Département de Mathématiques
Génie Mathématique et Informatique

Mini-Projet :
Minimisation des automates et algorithmes

Réalisé par : HANI Moad

Encadré par : Mr.Mustapha KABIL

Année universitaire 2016/2017

Remerciements

J'adresse mes sincères remerciements à mon professeur, Monsieur Mustapha Kabil, pour m'avoir offert ce projet, à l'occasion de ce travail, combien essentiel dans nos études et stages.

Il est aussi intéressant dans son sujet "Minimisation des automates et algorithmes" cherchant ainsi à gagner en la performance sans perdre en simplicité dans le traitement de problèmes spécifiques, tels le traitement de chaines de caractères ou la description de comportements dynamiques, et qui constituent aujourd'hui le levier de toute la technologie moderne.

Table des Matières

1. Introduction

En Informatique théorique, l'objectif de la théorie des automates est de proposer des modèles de mécanismes mathématiques qui formalisent les méthodes de calcul. Cette théorie est le fondement de plusieurs branches importantes de l'informatique théorique, comme :

- La calculabilité, par le modèle des machines de Turing.
- Les automates finis, et leurs variantes, qui sont utilisés dans l'analyse des langues naturelles, la traduction des programmes par les compilateurs, divers algorithmes de manipulation de textes comme les algorithmes de recherche de sous-chaine, ou la vérification automatique du fonctionnement de circuits logiques.
- La théorie de la complexité des algorithmes, visant à classifier les algorithmes en fonction des ressources temporelles et en mémoire nécessaires à leur exécution.

Les automates finis à états (automates finis en abrégé) offrent un formalisme de description peu puissant mais avec beaucoup d'algorithmes efficaces. Ils sont très utilisés notamment dans deux domaines : le traitement de chaînes de caractères et la description de comportement dynamique de systèmes.

2. Les automates minimales et Algorithme d

2.1 Définition

Deux automates finis sont équivalents si ils reconnaissent le même langage. Pour tout automate fini, un seul automate fini déterministe est minimal c'est-à-dire ayant un nombre minimal d'état qui est équivalent à l'automate donné.

2.2 Proposition

Etant donné un automate déterministe $A = (Q, \Sigma, \delta, q_0, F)$, on cherche a construire un automate déterministe équivalent contentent un nombre minimum d'état.

2.3 Algorithme de Moore: Minimisation d'un AFN

Certains AFDs possèdent un nombre d'états et de transitions important. États et transitions ne sont pas toujours indispensables. Typiquement, cette situation a lieu lorsque l'automate est construit à l'aide d'algorithmes automatiques comme dans le cas de la construction automatique díun AFN et de la transformation díun AFN en AFD. Afin d'améliorer la reconnaissance des chaînes et d'optimiser la place mémoire utilisée, il est intéressant d'essayer de minimiser un AFD. Une première approche de cette minimisation peut consister à étudier les caractéristiques des états et de supprimer ceux qui sont inutiles. Cependant, il est possible d'aller plus loin en garantissant que l'AFD obtenu est un AFD minimal. Ceci est ' de la méthode que l'on présente ici. Il existe de nombreuses stratégies pour minimiser un AFD . on trouvera aussi la formalisation et la caractérisation des classes d'équivalence. L'algorithme de Moore présenté ici est un algorithme, basé sur les classes d'équivalence d'états.

L'algorithme de minimisation du nombre d'états d'un AFD fonctionne en déterminant tous les groupes d'états qui peuvent être distingués par une chaîne d'entrée. Chaque groupe d'états indistinguables est alors fusionné en un état unique. L'algorithme travaille en mémorisant et en raffinant une partition de l'ensemble des états. Chaque groupe d'états à l'intérieur de la partition correspond aux états qui n'ont pas encore été distingués les uns des autres. Toute paire d'états extraits de différents groupes a été prouvée "distinguable" par une chaîne.

Initialement, la partition consiste à former deux groupes : les états d'acceptation et les autres. L'étape fondamentale prend un groupe d'états et un symbole puis étudie les transitions de ces états sur ce symbole. Si ces transitions conduisent à des états qui tombent dans au moins deux groupes différents de la partition courante, alors on doit diviser ce groupe. La division est effectuée avec pour objectif que les transitions depuis chaque sous-groupe soient confinées à un seul groupe de la partition courante. Ce processus de division est répété jusqu'à ce qu'aucun groupe n'ait plus besoin d'être divisé.

Les états du nouvel automate sont donnés par un représentant de chaque groupe. Les états finaux sont les représentants des groupes possédant un état final de l'AFD de départ. L'état de départ est le représentant du groupe possédant l'état de départ de l'AFD initial. Les transitions sont construites avec les transitions de chaque état représentant de groupe vers un état représentant d'un autre groupe si cet état engendre une transition vers un élément du groupe.

Sur les états crées, il faut supprimer les états stériles et les états non-accessibles depuis l'état initial.

Remarque :

Cette méthode de minimisation (comme pour la méthode de déterminisation) est applicable par la manipulation d'une table de transitions. La première ligne donne pour chacun des états son appartenance à un des deux ensembles de départ. Ensuite, pour chaque état et chaque symbole on détermine vers quel ensemble nous mène la transition correspondante. Une fois tous les couples étudiés, on divise les ensembles de départ en ensembles dont les états se comportent de la même manière. On recommence jusqu'à ce qu'il n'y ait plus de division d'ensemble. L'exemple suivant illustre cette manière de faire.

Théorème de Myhill-Nérode

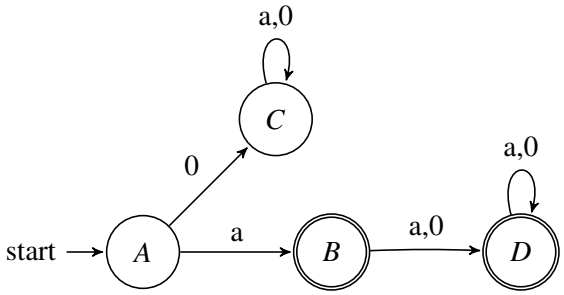
Soit L un langage rationnel. Parmi tous les AFD reconnaissant L, il en existe un et un seul qui a un nombre minimal d'états.

Démonstration

On vérifie par récurrence que $\cong \subseteq \cong_i$pour tout i≥ 0 $\cong \subseteq \cong_0$ est vérifie puisque L_q=$L_{q'}$ implique q$\in$F ssi $q' \in$F puis L_q=$L_{q'}$ implique q$\cong_i q'$par hypothése de récurrence et pour tout a de V $L_{\delta(q,a)}$=$L_{\delta(q',a)}$
donc encore par l'hypothése de récurrence δ(q,a)$\cong_i \delta(q',a)$
Notons que $\cong_{i+1} \subseteq \cong_i$pour tout i donc $\cong_{i+1}$ est d'index supérieur à celui de $\cong_i$ mais d'index inférieur à celui de $\cong$ qui est le nombre (fini) d'états de l'automate minimal.Donc il existe k tel que $\cong_k$=$\cong_{k+1}$=$\cong_{k+j}$ pour tout j≥ 0. Il reste à montrer que $\cong_k \subseteq \cong$.soient q et q$'$ deux états de Q tels que q$\cong_k$ q$'$ et w=$a1.....a_n$ dans L_q .Alors il existe deux exécution dans A
$q \rightarrow^{a_1} q_1 \rightarrow^{a_2} q_2$ ..... $\rightarrow^{a_n} q_n$
$q' \rightarrow^{a_1} q'_1 \rightarrow^{a_2} q'_2$..... $\rightarrow^{a_n} q'_n$
avec $q_j \cong_i q'_j$ pour j ≥ 1 et $q_n \in$ F ssi q'_n donc w est aussi dans $L_{q'}$.Cela montre $L_q \subseteq L_{q'}$ et on démontre de meme l'inclusion inverse .

2.4 Exemple d'application

Appliquons L'algorithme de Moore sur l'automate suivante:



	A	B	C	D
Initialisation	I	II	I	II
a				
0				
Bilan				

Table 2.1:

En effet, A et C ne sont pas des états finaux. Ils sont donc dans le même ensemble (noté I). De même, B et C sont finaux. Ils sont donc dans le même ensemble (noté II). Ensuite, pour chaque couple (état, symbole), on regarde vers quel ensemble nous mène la transition de l'automate (si elle existe, sinon on ne met rien).

Puis on effectue la séparation des ensembles. Deux états sont dans un même ensemble s'ils étaient déjà dans le même ensemble et si les transitions mènent dans les mêmes ensembles. Dans cet exemple, B et D se comportent exactement de la même manière, donc ils restent ensemble. Par contre, A et C qui faisaient partie du même ensemble se comportent différemment sur le symbole "a". Par conséquent, l'ensemble noté I se divise en deux. Nous obtenons donc :

	A	B	C	D
Initialisation	I	II	I	II
a	II	II	I	II
0	I	II	I	II
Bilan				

Table 2.2:

	A	B	C	D
Initialisation	I	II	I	II
a	II	II	I	II
0	I	II	I	II
Bilan 1	I	II	III	II

Table 2.3:

	A	B	C	D
Initialisation	I	II	I	II
a	II	II	I	II
0	I	II	I	II
Bilan 1	I	II	III	II
a	II	II	III	II
0	III	II	III	II
Bilan 2	I	II	III	II

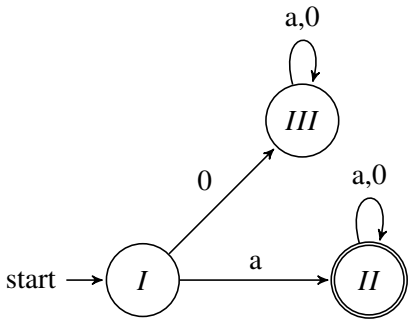
Table 2.4:

La situation courante (la ligne Bilan 1) est différente que la situation de départ. Donc il faut recommencer !

La ligne "Bilan 2" est identique à la ligne "Bilan 1". Par conséquent, dans chacun des ensembles restants, les états ne sont pas distinguables (ils se comportent exactement de la même manière et sont donc "redondants").

Nous pouvons donc construire le nouvel automate en associant un état à chaque ensemble. Les transitions sont indiquées par le tableau. L'état initial est celui contenant l'état initial de l'automate de départ : ici I contient l'état A, état initial. Les états finaux sont les états dont l'ensemble correspondant contient au moins un état final : ici, II contient B et D qui sont finaux.

Cependant, l'algorithme n'est pas encore terminé ! Pour l'instant, nous avons "fusionné" les états indistinguables. Il faut maintenant supprimer tous les états inutiles restant. Dans notre exemple, l'état III est un état stérile, donc inutile. Il faut donc le supprimer. D'où l'automate déterministe minimal de la figure suivante :



3. Lemme d'Arden

3.1 démonstartion du Lemme d'Arden

Soient U et V deux langages.

1.Quand $\varepsilon \notin U$, l'équation $X = UX + V$,où X est un langage inconnu, admet pour solution $X = U^*V$ et l'équation $X = XU + V$ admet pour solution $X = VU^*$.
2.Quand $\varepsilon \in U$, l'équation $X = UX + V$,admet pour solution $X = U^*(V + W)$ et l'équation $X = XU + V$ admet pour solution $X = (V + W)U^*$ avec W un langage quelconque.

Démonstration
Supposons qu'un certain langage X vérifie: $X = UX + V$ (1)
Démontrant par récurrence que:

$$X = U^{n+1}X + U^nV + U^{n-1}V + ... + U^2V + UV + V \ \forall n \in \mathbb{N} \ (2)$$

Il est clair que cette égalité est vraie pour $n = 0$.
D'autre part, en multipliant à gauche (1) par U^{n+1} il vient $U^{n+1}X = U^{n+2}X + U^{n+1}V$ et si (2) est vraie, le report de cette égalité dans (2) donne:

$$X = U^{n+2}X + U^{n+1}V + U^nV + U^{n-1}V + ... + U^2V + UV + V \ \forall n \in \mathbb{N}$$

Donc (2) est toujours vraie.
On déduit de (2) que U^nV est contenu dans X quel que soit n. Il en résulte que U^*V la réunion des langages U^nV, est contenu dans X.
Par conséquent il existe un langage W tel que $X = U^*V + W$. En reportant cette expression dans (2) on obtient:

$$X = U^{n+1}U^*V + U^{n+1}W + U^nV + U^{n-1}V + ... + U^2V + UV + V$$

qui prouve que $U^{n+1}W$ est contenu dans $X \ \forall n \in \mathbb{N}$.On peut donc ajouter tous ces langages à $X = U^* + W$ sans en changer la valeur, ce qui donne:

$$X = U^*V + W + (UW + U^2W + ... + U^nW + ...) = U^*V + U^*W = U^*(V + W)$$

Donc il existe W tel que: $X = U^*(V + W)$ (3)
Examinons quelle condition doit satisfaire W pour que le membre de droite de (3) soit une solution de (1).Calculons.

$$\begin{aligned} UX+V &= UU^*(V+W)+V \\ &= UU^*V+UU^*W+V \\ &= U^*V+UU^*+W \end{aligned}$$

Dans le cas où $\varepsilon \in U$ nous avons $UU^* = U^*$. Par conséquent $UX+V = U^*(V+W) = X$ et $U^*(V+W)$ est donc solution de (1) sans aucune condition sur W.
Examinons le cas $\varepsilon \notin U$ et montrons que $U^*W \subset U^*V$. Si ce n'était pas le cas,il y aurait un mot σ tel que $\sigma \in U^*W$ et $\sigma \notin U^*V$. Prenons σ de plus petit longueur possible. Par ce que $\sigma \in U^*W$ nous avons:

$$\sigma \in U^*V+U^*W = U^*(V+W) = X = UX+V = U^*V+UU^*W$$

et par ce que $\sigma \notin U^*V$, forcément $\sigma \in UU^*W = U(U^*W)$. Il existe donc $u \in U$ et $\tau \in U^*W$ tels que $\sigma = u\tau$.Mais $\tau \notin U^*V$ sinon $\sigma = u\tau \in UU^*V \subset U^*V$ ce qui est faux. donc $\tau \in U^*W$ et $\tau \notin U^*V$.
de plus, longueur(τ)=longueur(σ)-longueur(u) mais, $u \neq \varepsilon$ par ce que $\varepsilon \notin U$, donc longueur(τ) < longueur(σ), ce qui n'est pas possible compte tenu de la définition de σ.
Il en résulte que $U^*W \subset U^*V$ et par conséquent, $X = U^*V+U^*W = U^*V$.

4. Minimisation d'un AFD

4.1 Théorème d'existence de l'AFD minimal

4.1.1 Problème

Parmi les AFD reconnaissant un même langage L, peut-on en trouver un qui a le nombre minimal d'états?

Oui, et il est unique (à un renommage près des états) :

- On l'obtient en supprimant les états inaccessibles.
- Puis on identife les états restants qui jouent un rôle identique du point de vue de la reconnaissance.

4.1.2 Théorème de Myhill-Nérode

Théorème de Myhill-Nérode: Soit L un langage rationnel. Parmi tous les AFD reconnaissant L, il en existe un et un seul qui a un nombre minimal d'états.

4.2 Construction de l'AFD minimal

4.2.1 Définition

Nous allons maintenant définir de façon plus formelle ce qu'est l'automate minimal d'un langage. Soit $M = (Q;A;1;F;\delta)$ un AFD dont tous les états sont accessibles à partir de l'état initial.

Pour chaque état $i \in Q$, on défini le sous-ensemble $[i] \subseteq Q$ de la façon suivante:

$$[i] = \{ \text{k} \in \text{Q} \mid \text{k est équivalent à i} \}$$

Observation : Si i et j sont deux états dans Q alors $[i]$ et $[j]$ sont soit égaux ou soit disjoints.

4.2.2 Proposition

L'automate minimal reconnaissant le langage L(M) est l'AFD $M^{'}$=$(Q^{'};A;[1];F^{'};\delta')$ tel que:

- $Q^{'} = [i] \mid i \in Q.$
- $[1] \in Q^{'}$ est $l'etat$ $initial.$
- $F^{'} = \{[i] \in Q^{'} | i \in F\}$ est $l'ensemble$ des $etats$ $acceptants.$
- $\delta^{'} : Q^{'} \times \mathbf{A} \longrightarrow Q^{'}$ est la fonction de transition définie par:

$$\delta^{'}([i],a)=[\delta(i,a)]$$

4.2.3 Suppression des états inaccessibles

Construction par récurrence d'une suite d'ensembles Acc_i

- $Acc_0 = i.$
- $Acc_{i+1} = Acc_i \cup \delta(Acc_i, \Sigma).$

On s'arrête quand $Acc_{i+1} = Acc_i$, et on pose alors $Q = Acc_i$ (c'est-à-dire qu'on supprime les états de Q_i) .

Les états qu'on ne peut attendre à partir de l'état initial n'apportent aucune contribution au langage L reconnu par l'AFD.

4.2.4 Trouver les états équivalents

Comme nous venons de le voir, une fois que nous avons déterminé quels sont les états équivalents, il devient facile de construire l'automate minimal.
Le problème se réduit donc à trouver effectivement les états équivalents.
L'exemple que nous traitons après est relativement simple mais en général ce problème apparait beaucoup plus complexe. Heureusement, il existe un algorithme simple pour résoudre ce problème. En fait, plutôt que de chercher les paires d'états équivalents, nous allons déterminer quels sont les paires d'états qui ne le sont pas. Nous dirons que deux états sont distincts s'ils ne sont pas équivalents.
La première étape de l'algorithme consiste à construire un graphe dirigé G(M) à partir de l'AFD M = (Q; A; 1; F; δ). Afin de ne pas confondre ce graphe avec le graphe de transition nous l'appellerons graphe d'équivalence.
Les noeuds du graphe d'équivalence sont des paires d'états de Q, c'est-à-dire des sous-ensembles de Q contenant exactement deux éléments. Il y a un arc entre le noeud $\{i, j\}$ et le noeud $\{k, l\}$ si et seulement si $\{k, l\} = \{\delta(i, a), \delta(j, a)\}$ pour au moins une lettre a $\in$ A.

4.3 Exemple de minimisation d'automates

La figure 1 représente le graphe de transition d'un automate fini déterministe M1. Dans cette section nous nous posons la question suivante: Est-il possible de réduire le nombre d'états dans cet automate? De façon plus générale: Comment peut-on trouver le plus petit AFD possible qui reconnaisse un langage régulier donné?

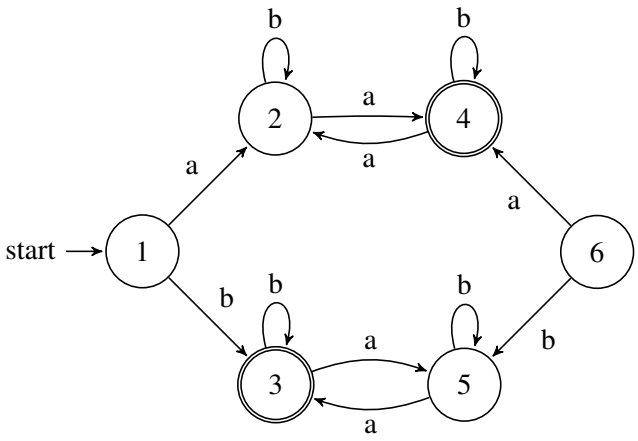


Figure 4.1: L'AFD $M_1 = (\{1,2,3,4,5,6\},\{a,b\},1,\{1,3,4\},\delta)$

Après avoir examiné M_1 on remarque que l'état 6 est inaccessible à partir de l'état initial. En d'autres termes, il n'existe aucun mot w $\in \{a,b\}^*$ tel que $\delta^*(1,w)$= 6. Si on enlève cet état, on obtient l'automate M_2 de la figure 2. On observe que M_2 est déterministe et reconnaît le même langage que M_1 puisque l'état que nous avons enlevé était, à toutes fins pratiques, inutile.
On peut faire de cette observation une règle générale:

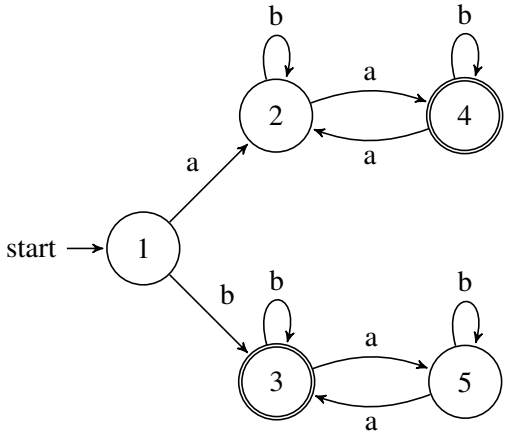


Figure 4.2: L'AFD M_2 obtenu en enlevant les états inaccessible de M_1

Régle 1 **:** Soit M un AFD et soit $M^{'}$ l'AFD obtenu en enlevant tous les états inacessibles de M. Alors $L(M) = L(M^{'})$.

Maintenant, après avoir appliqué la régle 1, peut-on encore réduire le nombre d'états de M_2? La réponse est encore oui! La figure 3 représente le graphe de transition de l'AFD M_3. Cet automate est le plus petit AFD reconnaissant $L(M_1)$.

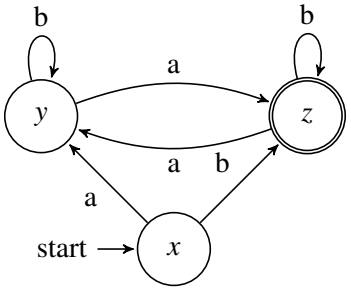


Figure 4.3: L'AFD minimal M_3 équivalent à M_1

Comment a-t-on obtenu cet automate? Pour répondre à cette question il est nécessaire de faire quelques observations sur l'AFD M_2 de la figure 2. Dénotons par L_i $(i = 1;2;3;4;5)$ l'ensemble des mots menant à un état final à partir de l'état i. Plus formellement, définissons

$$L_i = \{w \in a,b^* | \delta^*(i,w) \in \mathbf{F}\}$$

On observe que

- L_2 = $\{w \in a,b^*$ |w contient un nombre impair de a $\}$.
- L_3 = $\{w \in a,b^*$ | w contient un nombre pair de a $\}$.
- L_4 = $\{w \in a,b^*$ |w contient un nombre pair de a $\}$.
- L_5 = $\{w \in a,b^*$ |w contient un nombre impair de a $\}$.

On a donc

$$L_2 = L_5 \text{ et } L_3 = L_4$$

On peut interpréter cette observation de la façon suivante:

- À chaque fois que l'on se retrouve dans l'état 2, on pourrait poursuivre l'exécution à partir de l'ètat 5 sans rien changer au résultat.
- À chaque fois que l'on se retrouve dans l'état 3, on pourrait poursuivre l'exécution à partir de l'état 4 sans rien changer au résultat.
- À chaque fois que l'on se retrouve dans l'état 4, on pourrait poursuivre l'exécution à partir de l'état 3 sans rien changer au résultat.
- À chaque fois que l'on se retrouve dans l'état 5, on pourrait poursuivre l'exécution à partir de l'état 2 sans rien changer au résultat.

Cette observation est très importante pour notre propos et elle suggère la définition suivante:
Définition: Deux états i et j sont équivalents si $L_i = L_j$. En d'autres termes, i et j sont équivalents si pour tout $w \in A^*$, on a

$$\delta^*(i;w) \in F \text{ si et seulement si } \delta^*(j;w) \in F$$

Ainsi, dans l'AFD M_2 les états 2 et 5 sont équivalents et les états 3 et 4 sont équivalents. L'état 1 n'est équivalent à aucun autre état.

Remarque:
La définition précédente est une équivalence au sens mathématique du terme. De façon générale, si i et j sont deux états équivalents d'un automate donné, alors à chaque fois que l'on se retrouve dans l'état i, on pourrait poursuivre l'exécution à partir de l'état j sans rien changer au résultat. Cela conduit à l'observation suivante:
Observation : Il est toujours possible d'ajouter des transitions ε entre deux états équivalents sans rien changer au langage reconnu.

Si l'on applique cette observation sur l'AFD de la figure 2, on obtient l'automate M_4 dont le graphe de transition est représentée à la figure 4.

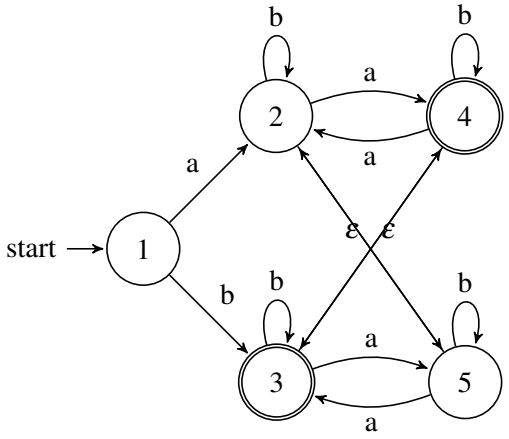


Figure 4.4: L'AFN M_4 obtenu à partir de M_2 en ajoutant des transitions ε entre les pairs d'états équivalents

Remarquons que l'automate obtenu n'est plus déterministe. Cependant, on peut obtenir un AFD M_5 à partir de L'AFN M_4 en utilisant la méthode vue précédemment. Le graphe de transition de l'AFD M_5obtenue est illustré à la figure 5.

On remarque que l'AFD M_5 est identique à l'automate M_3, l'automate minimal qui reconnaît le langage $L(M_1)$. On remarque surtout que la transformation de M_4 à M_5n'a pas augmenté le nombre d'états. Bien au contraire.

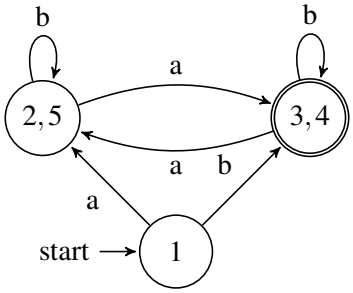
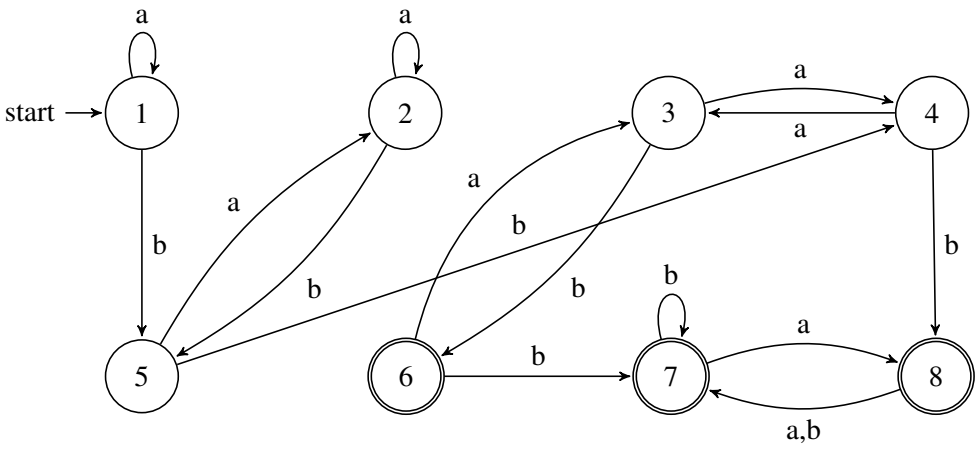


Figure 4.5: L'AFD M_5 obtenu à partie de l'AFN M_4

Les états de M_5 sont des ensembles d'états de M_4 mais ces ensembles sont disjoints. En d'autres termes, chaque état de M_4 se retrouve dans un unique état de M_5. Cela nous permet d'énoncer une seconde règle générale:

Règle 2 : Soit $M = (Q;A;q_0;F;\delta)$ un AFD et soit $M' = (Q';A;S';F';\delta')$ l'automate minimal qui reconnaît $L(M)$. Alors les éléments de Q' sont des sous-ensembles disjoints de Q. De plus, deux états de Q appartiennent au même sous-ensemble dans Q' si et seulement s'ils sont équivalents.

Exemple: Soit l'automate suivant:



$i = 1, F = \{6,7,8\}$

	1	2	3	4	5	6	7	8
a	1	2	4	3	2	3	8	7
b	5	5	6	8	4	7	7	7

Table 4.1:

voit facilement qu'aucun mot ne sépare les états 7 et 8. À partir de ces états, on peut lire n'importe quel mot sur a, b. Ces deux états pourront donc être confondus dans l'AFD minimal.

1. Équivalence $\sim_0$
Initialement, les deux classes sont celle des états finals $[6]_0$ = {6, 7, 8} et celle des autres

$$[1]_0 = \{1,2,3,4,5\}$$

	1	2	3	4	5	6	7	8
$\sim_0$	$[1]_0$	$[1]_0$	$[1]_0$	$[1]_0$	$[1]_0$	$[6]_0$	$[6]_0$	$[6]_0$

Table 4.2:

2. Équivalence $\sim_1$
Pour éventuellement scinder des classes, il faut regarder dans quelles classes vont les transitions partant des états. On en déduit les nouvelles classes pour $\sim_1$.

	1	2	3	4	5	6	7	8
$\sim_0$	$[1]_0$	$[1]_0$	$[1]_0$	$[1]_0$	$[1]_0$	$[6]_0$	$[6]_0$	$[6]_0$
a	$[1]_0$	$[1]_0$	$[1]_0$	$[1]_0$	$[1]_0$	$[1]_0$	$[6]_0$	$[6]_0$
b	$[1]_0$	$[1]_0$	$[6]_0$	$[6]_0$	$[1]_0$	$[6]_0$	$[6]_0$	$[6]_0$
$\sim_1$	$[1]_1$	$[1]_1$	$[3]_1$	$[3]_1$	$[1]_1$	$[6]_1$	$[7]_1$	$[7]_1$

Table 4.3:

3. Équivalence $\sim_2$
Dans le tableau ci-après, on n'indique que les transitions qui vont dans une autre classe.

	1	2	3	4	5	6	7	8
$\sim_1$	$[1]_1$	$[1]_1$	$[3]_1$	$[3]_1$	$[1]_1$	$[6]_1$	$[7]_1$	$[7]_1$
a								
b			$[6]_1$	$[7]_1$	$[3]_1$			
$\sim_2$	$[1]_2$	$[1]_2$	$[3]_2$	$[4]_2$	$[5]_2$	$[6]_2$	$[7]_2$	$[7]_2$

Table 4.4:

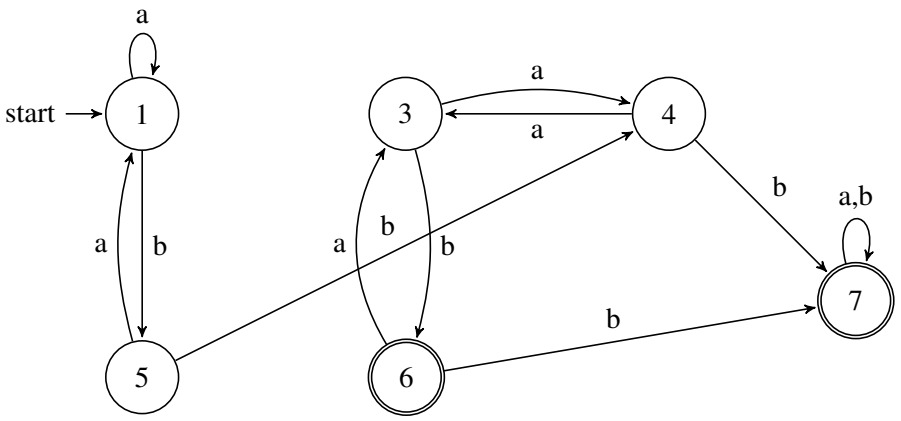
4. Équivalence $\approx 3 = \approx 2$
Il n'y a plus aucune scission de classes, donc la suite d'équivalences s'est stabilisée, et on obtient celle de l'AFD minimal.

	1	2	3	4	5	6	7	8
$\sim_2$	$[1]_2$	$[1]_2$	$[3]_2$	$[3]_2$	$[1]_2$	$[6]_2$	$[7]_2$	$[7]_2$
a b								
$\sim_3$	$[1]_3$	$[1]_3$	$[3]_3$	$[4]_3$	$[5]_3$	$[6]_3$	$[7]_3$	$[7]_3$

Table 4.5:

On voit donc qu'on peut confondre
- les états 1 et 2
- les états 7 et 8

On en déduit l'AFD minimal.



5. Algorithme de Brzozowski

5.1 Principe

L'algorithme de Brzozowski de minimisation d'un automate fini, publié par Janusz A. Brzozowski en 19631, est un algorithme de minimisation d'un automate fini fondé sur une double transposition et une double déterminisation.

L'algorithme est, avec l'algorithme de Moore et l'algorithme de Hopcroft, l'un des trois algorithmes principaux de minimisation d'un automate fini déterministe. Ce n'est pas le plus efficace, mais il est plus simple à expliquer.

5.2 L'automate transposé

L'automate transposé de $\mathscr{A}$, noté $tr(\mathscr{A})$, est l'automate obtenu en inversant le sens des transitions, et en échangeant les états initiaux avec les états terminaux. Formellement, c'est l'automate défini par

$tr(\mathscr{A}) = (Q, \mathscr{F}^R, T, I)$, avec $\mathscr{F}^R = \{(p,a,q) \mid (q,a,p) \in \mathscr{F}\}$.

L'algorithme de minimisation part d'un automate $\mathscr{A}$ reconnaissant un langage L L (déterministe ou non) et construit l'automate

$\mathscr{A}(L) = \det(\mathrm{tr}(\det(\mathrm{tr}(\mathscr{A}))))$. L'automate $\mathscr{A}(L)$ est l'automate déterministe minimal reconnaissant LL.

5.3 Algorithme

L'algorithme de minimisation d'un automate fini$\mathscr{A}$ est donc en deux étapes :

Calculer $\det(\mathrm{tr}(\mathscr{A}))$, en transposant l'automate puis en le déterminisant, par la procédure usuelle par exemple, qui ne conserve que les états accessibles ; Répéter l'opération 1 sur l'automate $\det(\mathrm{tr}(\mathscr{A}))$ obtenu. La première étape se fait à partir d'une automate fini $\mathscr{A}$ quelconque, déterministe ou non, la deuxième par contre prend en argument l'automate déterministe accessible complet $\det(\mathrm{tr}(\mathscr{A}))$

5.4 Complexité

La complexité en temps et en place de l'algorithme est exponentielle dans le pire des cas en fonction de la taille de l'automate de départ, car l'automate intermédiaire $\mathscr{A}^\sim$ peut être exponentiel. On pourrait penser que la complexité est doublement exponentielle puisqu'il y a deux déterminisations consécutives, mais ce n'est pas le cas : Le coût d'une déterminisation est en $O(|\mathscr{A}|+|\det(\mathscr{A})|)$, l'automate intermédiaire $\mathrm{tr}(\det(\mathrm{tr}(A)))$ est de taille $O(2^n)$, l'automate final aussi, d'où une complexité totale en $O(2^n)$. Un exemple où la borne est atteinte est fourni par le langage K_n des mots de longueur au moins n n sur un alphabet à deux lettres a a et b b, et dont la n n-ième lettre est un a a, en d'autres termes :

$K_n = \{a,b\}^{n-1}a\{a,b\}^*$. Ce langage est reconnu par un automate à $n+1$ états plus un état puits, mais son transposé, une fois déterminisé, requiert 2^n états3.

Il a été observé3 que l'algorithme semble donner des résultats meilleurs dans la pratique que l'on pourrait penser. Ceci pose la question de la complexité en moyenne de l'algorithme. Un article de De Felice et Nicaud4,5 apporte une réponse à cette question : l'algorithme de Brzozowski est génériquement super-polynomial. Ce terme technique signifie que la complexité en moyenne, et même que la complexité générique de l'algorithme est plus grande que tout polynôme. Le terme « complexité générique » signifie que l'on est autorisé à « ignorer », dans le calcul de la moyenne, un ensemble de cas particulièrement désavantageux, sous réserve que cet ensemble de cas soit de taille négligeable en un sens que l'on peut préciser.

5.5 Justification

L'énoncé précis qui justifie la construction est le suivant :

Proposition (Brzozowski) — Soit $\mathscr{A}$ un automate fini déterministe accessible, et soit $\mathscr{A}^\sim = \det(\mathrm{tr}(\mathscr{A}))$ l'automate fini déterministe complet accessible obtenu à partir de l'automate transposé par déterminisation et élimination des états inaccessibles. Alors l'automate $\mathscr{A}^\sim$ est minimal et reconnaît le langage miroir du langage reconnu par $\mathscr{A}$.

La démonstration n'est pas très difficile : soit L le langage reconnu par l'automate déterministe complet$\mathscr{A} = (Q, q_0, T)$ et soit $\mathrm{tr}(\mathscr{A}) = (Q, T, q_0,)$ l'automate transposé reconnaissant $M = L^\sim$.

Pour prouver que $\mathscr{A}^\sim$ est minimal, on montre que si $u^{-1}M = v^{-1}M$, alors $T\cdot u = T\cdot v$ dans l'automate $\mathscr{A}^\sim$. Ceci prouve que $\mathscr{A}^\sim$ est isomorphe à l'automate minimal de M.

Soit p un état de $T\cdot u$. Comme $\mathscr{A}$ est accessible, il existe un mot w w tel que $p\cdot w = q_0$ dans $\mathrm{tr}(\mathscr{A})$. On a donc $uw \in M$ et aussi $vw \in M$. Comme $\mathscr{A}$ est déterministe, p est l'unique état tel que $p\cdot w = q_0 dans \mathrm{tr}(\mathscr{A})$. Tout chemin de T vers q_0 étiqueté par vw passe par l'état p , et donc $p \in T\cdot v$. Ceci montre que $T\cdot u \subset T\cdot v$. L'inclusion dans l'autre sens se montre de la même façon.

6. Algorithme de Hopcroft

L'algorithme, appelé ainsi d'après son inventeur, est dû à John Hopcroft et il a été présenté en 1971; il opère par raffinement successif d'une partition initialement grossière de l'ensemble des états de l'automate déterministe fini à minimiser.

Les algorithmes de Moore et de Hopcroft sont des algorithmes de raffinements de partitions. On commence, dans les deux algorithmes, avec une hypothèse naïve, en supposant que l'automate minimal n'a que deux états et, en examinant la fonction de transition, on raffine la partition initiale tant que c'est nécessaire. Le raffinement consiste à trouver des états séparables au sens défini ci-dessus : deux états p et q sont séparables s'il existe un mot w w tel que l'état $p \cdot w$ $p \cdot$ w est final et $q \cdot w$ $q \cdot$ w n'est pas final, ou vice-versa.

Lorsque l'algorithme de raffinement s'arrête, chaque classe de la partition représente un ensemble d'états deux-à-deux inséparables, et deux classes distinctes sont séparables. L'automate minimal est obtenu en prenant comme états les classes de la partition, et comme transitions les transitions induites sur les classes par les transitions de l'automate de départ.

6.1 Principe

L'algorithme débute avec la partition la plus grossière, dont les deux classes sont composées des états terminaux et des autres. La partition est progressivement raffinée en un nombre croissant de classes plus petites. Chaque tour de l'algorithme partage des ensembles d'états en deux parties plus petites.

L'opération de base est la coupure (« splitting » en anglais). Un ensemble d'états X X est coupé par la paire (Z,a) formée d'un ensemble Z et d'une lettre a si les ensembles

$X' = \{q \in X \mid q \cdot a \in Z\}$ et $X'' = \{q \in X \mid q \cdot a \notin Z\}$ sont tous les deux non vides. Dans le cas contraire, on dit que X est stable pour (Z,a) (Z,a).

L'algorithme maintient un ensemble $\mathscr{W}$ $\mathscr{W}$ de couples (Z,a) , candidats à couper des éléments de la partition en cours; cet ensemble de candidats en attente est appelé le « waiting set » en anglais.

L'algorithme est décrit par le pseudo-code suivant :

P := F, Q F; "Partition initiale" W := l'ensemble vide; "Candidats en attente" for each a in A do ajouter (min(F, Q F),a) à W; "Initialisation de l'ensemble W" while (W l'ensemble vide) do choisir ensemble (Z,a) dans W et l'enlever de W for each X de P coupé par (Z,a) en X' et X" do "Calcul de la coupe" remplacer X in P par les deux ensembles X' et X" "Raffinement de la

partition" for each b in A do "Mise-à-jour de l'ensemble" if (X,b) est in W "des candidats en attente" remplacer (X,b) in W par (X',b) et (X",b) else ajouter le plus petit de (X',b) et (X",b) à W; end; end; end;

Le but de l'algorithme est d'obtenir par coupes successives un automate déterministe minimal. L'algorithme procède en testant la stabilité de chaque groupe d'états de la partition P P par toutes les coupes possibles.

L'algorithme débute avec la partition P composée de l'ensemble des états terminaux T et de l'ensemble des états non terminaux $Q \setminus T$. $\mathscr{W}$ est l'ensemble des candidats en attente pour raffiner la partition P. On ajoute à $\mathscr{W}$ tout couple de la forme (S,a), avec a une lettre et S le plus petit des ensembles T et $Q \setminus T$.

L'algorithme choisit itérativement un ensemble (Z,a) dans l'ensemble des candidats en attente $\mathscr{W}$, le retire de l'ensemble et, pour chaque partie X de la partition courante, il teste si X est coupé par (Z,a). Dans l'affirmative, la partition est mise à jour en remplaçant X par les deux parties X' et X'' résultant de la coupure. De plus, l'ensemble des candidats en attente est augmenté, pour toute lettre b, de (X',b) et (X'',b) ou du plus petit de (X',b) et (X'',b) , selon que (X,b) est dans ce waiting set ou non.

A chaque itération, soit l'ensemble $\mathscr{W}$ perd un élément, soit la partition est raffinée. Comme la partition ne peut être raffinée indéfiniment parce que l'automate est fini, le nombre d'itérations est donc fini. Ceci prouve donc la terminaison de l'algorithme.

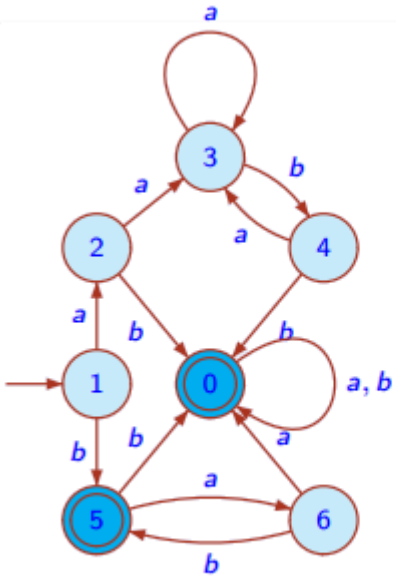
6.2 Complexité et optimalité

La complexité en temps de l'algorithme de Hopcroft, dans le pire des cas, est $O(s \cdot n \log n)$ où n est le nombre d'états de l'automate et s est la taille de l'alphabet. La borne est conséquence du fait que, pour chacune des sn transitions de l'automate, les ensembles retirés du waiting set qui contiennent l'état d'arrivé d'une transition ont une taille diminuée de moitié au moins à chaque fois, donc chaque transition participe à au plus $O(\log n)$ étapes de coupure dans l'algorithme.

L'algorithme de Hopcroft est le plus efficace des algorithmes de minimisation connus en 2010. L'algorithme initial demande que l'automate de départ soit déterministe et complet. On peut adapter l'algorithme au cas des automates déterministes finis incomplets12. L'implémentation est en temps $O(n + m \cdot \log m)$, où m m est le nombre de transitions de l'automate. On a toujours $m \leq snm \leq$sn.

Il reste un certain degré de liberté dans le choix du candidat que l'on retire de l'ensemble $\mathscr{W}$ des candidats en attente. Cela dépend aussi du choix de la structure de donnée choisie : l'ensemble peut être par exemple organisé en pile (structure LIFO) ou en file (structure FIFO). Il a été prouvé qu'un choix approprié de la stratégie permet à l'algorithme de Hopcroft d'être toujours meilleur que l'algorithme de Moore. En particulier, l'algorithme a une complexité en moyenne en $O(sn \log \log n)$.

Exemple :



Initiale partition $\mathcal{P}$:	$05\|12346$
Waiting set $\mathcal{W}$:	$(05, a), (05, b)$
Splitter chosen :	$(05, a)$
Split states :	$a^{-1}05 = 06$
First class to split:	$12346 \rightarrow 1234\|6$
Splitters to add :	$(6, a)$ and $(6, b)$
Second class to split :	$05 \rightarrow 0\|5$
Splitter to add:	$(5, a)$ (or $(0, a)$)
Splitter to replace:	$(05, b)$: by $(0, b)$ and $(5, b)$
New partition $\mathcal{P}$:	$0\|1234\|5\|6$
New waiting set $\mathcal{W}$:	$(0, b), (6, a), (6, b), (5, a), (5, b)$

Conclusion

De la nécessité de minimiser un AFD :

Deux AFD sont équivalents si et seulement si elles acceptent le même langage.

Pour chaque AFD, il existe un AFD équivalent avec le plus petit nombre d'états:

Sa table de transitions prend moins d'espace en mémoire

Pour minimiser un AFD, il faut d'abord enlever les états inaccessibles.

Ensuite, appliquer la définition inductive d'une paire d'états distinguables pour marquer toutes les paires indistinguables: si et sj sont distinguables ssi si est dans F et sj n'est pas dans F ou vice versa, ou Il existe a dans A tel que T(si, a) et T(sj, a) sont distinguable

À la fin du marquage, les paires d'états non marquées sont indistinguables.

L'AFD minimal correspond aux classes d'équivalence définies par les paires indistinguables: La classe d'un état s, [s], est l'ensemble des états s', tel que la paire (s,s') est indistinguable de s.

Bibliographie

-http://pop-art.inrialpes.fr/ girault/Cours/Automates/
-http://ehess.modelisationsavoirs.fr/marc/ens/langages/2006/AutChapitre6.pdf
-http://openclassrooms.com/courses/redigez-des-documents-de-qualite-avec-latex/
-tex.stackexchange.com/questions/
-fr.wikipedia.org/
-www.texample.net/tikz/examples/feature/automata