\ ,\ xy\ ,\ q\ ) \}$

S'il n'y a pas de transition (e,z,e), passer de p à q se fait simplement en lisant la chaîne " xy ".

La transition est ajoutée.

- $\mathcal{E}' \leftarrow \mathcal{E}' \setminus (\{ (p,x,e) \} \cup \{ (e,y,q) \})$
  $\mathcal{Q}' \leftarrow \mathcal{Q}' \setminus \{ e \}$

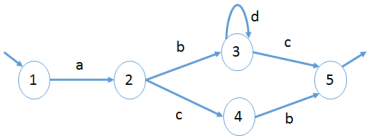
  Tous les couples transition entrante / transition sortante de e ont été dupliqués par de nouvelles transitions. Elles peuvent donc être supprimées.
  L'état e est également supprimé.

- $< \mathcal{A}', \{\alpha,\omega\}, \{\alpha\}, \{\omega\}, \{ (\alpha,L,\omega) \} >$

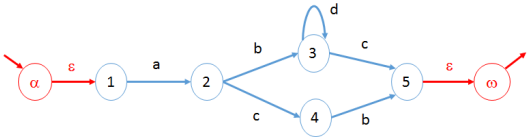
  On obtient finalement un automate à 2 états : un initial et un terminal. L'étiquette L de la transition qui les relie est l'expression rationnelle du langage.

Illustration 1

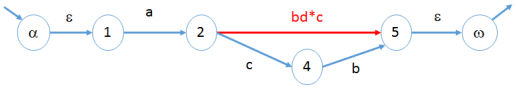
Automate de départ :

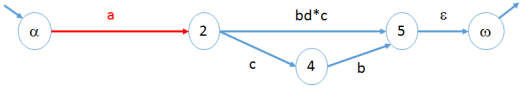


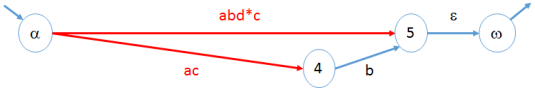
Ajout des états initial / terminal :

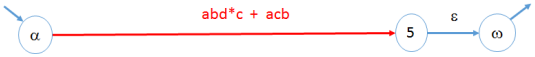


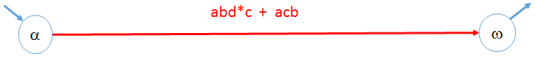
Suppressions successives des différents états :











Le langage reconnu est donc décrit par l'expression rationnelle : abd*c+acb

Système d'équations

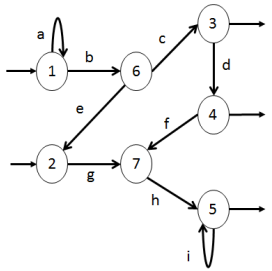
Il s'agit ici de développer un système d'équations utilisant les expressions rationnelles, afin de déterminer celle correspondant au langage reconnu par un automate.

Principe général

Notons $X_{p,q}$ l'expression rationnelle décrivant tous les mots faisant passer l'état courant de p à q.
Si un automate ne contient qu'un état initial i et un état terminal t, le langage reconnu par l'automate correspond bien évidemment à l'expression rationnelle $X_{i,t}$.
Si l'automate contient plusieurs états initiaux et/ou plusieurs états terminaux, le langage reconnu peut être décrit par une expression rationnelle qui est la somme (l'union) de toutes les expressions de la forme $X_{i,t}$, avec i un état initial et t un état terminal.

Par exemple, dans l'automate ci-contre, l'expression rationnelle décrivant le langage est :

$L = X_{1,3} + X_{1,4} + X_{1,5} + X_{2,3} + X_{2,4} + X_{2,5}$



Pour déterminer le langage reconnu par l'automate, il nous faut donc écrire L, c'est à dire $X_{1,3}$, $X_{1,4}$, $X_{1,5}$, $X_{2,3}$, $X_{2,4}$ et $X_{2,5}$ en n'utilisant que les symboles de l'alphabet (plus éventuellement le ε) et les opérateurs permettant de combiner les expressions rationnelles.

Décomposition d'une équation

L'objectif est de remplacer progressivement les formes "$X_{p,q}$" par des expressions n'utilisant que des symboles de l'alphabet.
Pour ce faire, nous devons procéder à plusieurs décompositions successives, tout en utilisant d'autres règles de ré-écriture telles que le Lemme d'Arden.

Deux décompositions sont possibles, selon que l'on s'intéresse au début (préfixe / facteur gauche) ou à la fin (suffixe / facteur droite) de l'expression $X_{p,q}$.

Prenons le premier exemple suivant :

$X_{p,q} = a\ X_{p_2,q}$ $\qquad$ $X_{p,q} = X_{p,p_{n-1}}\ b$

On voit
- à gauche la décomposition de $X_{p,q}$ selon son préfixe :
  reconnaître $X_{p,q}$, c'est d'abord reconnaître 'a' puis reconnaître une suite de symboles permettant de passer de l'état p_2 à l'état q ;
- et à droite sa décomposition selon son suffixe :
  reconnaître $X_{p,q}$, c'est d'abord reconnaître une suite de symboles permettant de passer de l'état p à l'état p_{n-1}, puis reconnaître 'b'.

Si l'on ajoute maintenant des transitions (boucles) sur les états p et q :

$X_{p,q} = c\,X_{p,q} + a\,X_{p_2,q}$ $\quad$ $X_{p,q} = X_{p,q}\,d + X_{p,p_{n-1}}\,b$

Reconnaître $X_{p,q}$ devient :
- à gauche :
  soit reconnaître 'c' suivi d'une suite de symboles permettant de passer de p à q,
  ou ("+") reconnaître 'a' suivi d'une suite de symboles permettant de passer de p_2 à q ;
- à droite :
  soit reconnaître une suite de symboles permettant de passer de p à q puis reconnaître 'd',
  ou reconnaître une suite de symboles permettant de passer de p à p_{n-1} puis reconnaître b.

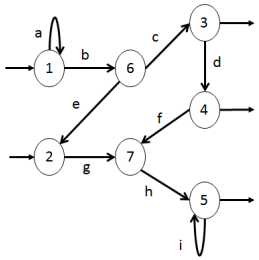
On note ici des écritures récursives de $X_{p,q}$. Récursivité que le lemme d'Arden nous permettra d'éliminer.

Ces deux visions (préfixe/facteur gauche ou suffixe/facteur droite) de la décomposition de $X_{p,q}$ conduisent aux 2 méthodes équivalentes décrites ci-dessous.
Ce ne sont finalement pas 2 méthodes, mais 2 façons d'utiliser la même méthode. Elles utilisent deux versions différentes du lemme d'Arden.

Les 2 méthodes

Puisqu'on s'intéresse aux expressions permettant de passer d'un état initial à un état terminal, nous allons pouvoir simplifier l'écriture "$X_{p,q}$", avec une interprétation liée à la méthode utilisée.

Ré-utilisons l'exemple précédent :



Méthode des suffixes / facteurs droite

Dans cette méthode, on s'attache à décomposer une expression en extrayant son suffixe, ou facteur droite.
On notera X_p l'expression rationnelle qui nous permet de passer d'un état initial, quel qu'il soit, à l'état p. L'état 'p' est l'arrivée de l'expression, c'est à dire l'état courant de l'automate après avoir analysé un mot correspondant à l'expression rationnelle, en partant d'un état initial quelconque.

Le langage de l'automate précédent est alors donné par l'expression : $L = X_{-3} + X_{-4} + X_{-5}$

Calculons donc ces 3 expressions X_{-3}, X_{-4} et X_{-5}, en utilisant la décomposition présentée précédemment :

$X_{-3} = X_{-6}\,c$	aller d'un état initial à l'état 6, puis reconnaître 'c'
$X_{-4} = X_{-3}\,d$	aller d'un état initial à l'état 3, puis reconnaître 'd'
$X_{-5} = X_{-7}\,h + X_{-5}\,i$	aller d'un état initial à l'état 7, puis reconnaître 'h', ou bien aller d'un état initial à l'état 5 lui-même, puis reconnaître 'i'

On remarque que pour décomposer une expression X_p, on regarde toutes les transitions entrantes sur l'état p, celles qui arrivent sur l'état p. Il y a autant d'élément dans la décomposition de X_p que de transitions entrantes sur l'état p.
Par exemple, il y a bien 2 transitions "entrantes" dans l'état 5 : (7,h,5) et (5,i,5).

Il faut ensuite définir :

$X_{-6} = X_{-1}\,b$

$X_{-7} = X_{-2}\,g + X_{-4}\,f$

puis

$X_{-1} = X_{-1}\,a + \varepsilon$
$X_{-2} = X_{-6}\,e + \varepsilon$

Un "epsilon" est ajouté aux définitions de X_{-1} et X_{-2} car ce sont des états initiaux, dont pour aller d'un état initial à ces états, on peut n'avoir rien reconnu, ce qui équivaut à avoir reconnu le mot "epsilon" !

La définition de X_{-1} est récursive. On peut lui appliquer le lemme d'Arden pour obtenir : $X_{-1} = \varepsilon a^* = a^*$
On fait la même chose sur X_{-5}.

$X_{-3} = X_{-6}\,c$
$X_{-4} = X_{-3}\,d$
$X_{-5} = X_{-7}\,h\,i^*$
$X_{-6} = X_{-1}\,b$
$X_{-7} = X_{-2}\,g + X_{-4}\,f$
$X_{-1} = a^*$
$X_{-2} = X_{-6}\,e + \varepsilon$

On remplace X_{-1} par sa valeur dans X_{-6}, puis X_{-6} dans X_{-3} :

$X_{-3} = a^*\,b\,c$
$X_{-4} = X_{-3}\,d$
$X_{-5} = X_{-7}\,h\,i^*$
$X_{-6} = a^*\,b$
$X_{-7} = X_{-2}\,g + X_{-4}\,f$
$X_{-1} = a^*$
$X_{-2} = X_{-6}\,e + \varepsilon$

On remplace X_{-3} dans X_{-4} et X_{-6} dans X_{-2} :

$X_{-3} = a^*\,b\,c$
$X_{-4} = a^*\,b\,c\,d$
$X_{-5} = X_{-7}\,h\,i^*$
$X_{-6} = a^*\,b$
$X_{-7} = X_{-2}\,g + X_{-4}\,f$
$X_{-1} = a^*$
$X_{-2} = a^*\,b\,e + \varepsilon$

X_{-2} et X_{-4} sont remplacés dans X_{-7} :

$X_{-3} = a^*\,b\,c$
$X_{-4} = a^*\,b\,c\,d$
$X_{-5} = X_{-7}\,h\,i^*$
$X_{-6} = a^*\,b$
$X_{-7} = (\,a^*\,b\,e + \varepsilon\,)\,g + a^*\,b\,c\,d\,f$
$X_{-1} = a^*$
$X_{-2} = a^*\,b\,e + \varepsilon$

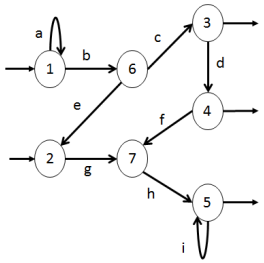
Finalement, X_{-7} est remplacé par sa valeur dans X_{-5} :

$X_{-3} = a^*\,b\,c$
$X_{-4} = a^*\,b\,c\,d$
$X_{-5} = [\,(\,a^*\,b\,e + \varepsilon\,)\,g + a^*\,b\,c\,d\,f\,]\,h\,i^*$
$X_{-6} = a^*\,b$
$X_{-7} = (\,a^*\,b\,e + \varepsilon\,)\,g + a^*\,b\,c\,d\,f$
$X_{-1} = a^*$
$X_{-2} = a^*\,b\,e + \varepsilon$

Le langage reconnu est $L = X_{-3} + X_{-4} + X_{-5}$
On a, après réécriture :

$L = a^*\,b\,c + a^*\,b\,c\,d + a^*\,b\,c\,d\,f\,h\,i^* + a^*\,b\,e\,g\,h\,i^* + g\,h\,i^*$

En comparant avec le graphe des états de l'automate, on peut aisément constater que ces 5 formes correspondent aux 5 suites possibles de transitions permettant de passer d'un état initial à un état terminal.



Méthode des préfixes / facteurs gauche

On note ici X_{p-} l'expression rationnelle décrivant les mots permettant de passer de l'état p à un état terminal, quel que soit cet état terminal. Le langage est déterminé par les expressions de ce type où p est un état initial.
En reprenant le même automate que précédemment, on a $L = X_{1-} + X_{2-}$

On initialise les équations :

$X_{1-} = a\ X_{1-} + b\ X_{6-}$
$X_{2-} = g\ X_{7-}$
$X_{3-} = d\ X_{4-} + \varepsilon$
$X_{4-} = f\ X_{7-} + \varepsilon$
$X_{5-} = i\ X_{5-} + \varepsilon$
$X_{6-} = c\ X_{3-} + e\ X_{2-}$
$X_{7-} = h\ X_{5-}$

On remarque le 'ε' ajouté à certaines égalités. Pour X_{3-}, par exemple, qui représente les mots permettant de passer de l'état 3 à un état terminal : l'état 3 étant lui-même un état terminal, on peut ne rien reconnaître, c'est-à-dire reconnaître le mot vide ε.
On procède comme dans la méthode précédente. Le lemme d'Arden est utilisé dans son autre forme.