Automates

Étudiant : Benjamin André
Directrice :Véronique Bruyère
17 juin 2020

Faculté des Sciences • Université de Mons • Place du Parc 20 • B-7000 Mons

Table des matières

1 Introduction

Deuxième version du document. *TODO : rédiger une introduction*

2 Langage

Cette section pose différents concepts et notations pour arriver à la notion de langage. Celle-ci reprennent les notations proposées par Hopcroft et al. [1].

2.1 Alphabet

Un *alphabet*, nommé Σ par convention, est un ensemble fini et non vide de *symboles*.

Exemple 2.1 *Voici trois alphabets :*
- — $\Sigma = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$, *l'alphabet des chiffres*
- — $\Sigma = \{a, b, c, ..., z, A, B, C, ..., Z\}$, *l'alphabet latin*
- — $\Sigma = \{0, 1\}$, *l'alphabet binaire*

2.2 Mots

Soit l'alphabet Σ et un entier naturel k. Un *mot* sur Σ est une suite finie de k éléments de Σ notée $w = a_1 \dots a_k$.

L'entier k est la *longueur* de ce mot aussi notée $|w| = k$.

Exemple 2.2 $w = 01110010$ *est un mot sur* $\Sigma = \{0, 1\}$

Le *mot vide* est un mot de taille $k = 0$, noté $w = \epsilon$.

Σ^k est l'ensemble des mots sur Σ de longueur k.

L'ensemble de tous les mots possibles sur Σ est noté $\Sigma^* = \bigcup_{k=0}^{\infty} \Sigma^k$.

La *concaténation* de deux mots $w = a_1 \dots a_k$ et $x = b_1 \dots b_j$ est l'opération consistant à créer un nouveau mot wx, mot de taille $i = k + j$ s'écrivant $wx = a_1 \dots a_k b_1 \dots b_j$.

Exemple 2.3 *Soient les mots* $x = 41$ *et* $y = 31$. *Alors* $xy = 4131$ *et* $yx = 3141$.

Lemme 2.4 ϵ *est* l'identité pour la concaténation, *à savoir pour tout mot* w, $w\epsilon = \epsilon w = w$. *Par définition de la concaténation, tout mot concaténé avec* ϵ *retourne le même mot.* $\square$

L'exponentiation d'un symbole a à la puissance k, notée a^k, retourne un mot de longueur k obtenu par la concaténation de copies du symbole a. Noter que $a^0 = \epsilon$.

2.3 Langage

Un ensemble de mots sur Σ est un *langage* [1], noté L. Alternativement, $L \subseteq \Sigma^*$. Étant donné que Σ^* est infini, L peut l'être également.

Exemple 2.5 *Voici des exemples, utilisant plusieurs modes de définition. Σ y est implicite, mais il peut être donné explicitement.*

- $L = \{12, 35, 42, 7, 0\}$, *un ensemble défini explicitement*
- $L = \{0^k 1^j | k + j = 7\}$, *les mots de 7 symboles sur* $\Sigma = \{0, 1\}$ *ne contenant pas* 10. *Ici,* L *est donné par notation ensembliste*
- *L est donné par "Tous les noms de villes belges.". Ici L est défini en français.*
- *$\emptyset$ est un langage sur tout alphabet.*
- $L = \{\epsilon\}$ *ne contient que le mot vide, et est un langage sur tout alphabet.*

Opérations sur les langages

Soient L et M deux langages. $L \cup M = \{w | w \in L \vee w \in M\}$ est l'*union* de ces deux langages et en donne un nouveau. Ce langage est composé des mots venant d'un des deux langages.

Le langage composé de tous les mots produit par la concaténation d'un mot de L avec un mot de M est une *concaténation* de ces deux langages et s'écrit LM.

La *fermeture* de L est notée L^* et donne un langage constitué de tous les mots qui peuvent être construits par un concaténation d'un nombre arbitraire de mots de L.

2.4 Expression régulière

Certains langages peuvent être exprimés par une *expression régulière*. Un exemple de celles-ci est $01 * 0$ qui décrit la langage constitué de tous les mots commençant et finissant par 0 avec uniquement des 1 entre les deux.

Les expressions régulières suivent un algèbre avec ses opérations et leur priorités. Le langage décrit par une expression est construit de façon inductive par ces différentes opérations. Pour une expression régulière E, le langage exprimé est noté $L(E)$. Un langage qui peut être exprimé par une expression régulière est dit *langage régulier*.

Cas de base Certains langages peuvent être construits directement sans passer par l'induction :

- ϵ est une expression régulière. Elle exprime le langage $L(\epsilon) = \{\epsilon\}$
- $\emptyset$ est une expression régulière décrivant $L(\emptyset) = \emptyset$
- Si a est un symbole, alors **a** est une expression régulière composée uniquement de a. $L(a) = \{a\}$.
- Une variable, souvent en majuscule et italique, représente un langage quelconque, par exemple L.

Induction Les autres langages réguliers sont construits suivant différentes règles d'induction présentées par ordre décroissant de priorité :

- Si E est une expression régulière, (E) est une expression régulière et $L((E)) = L(E)$.
- Si E est une expression régulière, E^* est une expression régulière représentant la fermeture de $L(E)$, à savoir $L(E^*) = L(E)^*$.

— Si E et F sont des expressions régulières, EF est une expression régulière décrivant la concaténation des deux langages représentés, à savoir $L(EF) = L(E)L(F)$. La concaténation étant commutative, l'ordre de groupement n'est pas important, mais par convention, la priorité est à gauche.
— Si E et F sont des expressions régulières, $E + F$ est une expression régulière donnant l'union des deux langages représentés, à savoir $L(E + F) = L(E) \cup L(F)$. Ici encore, l'opération est commutative et la priorité est à gauche.

Exemple 2.6 *Soit l'expression $E = (b + ab)b^*a(a + b)^*$ qui représente le langage L.*

— ***ba*** *fait partie de L. En effet, en développant E avec des choix sur les unions et le degré d'une fermeture, on obtient $E = (b)b^0a(a + b)^0 = b\epsilon a\epsilon = ba$.*
— ***ababbab*** *fait partie de L. En développant à nouveau E en posant des choix sur les unions et fermetures, on obtient $E = (ab)b^0a(a + b)^4 = ab\epsilon a(a + b)(a + b)(a + b)(a + b) = ababbab$.*
— ***aa*** *ne fait* ***pas*** *partie de L. Supposons par l'absurde que $aa \in L$. Alors il existerait une façon de décomposer E en aa. Or, les premiers symboles doivent être soit b, soit ab. Il y a contradiction : E ne peut pas être décomposé. Comme aa ne peut pas être construit par E, $aa \notin L$.*

3 Automate Déterministe Fini

3.1 Définition

Un *automate déterministe fini (ADF)* $A = (Q, \Sigma, q_0, \delta, F)$ est défini comme suit :
— Q est un ensemble fini d'*états*
— Σ est un alphabet
— $q_0 \in Q$ est l'*état initial*
— $\delta : Q \times \Sigma \to Q$ est la *fonction de transition*. A partir d'un état q de Q, en fonction d'un symbole a, elle retourne un état de Q : $\delta(q, a)$. Cette transition est dite *transition sur a*.
— $F \subseteq Q$ est un ensemble d'*états acceptants*.

Exemple 3.1 *On considère l'automate $A = (Q, \Sigma, q_0, \delta, F)$ défini comme suit :*
— $Q = \{q_0, q_1, q_2, q_3, q_4, q_5, q_6\}$
— $\Sigma = \{0, 1\}$
— *q_0 est l'état du même nom*
— *La fonction de transition δ est décrite par la table 1. L'intersection d'une ligne reprenant un élément $q \in Q$ et d'une colonne $a \in \Sigma$ donne l'état $\delta(q, a)$.*
— $F = \{q_d\}$

	a	b
$\rightarrow q_0$	q_2	q_1
q_1	q_3	q_5
q_2	q_4	q_5
q_3^*	q_3	q_3
q_4	q_4	q_4
q_5	q_3	q_1
q_6	q_4	q_5

FIGURE 1: La table de transitions δ

Via cette notation, Q et Σ sont explicites. En dénotant l'état initial par $\rightarrow$ et les états acceptants par $*$ en exposant, on obtient une définition complète d'un automate : $(Q, \Sigma, q_0, \delta, F)$.

3.2 Graphe d'automate déterministe fini

Le *graphe d'un automate déterministe fini* $A = (Q, \Sigma, q_0, \delta, F)$ est un graphe dirigé construit comme suit :

- Chaque nœud du graphe correspond à un état de Q
- Chaque arc a un symbole de Σ comme étiquette. Un arc relie un état q_0 à un état q_1. Cet arc défini $\delta(q_0, a) = q_1$, un transition de la fonction de transition. Si plusieurs symboles causent une même transition de q_0 à q_1, il n'y a qu'une seule étiquette sur l'arc, listant ces différents symboles.
- L'état initial est mis en évidence par une flèche entrante.
- Les états acceptants sont représentés par un double cercle, en opposition au simple cercle des autres nœuds.

Exemple 3.2 *Voici le graphe représentant l'automate défini par la table 1*

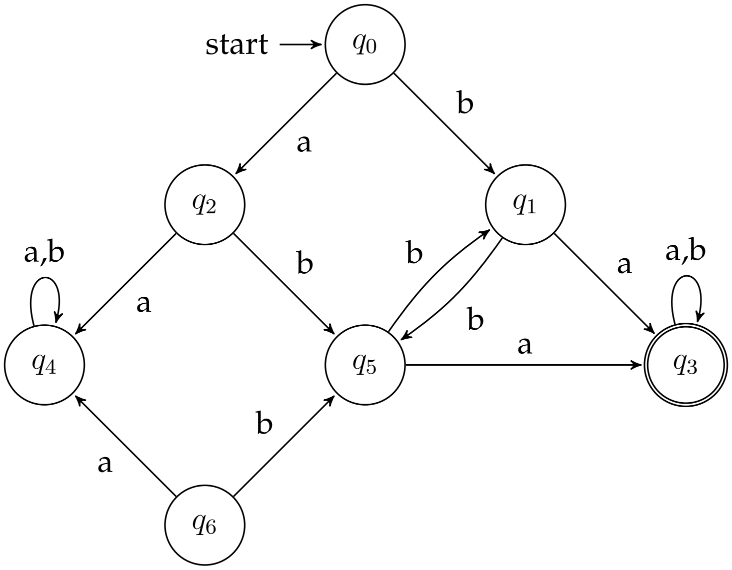


FIGURE 2: Automate A_1

Cette représentation d'un automate peut sembler plus naturelle pour un humain alors que la table de transitions est plus proche d'un langage informatique. De plus, dans la représentation par graphe, les ensembles Q et Σ sont implicites et doivent être définis ou déduits à part.

3.3 Chemin

La *fonction de transition étendue*

$$\hat{\delta} : Q \times \Sigma^* \to Q$$

prend en entrée un état de Q et un mot w sur Σ et retourne un état de Q.

$\hat{\delta}$ est définie de façon récursive par sur w :

Cas de base Il y a deux cas de base :

— w est un mot vide : $\hat{\delta}(q, \epsilon) = q$

— w est un symbole : $\hat{\delta}(q, w)$ avec $w = a \in \Sigma$. Alors, le chemin utilise la fonction de transition : $\hat{\delta}(q, a) = \delta(q, a)$.

Pas de récurrence Si $|w| > 1$, alors $w = xa$ avec x un mot sur Σ et a un symbole de Σ. Les chemins sur des mots de longueur strictement supérieure à 1 sont définis comme $\hat{\delta}(q, w) = \hat{\delta}(q, xa) = \delta(\hat{\delta}(q, x), a)$.

Il se peut que δ ne soit pas définie pour une paire d'arguments. Auquel cas, $\hat{\delta}$ ne l'est pas non plus.

Un *chemin* est une application de cette fonction sur un état et un mot.

Exemple 3.3 *Considérons l'automate A de la figure 2. Il existe un chemin de q_0 à q_5 : $\hat{\delta}(q_0, ab) = \delta(\hat{\delta}(q_0, a), b) = \delta(\delta(q_0, a), b) = \delta(q_2, b) = q_5$.*

3.4 Langage défini par un automate

Le langage représenté par un automate $A = (Q, \Sigma, q_0, \delta, F)$ peut alors se définir comme les mots qui, par l'application de $\hat{\delta}$ sur l'état initial, donnent un état acceptant :

$$L(A) = \{w \in \Sigma^* | \hat{\delta}(q_0, w) \in F\}$$

Ainsi, un mot w appartient à un langage L défini par l'automate A si $\hat{\delta}(q_0, w) \in F$. L'algorithme 1 représente cette appartenance pour un mot.

Algorithme 1 Appartenance d'un mot à un langage défini par un automate

Requis: un mot w, un automate $A = (Q, \Sigma, q_0, \delta, F)$ représentant L
Promet: si w appartient à L

```
1: q_c ← q_0 {q_c est l'état courant}
2: tant que |w| > 0 faire
3:     décomposer w en ax avec a ∈ Σ et x le reste du mot
4:     q_c ← δ(q_c, a) {passage à l'état suivant}
5:     w ← x
6: fin tant que
7: retourner si q_c appartient à F
```

Complexité *Si $|w| = n$, l'algorithme 1 est en $\mathcal{O}(n)$. En effet, les étapes 1 et 7 sont en temps constant. La boucle de l'étape 2 est parcourue n fois (la taille étant diminuée de 1 exactement à chaque itération). Le test de 2 et les opérations de 3 et 5 peuvent être faites en temps constant (par exemple, en voyant w comme une queue). L'étape 4, déterminante, peut être effectuée en temps constant également, par exemple avec l'utilisation d'un tableau de transition.*

3.5 La relation R_M

Soit un automate $A = (Q, \Sigma, q_0, \delta, F)$. Définissons la relation R_M entre deux états :

$$xR_My \iff (\forall w \in \Sigma^*, \hat{\delta}(x, w) \in F \iff \hat{\delta}(y, w) \in F)$$

Intuitivement, ces deux états sont en relation si tout mot lu à partir de celui-ci mène à des états étant simultanément acceptants ou non.

Proposition 3.4 *R_M est une relation d'équivalence.*

Preuve 3.4.1 *Montrer que R_M est une relation d'équivalence revient à montrer qu'elle est réflexive, transitive et symétrique.*

- ***Réflexive :*** *Soient un état $x \in Q_M$ et $w \in \Sigma^*$. Alors, $\hat{\delta}(x, w) \in F \iff \hat{\delta}(x, w) \in F$ et par définition, xR_Mx.*
- ***Transitive :*** *Soient les états $x, y, z \in Q_M$ tels que xR_My et yR_Mz ainsi que $w \in \Sigma^*$. Par hypothèse, $\hat{\delta}(x, w) \in F \iff \hat{\delta}(y, w) \in F$ et $\hat{\delta}(y, w) \in F \iff \hat{\delta}(z, w) \in F$. Par transitivité de l'implication, on obtient $\hat{\delta}(x, w) \in F \iff \hat{\delta}(z, w) \in F$. On a donc xR_Mz.*

— ***Symétrique :*** *Soient les états $x, y \in Q_M$ tels que xR_My et un mot $w \in \Sigma^*$. Par hypothèse, $\hat{\delta}(x,w) \in F \iff \hat{\delta}(y,w) \in F$. En lisant la double implication depuis la droite, on a bien $\hat{\delta}(y,w) \in F \iff \hat{\delta}(x,w) \in F$ et donc yR_Mx.*

Corrolaire 3.4.2 *R_M sépare les états de Q en classes d'équivalence.*

La classe d'équivalence de tous les états en relation R_M avec q (qui sert alors de *représentant*) se note $[[q]]$ ou par une lettre majuscule, typiquement S ou T.

La *congruence à droite* d'une relation R entre des mots sur un alphabet Σ est définie comme :

$$\forall x, y \in \Sigma^*, xRy \Rightarrow \forall a \in \Sigma, xaRya$$

Proposition 3.5 *R_M est congruente à droite.*

Preuve 3.5.1 *Si la relation est vraie pour deux état, elle reste valable pour les états atteints par la lecture d'un symbole quelconque. Soient les états $x, y \in Q_M$ tels que xR_My. Soit un symbole $a \in \Sigma$. Par hypothèse,*

$$\forall w \in \Sigma^*, \hat{\delta}(x,w) \in F \iff \hat{\delta}(y,w) \in F$$

C'est donc vrai en particulier pour $w = au, u \in \Sigma*$. Dès lors,

$$\hat{\delta}(x,au) \in F \iff \hat{\delta}(y,au) \in F$$

$$\hat{\delta}(\delta(x,a),u) \in F \iff \hat{\delta}(\delta(y,a),u) \in F$$
$$\hat{\delta}(p,u) \in F \iff \hat{\delta}(q,u) \in F$$

Corrolaire 3.5.2 *Pour chaque symbole, toutes les transitions sortant d'une classe d'équivalence mènent à une même classe d'équivalence :* $\forall a \in \Sigma, \exists T, \forall q \in S, \delta(q,a) \in T$ *avec T une classe d'équivalence.*

3.6 Automate et problème de décision

Une notion liée aux langages est celle de *problème*. Une forme de problème est celle dite de *décision* : une question à laquelle la réponse est oui ou non.

Ces problèmes de décision peuvent être exprimés en terme d'appartenance d'un mot à un langage.

Par exemple, prenons l'alphabet des chiffres $\Sigma = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$. Considérons ensuite le langage $L = \{w|\text{le nombre représenté par } w \text{ est pair}\}$.

Demander si un nombre est pair peut alors être traduit par l'appartenance d'un mot le représentant à L. Si le langage peut être représenté par un automate déterministe fini, la réponse peut être trouvée par l'exécution de celui-ci.

4 Automate Non-déterministe Fini

4.1 Définition

Une automate non-déterministe fini est une variété d'automate similaire aux ADF, moyennant quelques modifications. Un automate non-déterministe fini s'écrit également :

$$A = (Q, \Sigma, \delta, q_0, F)$$

mais avec :
- Q un ensemble fini d'états
- Σ un alphabet
- q_0 l'état initial
- $F \subseteq Q$ l'ensemble des états acceptants
- $\delta : Q \times \Sigma \cup \{\epsilon\} \to 2^Q$ où 2^Q est *l'ensemble des parties* de Q. Cela signifie qui la fonction δ retourne un ensemble d'états de Q

Dans la littérature **[?]** les automates non-déterministes finis sont divisés en deux groupes :

1. Ceux pour lequel au moins une transition de δ est définie pour ϵ.
2. Ceux pour lequel aucune transition n'est définie pour ϵ. En pratique, la définition de delta devient $\delta : Q \times \Sigma \to 2^Q$.

N'étant pas le sujet de ce document, ces deux ne reçoivent aucune distinction et sont tous deux notés ANF pour automate non-déterministe fini. Une transition sur ϵ est susceptible d'être définie pour tout ANF.

Exemple 4.1 *De la même façon que pour l'exemple 3.1 de la section 3.1, considérons un automate $A = (Q, \Sigma, q_0, \delta, F)$ défini comme suit :*
- $Q = \{q_0, q_1, q_2\}$
- $\Sigma = \{a, b, c\}$
- q_0 *est l'état du même nom*
- δ *est donnée par la table 3.*
- $F = \{q_2\}$

A est un ANF ; une colonne supplémentaire sert à représenter la transition sur ϵ.

	ϵ	a	b	c
$\to q_0$	$\{q_1, q_2\}$	$\emptyset$	$\{q_1\}$	$\{q_2\}$
q_1	$\emptyset$	$\{q_0\}$	$\{q_2\}$	$\{q_0, q_1\}$
q_2^*	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$

FIGURE 3: δ

De plus, A peut être représenté par un graphe suivant la même méthodologie que dans la sous-section 3.2 pour les ADF. Additionnellement, ϵ peut servir d'étiquette même s'il n'appartient pas à Σ.

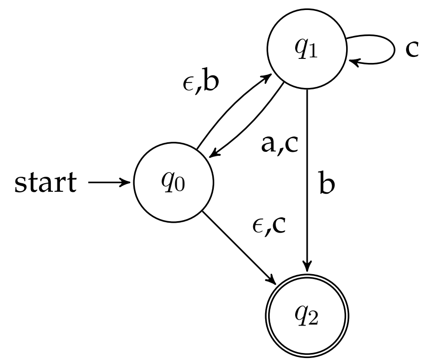


FIGURE 4: Automate A

4.2 Fermeture sur ϵ

Pour chaque état q d'un ANF, un ensemble d'états peut être atteint sans lire de symbole. Il s'agit de l'état en question et de tous ceux pouvant être atteint uniquement par des transitions sur ϵ. Cet ensemble s'appelle la *fermeture sur epsilon* : ECLOSE(q). Il peut être construit récursivement.

Soit un automate $A = (Q, \Sigma, q_0, \delta, F)$. Soit q un état dans Q.

Cas de base q est dans ECLOSE(q)

Pas de récurrence Si p est dans ECLOSE(q) et qu'il existe un état r tel quel $r \in \delta(p, \epsilon)$, alors r est dans ECLOSE(q)

Exemple 4.2 *Considérons l'automate A de l'exemple 4.1. Les différentes fermetures peuvent être calculées :*

— *ECLOSE(q_0) = $\{q_0, q_1, q_2\}$. En effet, q_0 appartient à sa fermeture, selon le cas de base. Aussi, $q_1, q_2 \in \delta(q_0, \epsilon)$*
— *ECLOSE(q_1)=$\{q_1\}$ par le cas de base.*
— *ECLOSE(q_2)=$\{q_2\}$ par le cas de base.*

4.3 Chemin

La notion de fermeture permet de faciliter l'expression d'une fonction de transition étendue pour un ANF, ce qui permet d'exprimer des chemins et donc le langage.

Soit un ANF $A = (Q, \Sigma, q_0, \delta, F)$. La fonction de transition étendue $\hat{\delta}$ retourne un ensemble d'états atteints par la lecture d'un mot depuis un état : $\hat{\delta}(q, w)$ est un ensemble d'états atteignables par un chemin formant le mot w, avec éventuellement des transitions sur ϵ.

$\hat{\delta}$ vaut, de façon récursive sur le mot w pour un état q :

Cas de base $\hat{\delta}(q, \epsilon)$=ECLOSE($q$). ECLOSE permet de calculer l'ensemble des états atteints uniquement par des transitions sur ϵ, ce qui correspond à ce cas de base

Pas de récurrence Supposons que $w = xa$. a est le dernier symbole de w et $\hat{\delta}(q, x)$ est défini par récurrence. Alors, pour obtenir $\hat{\delta}(q, w)$:

1. Posons $\{p_1, p_2, \ldots, p_k\} = \hat{\delta}(q, x)$. Ce sont les états atteints par la lecture de x, certains ont potentiellement été atteints par des transitions sur ϵ.
2. Posons $\{r_1, r_2, \ldots, r_m\} = \bigcup_{i=1}^{k} \delta(p_i, a)$. Ce sont les nouveaux états atteints par la lecture de a. Comme δ retourne un ensemble, l'union permet de regrouper les états en un seul ensemble.
3. Finalement, $\hat{\delta}(q, w) = \bigcup_{j=1}^{m}$ECLOSE($r_j$). Cette étape de fermeture permet d'ajouter les états décrivant la lecture de w suivie de transition sur ϵ, ce qui exprime toujours le mot w.

Le langage exprimé par un ANF $A = (Q, \Sigma, q_0, \delta, F)$ est alors défini par :

$$L(A) = \{w \in \Sigma^* | \hat{\delta}(q_0, w) \cap F \neq \emptyset\}$$

4.4 Transformation d'ANF à ADF

Cette section présente une méthode permettant de créer un ADF à partir d'un ANF.
Soit un ANF $A = (Q, \Sigma, q_0, \delta, F)$. Alors l'ADF équivalent

$$D = (Q_D, \Sigma, \delta_D, q_D, F_D)$$

est défini par :

— $Q_D = \{S | S \subseteq Q$ et S est *fermé sur epsilon*$\}$. Concrètement, Q_D est l'ensemble des partie des Q fermées sur ϵ. Cette fermeture s'écrit S=ECLOSE(S), ce qui signifie que chaque transition sur ϵ depuis un état de S mène à un état également dans S. L'ensemble $\emptyset$ est fermé sur ϵ.

— q_D=ECLOSE(q_0). L'état initial de D est l'ensemble des états dans la fermeture sur ϵ des états de A.

— $F_D = \{S | S \in Q_D$ et $S \cap F \neq \emptyset\}$ contient les ensembles dont au moins un état est acceptant pour A.

— $\delta_D(S, a)$ est construit, $\forall a \in \Sigma, \forall S \in Q_D$ par :
  1. Soit $S = \{p_1, p_2, \ldots, p_k\}$.
  2. Calculer $\bigcup_{i=1}^{k} \delta(p_i, a)$. Renommer cet ensemble en $\{r_1, r_2, \ldots, r_m\}$.
  3. Alors $\delta_D(S, a) = \bigcup_{j=1}^{m}$ ECLOSE(r_j).

Exemple 4.3 *Considérons l'automate $A = (Q, \Sigma, q_0, \delta, F)$ de l'exemple 4.1 et les fermetures calculées dans l'exemple 4.2.*

Alors, l'automate $D = (Q_D, \Sigma, \delta_D, q_D, F_D)$ est donné par :

— $Q_D = \{\emptyset, \{q_1\}, \{q_2\}, \{q_1, q_2\}, \{q_0, q_1, q_2\}\}$. *Les ensembles $\{q_0, q_1\}$ et $\{q_0, q_2\}$ sont des sous-ensembles de Q mais ne sont pas fermé sur ϵ.*

— $q_D = \{q_0, q_1, q_2\} =$ *ECLOSE(q_0).*

— $F_D = \{\{q_2\}, \{q_1, q_2\}, \{q_0, q_1, q_2\}\}$, *les ensembles contenant q_2, étant acceptant de A.*

— δ_D *est exprimé sur le graphe de la figure 5.*

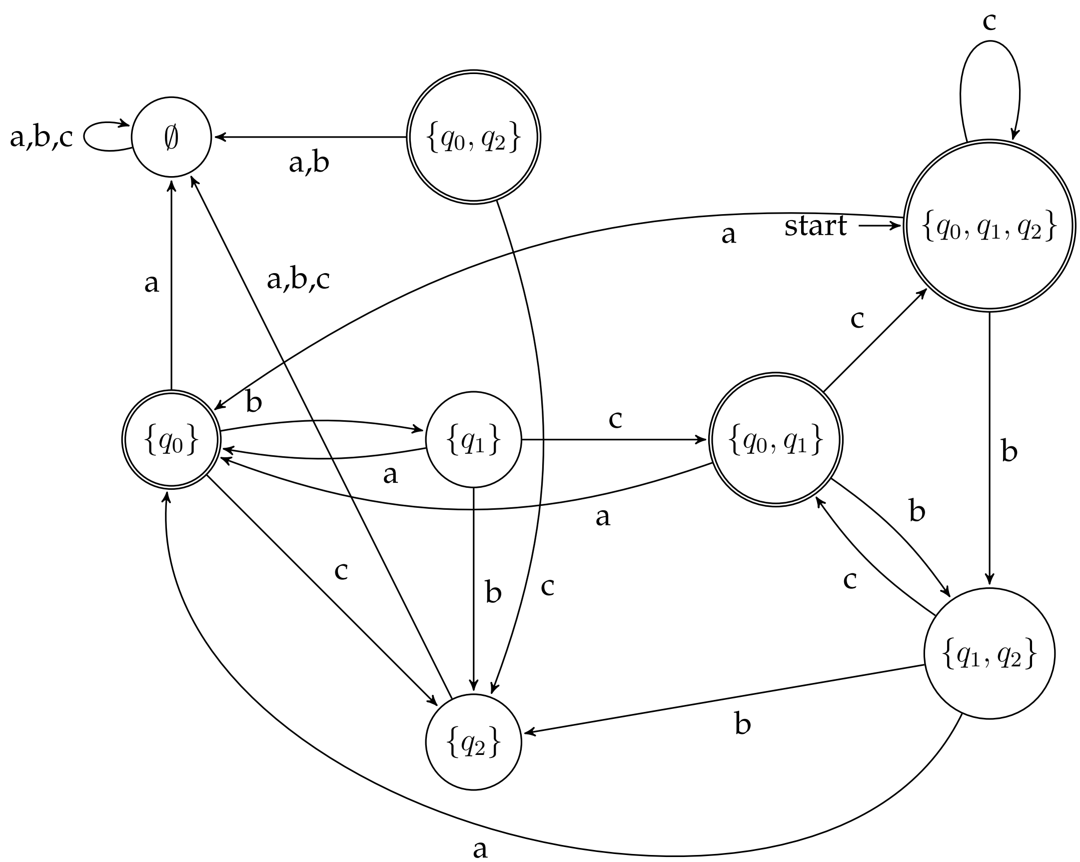


FIGURE 5: Automate D. De par la construction par les parties de Q, le nombre de parties est exprimé en exponentiel, d'où la complexité du graphe. Ici, $\{q_0, q_2\}$ n'est pas atteignable et peut être supprimé. De même $\emptyset$ est souvent omis pour clarifier la représentation.

Théorème 4.4 *Un langage L peut être représenté par un ANF si et seulement si il peut l'être par un ADF.*

Preuve 4.4.1 *Soit L un langage. Cette preuve étant une double implication, chacune peut être prouvée séparément.*

($\Leftarrow$) L peut être représenté par un ADF $\implies$ L peut être représenté par un ANF. Supposons qu'un automate $D = (Q_D, \Sigma, \delta_D, q_D, F_D)$ représente L : $L(D) = L$. L'ANF $A = (Q, \Sigma, q_0, \delta, F)$ correspondant est construit comme suit :

- *— $Q = \{\{q\} | q \in Q_D\}$*
- *— δ contient les transitions de D modifiées. Les objets retournés deviennent des ensembles d'états. C'est-à-dire, si $\delta_D(q, a) = p$ alors $\delta(q, a) = \{p\}$. De plus, pour chaque état $q \in Q_D$, $\delta(q, \epsilon) = \emptyset$.*
- *— $q_0 = \{q_D\}$*
- *— $F = \{\{q\} | q \in F_D\}$*

Dès lors, les transitions sont les mêmes entre D et A, mais A précise explicitement qu'il n'y a pas de transition sur ϵ. Comme A représente le même langage, un ANF représente L.

($\Rightarrow$) L peut être représenté par un ANF $\implies$ L peut être représenté par un ADF. Soit l'automate $A = (Q, \Sigma, q_0, \delta, F)$. Supposons qu'il représente L ($L = L(A)$). Considérons l'automate obtenu par la transformation détaillée à la section précédente 4.4 :

$$D = (Q_D, \Sigma, \delta_D, q_D, F_D)$$

Montrons que $L(D) = L(A)$. Pour ce faire, montrons que les fonctions de transition étendues sont équivalentes. Auquel cas, les chemins sont équivalents et donc les langages également. Montrons que $\hat{\delta}(q_0, w) = \hat{\delta}_D(q_D, w)$ pour tout mot w, par récurrence sur w.

Cas de base *Si $|w| = 0$, $w = \epsilon$. $\hat{\delta}(q_0, \epsilon) =$ ECLOSE(q), par définition de la fonction de transition étendue. q_D=ECLOSE(q_0) par la construction de q_D. Finalement, pour un ADF (ici, D), $\hat{\delta}(p, \epsilon) = p$, pour tout état p. Par conséquent, $\hat{\delta}_D(q_D, \epsilon) = q_D =$ ECLOSE(q_0) $= \hat{\delta}(q_0, \epsilon)$.*

Pas de récurrence *Supposons $w = xa$ avec a le dernier symbole de w. Notre hypothèse de récurrence est que $\hat{\delta}_D(q_D, x) = \hat{\delta}(q_0, x)$. Ce sont bien les mêmes objets car $\hat{\delta}_D$ retourne un état de D qui correspond à un ensemble d'états de A. Notons celui-ci $\{p_1, p_2, \ldots, p_k\}$. Par définition de $\hat{\delta}$ pour un ANF, $\hat{\delta}(q_0, w)$ est obtenu en :*

1. *Soit $\{r_1, r_2, \ldots, r_m\}$ donné par $\bigcup_{i=1}^{k} \delta(p_i, a)$, les états obtenus par la lecture du symbole a à partir de $\{p_1, p_2, \ldots, p_k\}$.*
2. *Alors $\hat{\delta}(q_0, w) = \bigcup_{j=1}^{m}$ECLOSE($r_j$). Un état atteint par la lecture de a l'est aussi par $a\epsilon$.*

D a été construit avec ces deux mêmes étapes pour $\delta_D(\{p_1, p_2, \ldots, p_k\}, a)$. Dès lors, $\hat{\delta}_D(q_D, w) = \delta_D(\{p_1, p_2, \ldots, p_k\}, a) = \bigcup_{j=1}^{k}$ECLOSE($p_j$)$= \hat{\delta}(q_0, w)$.

On a bien $\hat{\delta}_D(q_D, w) = \hat{\delta}(q_0, w)$. Les langages sont équivalents.

Complexité *La complexité d'une conversion ANF vers ADF peut être exprimée en fonction de n le nombre d'états de l'ANF. La taille de l'alphabet Σ est ici comptée comme une constante, elle est ignorée dans l'analyse grand-O. L'algorithme de conversion se fait en deux étapes. Le calcul de ECLOSE et la construction à proprement parler.*

— *ECLOSE : Pour chacun des n états, il y a au plus n^2 transitions à suivre sur ϵ pour construire la fermeture. Ceci représente le cas où tous les états sont reliés avec tous les autres par des transitions sur ϵ. Le coût de cet algorithme est dès lors de $n * \mathcal{O}(n^2) = \mathcal{O}(n^3)$.*

— *Construction : Posons s le nombre d'états dans l'ADF (qui, dans le pire des cas vaut $s = 2^n$ par la construction des sous-ensembles). La création d'un état est en temps $\mathcal{O}(n)$, correspondant au plus à n états de l'ANF. Pour ce qui est des transitions, pour chacun des s nouveaux états, ECLOSE contient au plus n éléments. Chacune des n^2 transitions de l'ADF sont alors suivies pour chaque symbole $a \in \Sigma$. Le coût de construction d'une transition est alors de $n * \mathcal{O}(n^2) = \mathcal{O}(n^3)$ auquel vient s'ajouter $\mathcal{O}(n^2)$, négligeable, pour l'union de l'ensemble obtenu.*

La complexité totale est $\mathcal{O}(n^3) + s * \mathcal{O}(n^3) = \mathcal{O}(s * n^3) = \mathcal{O}(2^n * n^3)$. Le détail est donné sur s car, comme mentionné par Hopcroft et Al. [1], en pratique le nombre de l'état dans l'ADF obtenu est rarement de l'ordre de 2^n, typiquement de l'ordre de n.

Complexité *La conversion d'un ADF $A = (Q, \Sigma, q_0, \delta, F)$ vers un ANF consiste au remplacement d'états par des ensembles d'états. Si l'ADF contient n états, cette étape est en $\mathcal{O}(n)$. De plus, une colonne pour ϵ doit être ajoutée à la table de transition (pour la fonction δ), et ce pour chacun des états. Cette étape se fait également en $\mathcal{O}(n)$.*

La complexité totale d'une conversion d'un ADF vers un ANF est en $\mathcal{O}(n)$.

5 Opérations sur un automate

5.1 Équivalence avec une expression régulière

Proposition 5.1 *Un langage peut être exprimé par un automate déterministe fini si et seulement si il peut être exprimé par une expression régulière.*

Cette proposition étant une double implication, elle est vraie si les deux implications le sont. Soit un langage L.

Théorème 5.2 *Il existe un automate déterministe A tel que $L(A) = L \implies$ il existe une expression régulière E telle que $L(E) = L$.*

Preuve 5.2.1 *Supposons qu'il existe un ADF $A = (Q, \Sigma, q_0, \delta, F)$ tel que $L(A) = L$. Q étant un ensemble fini, on peut définir sa cardinalité : $|Q| = n$. Supposons que ses états soient nommés $\{1, 2, \ldots, n\}$. Il est possible de construire des expressions régulières par induction sur le nombre d'états considérés.*

Posons E_{ij}^k l'expression régulière exprimant un langage constitué des mots w tels que $\delta(i, w) = j$ et qu'aucun état intermédiaire n'ait un nombre supérieur à k. Il n'y a pas de contrainte sur i et j.

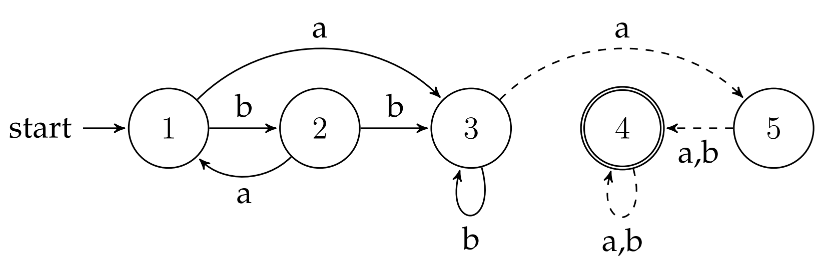


FIGURE 6: Exemple : automate mettant $E_{1,3}^3$ en évidence

L'exemple ci-dessus illustre ce fait qu'aucun état supérieur à k ne peut faire partie des intermédiaires. Dans cet exemple, $E_{5,4}^3$ tolère la transition de 5 à 4 bien que supérieure à k : ce ne sont pas des intermédiaires. Construisons le langage par induction sur les états autorisés.

Cas de base $k = 0$. *Comme tout état est numéroté 1 ou plus, aucun intermédiaire n'est accepté. La première possibilité est $i = j$ et indique un chemin de longueur 0. Auquel cas l'expression régulière représentant un chemin sans symbole est ϵ. Ce chemin doit être ajouté aux possibilités si $i = j$. La deuxième possibilité est $i \neq j$. Alors les chemins possibles ne se composent que d'un arc allant directement de i à j. Pour les construire :*

Pour chaque paire i, j :

— *Il n'existe pas de symbole a tel que $\delta(i,a) = j$. Alors, $R^0_{ij} = \emptyset(+\epsilon)$*
— *Il existe un unique symbole a tel que $\delta(i,a) = j$. Alors, $R^0_{ij} = a(+\epsilon)$*
— *Il existe des symboles $a_1, a_2, \ldots, a_k$ tels que $\forall l \in \{1, \ldots, k\}, \delta(i, a_l) = j$. Alors, $R^0_{ij} = a_1 + a_2 + \cdots + a_k(+\epsilon)$*

Pas de récurrence *Supposons qu'il existe un chemin allant de i à j ne passant par aucun état ayant un numéro supérieur à k. La première possibilité est que le-dit chemin ne passe pas par k. Alors, le mot représenté par ce chemin fait partie du langage de E^{k-1}_{ij}. Seconde possibilité, le chemin passe par k une ou plusieurs fois comme représenté à la figure 7.*

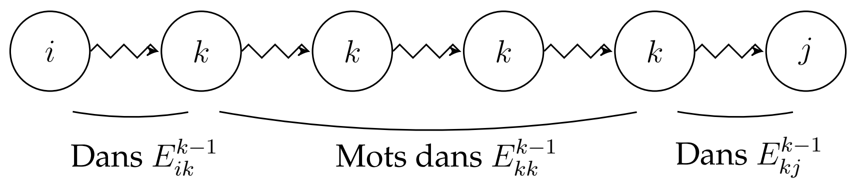


FIGURE 7: Un chemin de i à j peut être découpé en différent segment en fonction de k

Auquel cas, ces chemins sont composés d'une sous-chemin donnant un mot dans E^{k-1}_{ik}, suivi d'un sous-chemin donnant un ou plusieurs mots dans E^{k-1}_{kk} et finalement un mot dans E^{k-1}_{kj}.

En combinant les expressions des deux types, on obtient :

$$E^k_{ij} = E^{k-1}_{ij} + E^{k-1}_{ik}(E^{k-1}_{kk}) * E^{k-1}_{kj}$$

En commençant cette construction sur E^n_{ij}, comme l'appel se fait toujours à des chaînes plus courtes, éventuellement on retombe sur le cas de base. Si l'état initial est numéroté 1, alors l'expression régulière E exprimant L est l'union $(+)$ des E^n_{1j} tel que j est un état acceptant.

Complexité *Soit un ADF $A = (Q, \Sigma, q_0, \delta, F)$ comportant n états. La complexité peut se décomposer en deux facteurs : la longueur d'une expression régulière et le nombre de celles-ci.*

A chacune des n itérations (ajoutant progressivement des nouveaux états admis pour état intermédiaire), la longueur de l'expression peut quadrupler : elle est exprimée par 4 facteurs. Ainsi, après n étapes, cette expression peut être de taille $\mathcal{O}(4^n)$.

Le nombre d'expressions à construire, lui, est décomposable en deux facteurs également : le nombre d'itérations et celui de paires i, j possibles. Le premier facteur est n, quand aux paires, leur nombre s'exprime par n^2. n^3 expressions sont construites.

En regroupant ces deux facteurs, on obtient $n^3\mathcal{O}(4^n) = \mathcal{O}(n^3 4^n)$. Comme n correspond au nombre d'états, si la transformation se fait depuis un ANF, via un ADF, vers une expression régulière, la complexité devient doublement exponentielle, la première transformation étant elle-même exponentielle en le nombre d'états de l'ANF.

Exemple 5.3 *Construction d'une expression régulière à partir de l'automate de la figure suivante :*

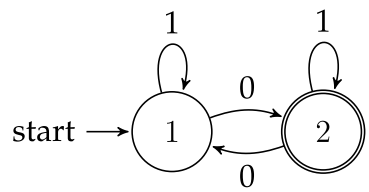


FIGURE 8: Un automate acceptant tout mot ayant un nombre impair de 0

La construction par récurrence commençant avec $k = 0$ le processus peut être représenté par des tableaux correspondant à différents k de façon croissante.

Première itération *Dans la première itération, chaque expression se résume à un des trois cas de base, avec éventuellement ϵ si $i = j$ pour l'expression analysée.*

	Cas de base
E^0_{11}	$1 + \epsilon$
E^0_{12}	0
E^0_{21}	0
E^0_{22}	$1 + \epsilon$

Seconde itération *Ensuite, l'état 1 est autorisé comme état intermédiaire : $k = 1$. Ayant potentiellement un état intermédiaire, la formule de récurrence est utilisée.*

	Formule de récurrence	Détail	Simplification
E^1_{11}	$E^0_{11} + E^0_{11}(E^0_{11})^* E^0_{11}$	$(1+\epsilon) + (1+\epsilon)(1+\epsilon)^*(1+\epsilon)$	1^*
E^1_{12}	$E^0_{12} + E^0_{11}(E^0_{11})^* E^0_{12}$	$0 + (1+\epsilon)(1+\epsilon)^*0$	1^*0
E^1_{21}	$E^0_{21} + E^0_{21}(E^0_{11})^* E^0_{11}$	$0 + 0(1+\epsilon)^*(1+\epsilon)$	01^*
E^1_{22}	$E^0_{22} + E^0_{21}(E^0_{11})^* E^0_{12}$	$(1+\epsilon) + 0(1+\epsilon)^*0$	$1 + 01^*0$

Troisième itération *A la troisième itération, l'état 2 est autorisé comme état intermédiaire.*

	Formule de récurrence	Détail	Simplification
E^2_{11}	$E^1_{11} + E^1_{12}(E^1_{22})^* E^1_{21}$	$1^* + 1^*0(1+01^*0)^*01^*$	$1^* + 1^*0(1+01^*0)^*01^*$
E^2_{12}	$E^1_{12} + E^1_{12}(E^1_{22})^* E^1_{22}$	$1^*0 + 1^*0(1+01^*0)^*(1+01^*0)$	$1^*0(1+01^*0)^*$
E^2_{21}	$E^1_{21} + E^1_{22}(E^1_{22})^* E^1_{21}$	$01^* + (1+01^*0)(1+01^*0)^*01^*$	$(1+01^*0)^*01^*$
E^2_{22}	$E^1_{22} + E^1_{22}(E^1_{22})^* E^1_{22}$	$(1+01^*0) + (1+01^*0)(1+01^*0)^*(1+01^*0)$	$(1+01^*0)^*$

Pour obtenir une expression régulière correspondant à l'automate, on s'intéresse à celle qui décrit un chemin entre l'état initial (1) et les états acceptants (uniquement 2 ici). Dès lors, $L(1^*0(1+01^*0)^*) = L$.

Cette expression régulière $1^*0(1+01^*0)^*$ décrit bien un nombre impair de 0. Il en faut absolument un, et tout ajout supplémentaire de se fait par paire. Cela correspond bien à un nombre impair.

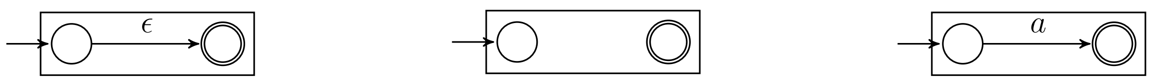
Théorème 5.4 *(⇐) Il existe une expression régulière E telle que $L(E) = L \implies$ il existe un automate déterministe A tel que $L(A) = L$.*

Preuve 5.4.1 *Comme tout ANF a un ADF équivalent (théorème 4.4), montrer qu'une expression régulière E a un ANF équivalent est suffisant pour obtenir cet ADF.*

Soit L. Soit E une expression régulière telle que $L(E) = L$. On peut construire l'automate récursivement sur la définition des expressions régulières à la section 2.4. Cette preuve par récurrence repose sur trois invariants portant sur chaque ANF construit :

1. *Il y a un unique état acceptant*
2. *Aucune transition ne mène à l'état initial*
3. *Aucune transition ne part de l'état acceptant*

Cas de base *Les ANF de la figure 9 représentent les automates correspondant aux trois cas de base.*

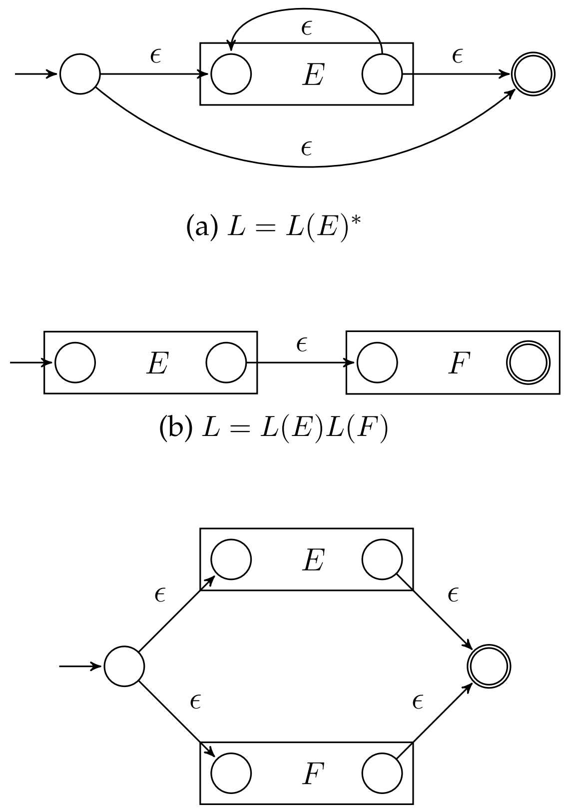


(a) $L = \{\epsilon\}$ (b) $L = \emptyset$ (c) $L = \{a\}$

FIGURE 9: Blocs de base pour la construction d'un automate à partir d'une expression régulière

En effet, l'automate (a) correspond à l'expression ϵ : le seul arc de l'état initial à un état final est ϵ. L'automate (b) ne propose pas d'arc atteignant l'état final. Aucun mot n'appartient au langage d'où la construction de $\emptyset$. Finalement, (c) propose un arc pour a, donnant le seul mot a comme faisant partie du langage, faisant de a une expression régulière équivalente. De plus, ces automates respectent bien l'invariant de récurrence proposé.

Pas de récurrence *Les ANF* abstraits *de la figure 10 représentent la façon dont un automate peut être construit récursivement en fonction des règles de récurrence des expressions régulières. Ces ANF sont abstraits car le contenu d'un bloc R ou S n'est pas représenté explicitement. Cependant, celui-ci respecte les invariants de récurrence.*



(a) $L = L(E)^*$

(b) $L = L(E)L(F)$

(c) $L = L(E) + L(F)$

FIGURE 10: Enchaînement de blocs pour une construction récursive

Les quatre règles de récurrence sur une expression régulière permettent de construire les automates :

— *Pour une expression régulière de forme (E), le langage $L(E)$ étant équivalent à $L((E))$, l'automate construit pour E reste valable.*

— *L'expression régulière est de forme E^*. Par induction, il existe un automate exprimant le même langage que E. L'automate pour E^* est construit comme en (a). Cet automate comprend un arc ϵ de l'état initial à l'état acceptant pour représenté le cas E^0. Ensuite, un arc ϵ permet de concaténer plus chemins dans E, donnant des mots représentés par E^1,E^2,E^3,... Le tout complétant l'ensemble des mots possibles des $L(E)^*$. On a bien $L(E^*) = L(E)^*$.*

— *L'expression régulière est de forme EF. Par induction, il existe des automates représentants les mêmes langages que E et F et respectant notre invariant. L'automate abstrait (b) représente cette concaténation. En effet, un mot de cet automate doit se composer d'un mot $v \in L(E)$ et d'un mot $w \in L(F)$. Les mots possibles sont alors de la forme $v\epsilon w$. Donc (b) représente bien, selon la définition d'un expression régulière $L(EF) = L(E)L(F)$.*

— *L'expression régulière est de forme $E + F$. Alors, comme mis en évidence par l'automate abstrait (c), il existe des automates correspondants aux expression E et F. Par cette construction, en particulier les transitions sur ϵ, permettent à c de représenter tout mot de $L(E)$ ou $L(F)$. Le langage est alors, en concordance avec la définition d'une expression régulière $L(E + F) = L(E) \cup L(F)$.*

Les automates (a), (b) et (c) respectent bien l'invariant de récurrence : pas de transition vers l'état *initial, un seul état acceptant n'ayant pas de transition sortante. Chaque automate abstrait pour E ou F peut lui même être construit récursivement jusqu'au cas de base.*

Complexité *Soit une expression régulière E de longueur n représentant un langage L. Si un* arbre syntaxique *est créé pour E, il est possible de construire un ANF pour L en $\mathcal{O}(n)$. En effet :*

— *Cas de base : n ANFs sont créés. Cependant, chacun est constitué de 2 états et au plus 1 transition. Ces nombres sont des constantes. Ce cas de base est effectué en $\mathcal{O}(n * 3) = \mathcal{O}(n)$*

— *Récurrence : L'arbre syntaxique requiert au plus n lectures d'opération de récurrence pour fusionner les n ANF en un seul. Cependant, chacune de ses opération implique au plus la création de 2 états et 4 arcs. Ces nombres sont des constantes. La récurrence s'effectue en $\mathcal{O}(n*6) = \mathcal{O}(n)$*

La complexité totale de cette conversion est en $\mathcal{O}(n)$ vers un ANF. La conversion vers un ADF, comme mentionné dans la section 4 peut quand à elle être exponentielle.

5.2 Équivalence d'états

Certains états d'un automate peuvent être *équivalents* selon la relation R_M . Celui-ci peut alors être simplifié. Une façon de détecter ces équivalences est de construire un tableau via l'*algorithme de remplissage de tableau*.

Celui-ci détecte les paires *différenciables*, récursivement sur un automate $A = (Q, \Sigma, q_0, \delta, F)$. Un paire $\{p, q\}$ est différenciable s'il existe un mot w tel qu'un chemin $\hat{\hat{\delta}}(p, w)$ mène à un état acceptant et $\hat{\hat{\delta}}(q, w)$ mène à un état non-acceptant ou vice-versa. w sert alors de *mot témoin*.

***TODO** : Environnement algorithmicx ?*

Cas de base : Si p est un état acceptant et que q ne l'est pas, alors la paire $\{p, q\}$ est différenciable. Le mot témoin est ϵ.

Pas de récurrence : Soient p, q des états de Q et un symbole $a \in \Sigma$ tel que $\delta(p, a) = r$ et $\delta(q, a) = s$. Si r et s sont différenciables, alors p et q le sont aussi. En effet, il existe un mot *témoin* w qui permet de différencier r et s. Alors le mot aw est le mot témoin qui permet de différencier p et q.

Théorème 5.5 *Si deux états ne sont pas distingués par l'algorithme de remplissage de tableau, les états sont équivalents (ils respectent la relation R_M).*

Preuve 5.5.1 *Considérons un automate déterministe fini quelconque $A = (Q, \Sigma, q_0, \delta, F)$. Supposons par l'absurde qu'il existe une paire d'états $\{p, q\}$ tels que :*

1. *p et q ne sont pas distingués par l'algorithme de remplissage de table.*
2. *Les états ne sont pas équivalents, $\not p R_M q$. Par extension, il existe un mot témoin w différenciant p et q.*

Une telle paire est une mauvaise paire. *Si il y a des mauvaises paires, chacune distinguée par un mot témoin, il doit exister un paire distinguée par le mot témoin le plus court. Posons $\{p, q\}$ comme étant cette paire et $w = a_1 a_2 \ldots a_n$ le mot témoin le plus court qui les distingue. Dès lors, soit $\hat{\hat{\delta}}(p, w)$ est acceptant, soit $\hat{\hat{\delta}}(q, w)$ l'est, mais pas les deux.*

Ce mot w ne peut pas être ϵ. Auquel cas, la table aurait été remplie dès l'étape d'induction de l'algorithme. La paire $\{p, q\}$ ne serait pas une mauvaise paire, ne respectant pas l'hypothèse 1.

w n'étant pas ϵ, $|w| \geq 1$. Considérons les états $r = \delta(p, a_1)$ et $s = \delta(q, a_1)$. Ces états sont différenciés par $a_2 a_3 \ldots a_n$ car $\hat{\delta}(p, w) = \hat{\delta}(r, a_2 a_3 \ldots a_n)$ et $\hat{\delta}(q, w) = \hat{\delta}(s, a_2 a_3 \ldots a_n)$ et p et q sont différenciables.

Cela signifie qu'il existe un mot plus petit que w qui différencie deux états : le mot $a_2 a_3 \ldots a_n$. Comme on a supposé que w est le mot le plus petit qui différencie une mauvaise paire, r et s ne peuvent pas être une mauvaise paire. Donc, l'algorithme a du découvrir qu'ils sont différenciables.

Cependant, le pas de récurrence impose que $\delta(p, a_1)$ et $\delta(q, a_1)$ mènent à deux états différentiables implique que p et q le sont aussi. On a une contradiction de notre hypothèse : $\{p, q\}$ n'est pas une mauvaise paire.

Ainsi, s'il n'existe pas de mauvaise paire, c'est que chaque paire différenciable est reconnue par l'algorithme.

Exemple 5.6 *Voici une application de cet algorithme sur l'automate A_2, version réduite de l'automate A_1 de la figure 2.*

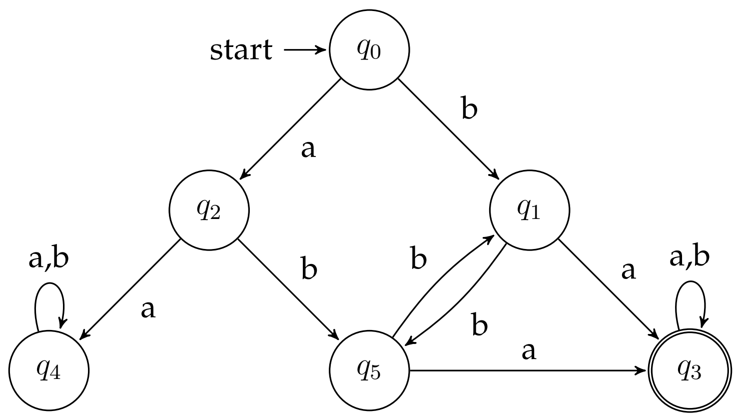


FIGURE 11: Automate A_2

La première étape est de remplir la table avec l'algorithme précédant. Tout état est distinguable de q_3 : il est le seul état acceptant. 5 cases peuvent déjà êtres cochées. Le reste de la table est remplie par induction.

q_1	X				
q_2	X	X			
q_3	X	X	X		
q_4	X	X	X	X	
q_5	X		X	X	X
	q_0	q_1	q_2	q_3	q_4

FIGURE 12: Table filling pour A_2, décelant des équivalences d'états

Complexité *Considérons n le nombre d'états d'un automate, et k la taille de l'alphabet Σ supporté.*

Si il y a n états, il y a $\binom{n}{2}$ soit $\frac{n(n-1)}{2}$ paires d'états. A chaque itération (sur l'ensemble de la table), il faut considérer chaque paire, et vérifier si un de leur successeurs est différentiable. Cette étape prend *au plus $\mathcal{O}(k)$ pour tester chaque successeurs potentiel (en fonction du symbole lu). Ainsi, une itération sur la table se fait en $\mathcal{O}(kn^2)$. Si une itération ne découvre pas de nouveaux état différentiable s'arrête. Comme la table a une taille en $\mathcal{O}(n^2)$ et qu'à chaque étape un élément au minimum doit y être coché, la complexité totale de l'algorithme est en $\mathcal{O}(kn^4)$.*

Cependant, il existe des pistes d'amélioration. La première est d'avoir, pour chaque paire $\{r, s\}$ une liste des paire $\{p, q\}$ qui, pour un même symbole, mènent à $\{r, s\}$. On dit de ces paires qu'elles sont dépendantes. Si la paire $\{r, s\}$ est marquée comme différenciable, leurs paires dépendantes seront de facto différenciables.

Cette liste peut être construite en considérant chaque symbole $a \in \Sigma$ et ajoutant les paires $\{p, q\}$ à chacune de leur dépendance $\{\delta(p, a), \delta(q, a)\}$. Cette étape prend au plus $k.\mathcal{O}(n^2) = \mathcal{O}(kn^2)$. (Le nombre de symboles multiplié par le nombre de paires à considérer).

Ensuite, il suffit de partir des cas initiaux (se reposant sur le cas de base de l'algorithme), et de marquer tous leurs états dépendants comme différentiables, tout en ajoutant leur propre liste à chaque fois. La complexité de cette exploration est bornée par le nombre d'éléments dans une liste et le nombre de listes. Respectivement, k et $\mathcal{O}(n^2)$, ce qui donne $\mathcal{O}(kn^2)$ pour cette exploration.

La complexité totale revient à $\mathcal{O}(kn^2)$.

5.3 Équivalence d'automates

Considérons les automates A_H et A_I donnés dans les figures 13 et 14

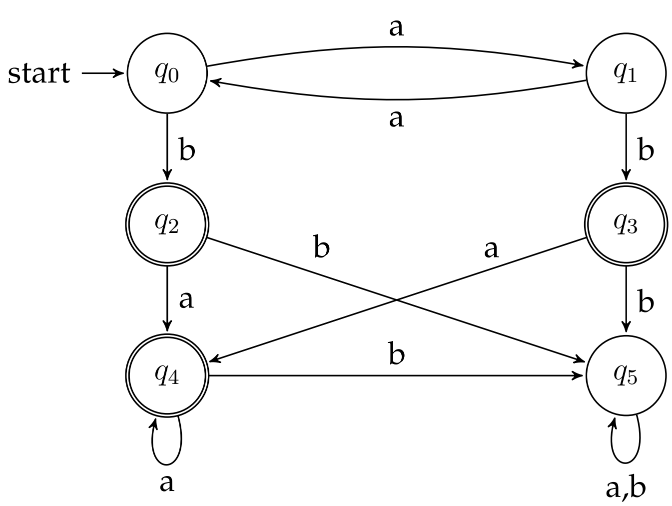


FIGURE 13: Automate A_H, du livre d'Hopcraft et al. de 1979[2] (Fig3.2)

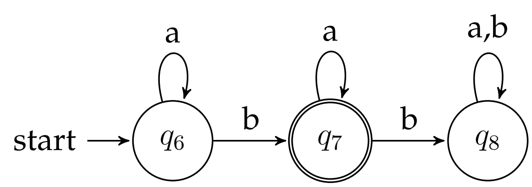


FIGURE 14: Automate A_I, provenant également de [2]. Les états ont été renommés.

Il est possible de remplir un tableau via l'algorithme éponyme. Pour ce faire, les deux automates sont considérés comme un seul dont les états sont disjoints.

q_1								
q_2	x	x						
q_3	x	x						
q_4	x	x						
q_5	x	x	x	x	x			
q_6			x	x	x	x		
q_7	x	x				x	x	
q_8	x	x	x	x	x		x	x
	q_0	q_1	q_2	q_3	q_4	q_5	q_6	q_7

FIGURE 15: Tableau généré par l'application de l'algorithme sur A_H et A_I

De cette table, toujours grâce aux conclusions précédentes, il est possible d'extraire des classes d'équivalences :
— $C_0 = \{q_0, q_1, q_6\}$
— $C_1 = \{q_2, q_3, q_4, q_7\}$
— $C_2 = \{q_5, q_8\}$

En particulier, la classe C_0 souligne que les états initiaux sont équivalents. Cela signifie, par définition, que tout mot w lu en partant d'un de ces états sera soit accepté dans les deux automates, soit refusé dans les deux. A_H et A_I définissent donc le même langage.

Complexité *Reposant sur la construction de la table d'équivalence d'états, la complexité est en $\mathcal{O}(kn^2)$, avec k la taille de l'alphabet et n le nombre d'états. L'étape supplémentaire, la lecture de cette table, est en temps constant et n'impacte pas la complexité.*

Les différentes notions liées à l'égalité : les propriétés de réflexivité, transitivité et symétrie ont été démontrées dans la section 3.5.

5.4 Minimisation d'automate

La minimisation d'automate se fait en deux étapes :

1. Se débarrasser de tous les états injoignables : ils ne participent pas à la construction du langage représenté
2. Grâce aux équivalences d'états trouvées grâce à l'algorithme de remplissage de tableau défini au point 5.2, construire un nouvel automate.

Soit un automate déterministe fini $A = (Q, \Sigma, q_0, \delta, F)$. Les états non-atteignables peuvent être supprimés de Q et de δ.

Pour minimiser cet automate, il faut :

1. Générer la table de différenciation.
2. Séparer Q en classes d'équivalences
3. Construire l'automate canonique $C = (Q_C, \Sigma, \delta_C, q_C, F_C)$:
   — Soit S une des classes d'équivalence obtenues par la table de différenciation.
   — Ajouter S à Q_C et à F_C si S contient un état acceptant : $q \in S, q \in F$.

— Si S contient q_0 l'état initial de A, alors S est q_C l'état initial de C.
— Pour un symbole $a \in \Sigma$, alors il doit exister une classe d'équivalence T tel que pour chaque état $\forall q \in S, \delta(q, a) \in T$. Si ce n'est pas le cas, c'est que deux états p et q dans S mènent à différentes classes d'équivalences. Or, ces deux états sont différenciables, et ne pourraient pas appartenir tous deux à S par construction. On peut écrire $\delta_C(S, a) = T$. Pour rappel, la fonction δ est définie pour tout état et tout symbole. Rien n'empêche $T = S$.

Exemple 5.7 *Considérons l'automate A_1 représenté à la figure 2. En supprimant l'état q_6 qui n'est pas atteignable, on obtient l'automate A_2 de la figure 11.*

Le tableau de la figure 12 sert d'exemple pour l'algorithme de remplissage de tableau, sur A_2. A_3.

En appliquant l'algorithme, qui peut se résumer intuitivement à fusionner les états équivalents, on obtient l'automate A_3 de la figure 16.

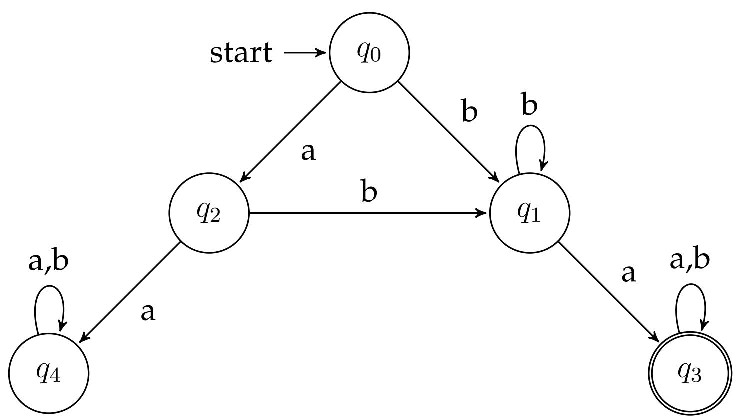


FIGURE 16: Automate A_3

Une expression régulière ($(b + ab)b^*a(a + b)^*$) peut être déduite pour L grâce à cet automate. Cette expression régulière est celle de l'exemple 2.6

Théorème 5.8 *Soit un ADF A et soit C l'automate construit par cet algorithme de minimisation. Aucun automate équivalent à A n'a moins d'états que C. De plus, chaque automate ayant autant d'états que C peut être transformé en celui-ci par homomorphisme.*

Preuve 5.8.1 *Prouvons que l'algorithme de minimisation fourni un automate minimum (il n'en existe aucun comportant moins d'états pour un même langage) Soient un ADF A et C l'automate obtenu par l'algorithme de minimisation. Posons que C comporte k états.*

Par l'absurde, supposons qu'il existe M un ADF minimisé équivalent à A mais comptant moins d'états que C. Posons qu'il en comporte $l < k$. Appliquons l'algorithme de remplissage de table sur C et M, comme s'ils étaient un seul ADF, comme proposé dans la section 5.3. Les états initiaux sont équivalents (pas différentiables) puisque $L(C) = L(M)$. Dès lors, les successeurs pour chaque symboles sont eux aussi équivalent. Le cas contraire impliquerait que états initiaux sont différentiables, ce qui n'est pas le cas. De plus, ni C ni M n'ont un état inaccessible, sinon il pourrait être éliminé, résultant en un automate comportant moins d'états pour un même langage. Soit p un état de C. Soit un mot $a_1a_2 \ldots a_i$, *qui mène de l'état initial de C à p. Alors, il existe un état q de M équivalent à p. Puisque les états initiaux sont équivalents, et que par induction, les états obtenus par la lecture d'un symbole le sont aussi, l'état q dans M obtenu par la lecture du mot $a_1 a_2 \ldots a_i$ est équivalent à p. Ceci signifie que tout état de C est équivalent à au moins un état de M. Or, $lk > l$. Cela signifie qu'il doit exister au moins deux états de C équivalents à un même état de M et donc équivalent entre eux. Il y a la contradiction : par construction, les états de C sont tous différentiables les uns des autres. La supposition de l'existence de M est fausse. Il n'existe pas d'automate équivalent à A comportant moins d'états que C.*

Preuve 5.8.2 *Prouvons que tout automate minimal pour un langage est C, à un isomorphisme sur les noms des états près.*

Soit A un ADF pour un langage L. Soient C un ADF obtenu par l'algorithme de minimisation et M un automate minimal comportant autant d'états que C.

Comme mentionné dans la preuve précédente, il doit y avoir une équivalence 1 à 1 entre chaque état de C et de M. (Au minimum 1 et au plus 1). De plus, aucun état de M ne peut être équivalent à 2 états de C, selon le même argument.

Dès lors, l'automate minimisé, dit canonique *est unique à l'exception du renommage des différents états.*

5.5 Construction d'automate depuis un langage

Soit le langage $A_N = \{w|w \text{ fini par b et ne contient pas bb}\}$ défini sur $\Sigma_N = a, b$.

On peut diviser les mots en 3 ensembles :

— W_0 le sous-ensemble des mots ne finissant pas le symbole b

— W_1 celui des mots finissant par le symbole b mais ne contenant pas bb

— W_2 celui des mots contenant au moins bb

Il y a d'autres façons de construire des sous-ensembles, mais celle-ci à l'avantage de rendre la question de l'appartenance à L_N triviale : un mot appartient au second ensemble si et seulement si il fait partie du langage, par définition.

De plus, tous les éléments d'un sous-ensemble respectent la relation R_L entre eux. (R_L : $xR_Ly \Leftrightarrow \forall z \in \Sigma^*, xz \in L \Leftrightarrow yz \in L$). Cela en fait des classes d'équivalence sur cette relation.

Cela peut être démontré pour chaque sous-ensemble :

— Soient $x, y \in W_0$. Soit $z \in \Sigma^*$. Dès lors, si $xz \in L_N$, c'est que z fini par b mais ne contient pas bb, et donc $yz \in L_N$. Si $yz \in L_N$, le même argument peut être appliqué.

— Soient $x, y \in W_1$. Soit $z \in \Sigma^*$. Dès lors, si $xz \in L_N$, c'est que z ne commençait pas le symbole b et ne contenait pas bb, yz ne contiendra donc pas bb, puisque cette chaîne n'est ni dans z ni dans y, ni a cheval sur les deux, z ne commençant pas par b. Ainsi, $yz \in L_N$. Si $yz \in L_N$, le même argument peut être appliqué.

— Soient $x, y \in W_2$. Soit $z \in \Sigma^*$. Comme x contient déjà bb, $x \notin L_N$ et, a fortiori, $xz \notin L_N$. Comme la prémisse est fausse, l'implication $xz \in L \Rightarrow yz \in L$ est vraie. La même logique peut être appliquée à partir de y pour justifier l'implication inverse.

De plus, ces sous-ensembles sont disjoints. Cela peut se prouver en invalidant la relation pour certains éléments entre eux, mais dans ce cas-ci, la propriété est assurée par définition.

Ceci revient à démontrer que W_0, W_1, W_2 sont des classes d'équivalence. De plus, R_L respecte la congruence à droite, comme démontré dans la preuve du théorème de Myhill-Nérode.

Ce même théorème donne une méthode pour construire un automate : prendre un représentant pour chaque classe et en faire un état.
- $\Sigma = \{a, b\}$ est connu.
- $Q = \{[[\epsilon]]\ [[b]], [[bb]]\} = \{q_\epsilon, q_b, q_{bb}\}$
- $q_0 = q_\epsilon$
- $F = \{q_b\}$ l'union des classes acceptant
- δ défini en utilisant des exemples tirés des classes d'équivalence.

Ce qui donne l'automate de la figure 17

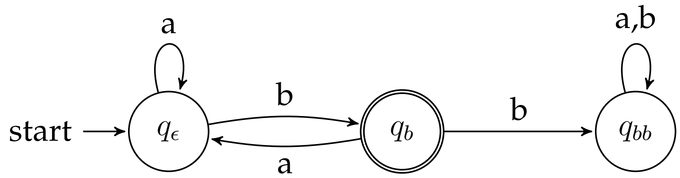


FIGURE 17: Automate A_N, exemple d'une thèse[3]

Cet automate est bien une représentation du langage L_N. Seul un mot finissant par b mais ne contenant pas bb se termine à l'état q_b.

6 Théorème de Myhill-Nérode

Cette section apporte le détail sur la relation de Myhill-Nérode, en prouve les propriétés avant d'en faire l'usage dans le théorème du même nom.

6.1 Relation de Myhill-Nérode

Soit un langage L sur un alphabet Σ.

Soit la relation R_L portant sur deux mots (ne faisant pas nécessairement partie de L). Deux mots x et y respectent la relation de Myhill-Nérode R_L si

$$\forall z \in \Sigma^*, xz \in L \Leftrightarrow yz \in L$$

Intuitivement, deux mots sont en relation si pour tout mot qu'on leur concatène, les deux mots résultants sont tous deux dans le langage ou non.

Lemme 6.1 *Cette relation est une relation d'équivalence. De plus, elle respecte la congruence à droite. C'est à dire que si xR_Ly, alors pour tout symbole $a \in \Sigma$, xaR_Lya*

Preuve 6.1.1 (Equivalence et Congruence à droite) *Dire d'une relation qu'elle décrit une équivalence, revient à dire qu'elle est réflexive, transitive et symétrique*
- *R_L est réflexive. Soit $x \in \Sigma^*$. Soit $z \in \Sigma^*$. Montrer que xR_Lx est vrai revient à montrer que $xz \in L \Leftrightarrow xz \in L$ est vrai. R_L est donc réflexive.*

— *R_L est symétrique. Soient $x, y \in \Sigma^*$ tels que xR_Ly. Soit $w \in \Sigma^*$. Montrer que yR_Lx revient à montrer que $yw \in L \Leftrightarrow xw \in L$. Or, par hypothèse, $xz \in L \Leftrightarrow yz \in L$, qui peut s'écrire aussi $yz \in L \Leftrightarrow xz \in L$ pour tout $z \in \Sigma^*$, et en particulier $z = w$.*
— *R_L est transitive. Soient $x, y, u \in \Sigma^*$ tels que xR_Ly et yR_Lz. Soit $w \in \Sigma^*$. Comme $xz \in L \Leftrightarrow yz \in L$ et $yz \in L \Leftrightarrow uz \in L$ pour tout $z \in \Sigma^*$ (par hypothèse), c'est vrai en particulier pour $z = w$. Dès lors, $xw \in L \Leftrightarrow yw \in L$et $yw \in L \Leftrightarrow uw \in L$. Par transitivité de l'implication, on obtient $xw \in L \Leftrightarrow uw \in L$, à savoir xR_Lu.*
— *R_L est congruente à droite. Soient $x, y \in \Sigma^*$ tels que xR_Ly. Soit $a \in \Sigma$. Par hypothèse, $xz \in L \Leftrightarrow yz \in L$ pour tout $z \in \Sigma^*$. Cela doit donc être vrai en particulier pour le mot $z = aw$ avec w quelconque. En remplaçant dans l'hypothèse, on obtient $xaw \in L \Leftrightarrow yaw \in L$. Ce qui montre que xaR_Lya.*

6.2 Théorème de Myhill-Nerode

Théorème 6.2 *Les 3 énoncés suivants sont équivalents :*

1. *Un langage $L \subseteq \Sigma^*$ est accepté par un DFA*
2. *L est l'union de certaines classes d'équivalence d'index fini respectant une relation d'équivalence et de congruence à droite*
3. *Soit la relation d'équivalence $R_L : xR_Ly \Leftrightarrow \forall z \in \Sigma^*, xz \in L \Leftrightarrow yz \in L$ (la relation de Myhill-Nérode définie précédemment). R_L est d'index fini.*

Preuve 6.2.1 *La preuve d'équivalence se fait en prouvant chaque implication de façon cyclique :*

$(1) \rightarrow (2)$ *Supposons que (1) soit vrai, c'est à dire que le langage L est accepté par un automate déterministe fini A. Considérons la relation d'équivalence R_M étant vraie pour les mots x, y si $\hat{\delta}(q_0, x) \in F \iff \hat{\delta}(q_0, y) \in F$. Elle a été définie en 3.5. Il y est prouvé qu'elle est congruente à droite. Comme il y a au plus une classe d'équivalence pour R_M par état de A. Comme ce nombre d'états est fini, R_M est d'index fini. De plus, L est l'union de classes contenant un mot w tel que $\hat{\delta}(q_0, w) \in F$, (or, ce chemin retourne un état. Il s'agit donc d'une union des classes correspondant aux états acceptants).*

$(2) \rightarrow (3)$ *Montrons que pour toute relation E satisfaisant (2), chaque classe est intégralement contenue dans une seule classe de R_L. Ces classes étant d'index fini, c'est un argument suffisant pour déduire que R_L est d'index fini. Considérons x, y tels que xEy. Comme E est congruente à droite, pour tout mot $z \in \Sigma^*$, on sait que $xzEyz$. Comme L est un union de ces classes d'équivalence, $xzEyz$ implique que $xz \in L \Leftrightarrow yz \in L$, ce qui revient à xR_Ly. Cela signifie que tout mot dans la classe d'équivalence de x définie par E se retrouve dans la même classe d'équivalence que x par R_L. Ce qui permet de conclure que chaque classe d'équivalence de E est contenue dans une classe d'équivalence de R_L.*

$(3) \rightarrow (1)$ *Considérons la relation R_L définie précédemment, et déduisons-en Q' les classes d'équivalence sur L et $[[x]]$ l'élément(la classe) de Q' qui contient x. Puisque R_L a été démontré comme congruent à droite, on peut définir des transitions : $\delta'([[x]], a) = [[xa]]$. En choisissant un élément y dans $[[x]]$ (ce qui signifie que xR_Ly), on obtient $\delta'([[x]], a) = [[ya]]$. Sauf que par définition, xR_Ly signifie qu'en y ajoutant n'importe quel mot z, xz et yz appartiennent tous deux où non à L. C'est vrai en particulier pour $z = az'$. Ainsi, xaz' et yaz' appartiennent tous deux à L ou non. Ce qui signifie que xaR_Lya et donc $[[xa]] = [[ya]]$. Posons $q_0' = [[\epsilon]]$ et $F' = \{[[x]] | x \in L\}$. Tous ces éléments forment l'automate*

$M' = (Q', \Sigma, \delta', q'_0, F')$. Il est déterministe par la définition de δ', fini car Q' est fini par construction (le nombre de classes d'équivalence est fini). De plus, il accepte L puisque $\delta'(q'_0, x) = [[x]]$, ce qui signifie que $x \in L(M')$ si et seulement si $[[x]] \in F'$, qui a été défini comme tel.

Corrolaire 6.2.2 *Grâce à la preuve du théorème de Myhill-Nérode, en particulier la justification partant de la relation d'équivalence R_L pour montrer que la langage $L \subseteq \Sigma^*$ est accepté par un DFA, on a une méthode pour construire un automate à partir d'un langage.*

7 Algorithme d'Angluin

L'algorithme d'Angluin repose, en plus des éléments précédents sur quatre concepts :
— Une table d'observation
— La relation R_O, se basant sur la table d'observation et semblable à la relation R_L
— La propriété de fermeture (closure en anglais)
— La propriété de cohérence (consistence en anglais)

Cette section commence par décrire cette table en 7.1, la relation R_O en 7.2, la fermeture en 7.3, la cohérence en 7.4.

Une fois toutes ces bases posées, une exécution de l'algorithme sur un exemple est proposée en 7.5, suivie du fonctionnement formel de l'algorithme et des preuves sur son exactitude et sa complexité en 7.6, 7.7 et 7.8.

7.1 Table d'observation

7.2 Relation R_O

7.3 Fermeture

La propriété de fermeture (closure) s'exprime mathématiquement par

$$\forall u \in R, \forall a \in \Sigma, \exists v \in R, uaR_Ov$$

En pratique, pour vérifier cette propriété, il suffit de de suivre cet algorithme, expliqué de façon visuelle sur la table O :

Algorithme 2 Vérification de la fermeture

Promet: si la fermeture est respectée ou non

```
pour chaque élément w de la section R faire
  pour chaque symbole a dans Σ faire
    si wa est dans R alors
      continuer
    sinon
      {wa est dans R.Σ par construction}
      si La ligne de wa dans T est différente de celle de w alors
        retourner Faux
      fin si
    fin si
  fin pour
fin pour
retourner Vrai
```

7.4 Cohérence

La propriété de cohérence (consistence) se définit mathématiquement comme

$$\forall u, v \in R, uR_O v \Rightarrow \forall a \in \Sigma, uaR_O va$$

Concrètement, il s'agit de prendre deux mots (u, v) dans R ayant la même ligne dans T et vérifier, pour chaque symbole (a), s'ils (ua, va) ont la même ligne dans T.

7.5 Exemple

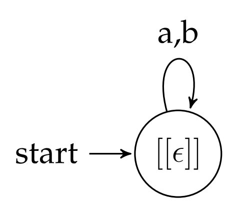
Considérons l'automate A_3 de la figure 16 construit à la section 5.4 sur la minimisation.

***TODO** : Marquer la différence entre R_L et R_O*

7.5.1 Première itération

L'algorithme d'Angluin précise, pour son cas de base, une initialisation de la table T avec les ensembles R et S contenant uniquement ϵ. Le champ $R.\{a, b\}$ $(R.\Sigma)$ est rempli via des requête d'appartenance sur les mots a et b.

O_0	ϵ
ϵ	0
a	0
b	0



Automate O_0

L'étape suivante consiste à vérifier la *closure* de la table d'observation O_0. Mathématiquement :

$$\forall u \in R, \forall a \in \Sigma, \exists v \in R, uaR_Lv$$

Intuitivement, pour chaque symbole (ici, $\{a, b\}$, et ce sera vrai jusqu'à la dernière itération), tout mot candidat (dans R, la partie supérieure de la table) doit se retrouver, complété de ce symbole, dans une classe d'équivalence d'un autre candidat de R. Ici, de toute évidence, les mots a et b sont dans la même classe d'équivalence que ϵ. Dès lors, la propriété de *closure* est respectée.

Si la *closure* est respectée, alors la question de la *consistence* (cohérence) se pose. Mathématiquement :

$$\forall u, v \in R, uR_Lv \Rightarrow \forall a \in \Sigma, uaR_Lva$$

Intuitivement, si deux candidats semblent être dans la même classe d'équivalence (leur lignes dans la table supérieure sont identiques), alors pour n'importe quel symbole, les deux nouveaux mots sont également dans une même classe d'équivalence (leur lignes, potentiellement dans la partie inférieure de la table, sont identiques). N'ayant qu'un seul candidat, cette propriété est forcément respectée (R_L est réflexive).

Les deux propriétés étant respectées, les classes d'équivalences sont calculées (trivialement ici), et un automate O_0 est proposé à l'enseignant pour vérification.

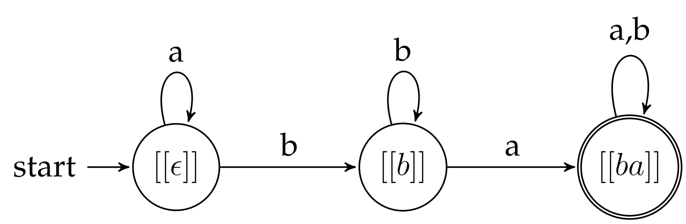
Sur cette itération, un automate initial a été proposé, et aucun état final ne pouvant être atteint avec un seul symbole, la version est minime.

7.5.2 Seconde itération

L'enseignant répond que non, les automates ne sont pas équivalents. Il fourni le contre-exemple ba. Comme il est rejeté par O_0, cela signifie qu'il est accepté par A_4. Une nouvelle table est alors construite, en ajoutant ba et ses préfixes (ici, juste b) à R. $R.\Sigma$ est calculé et les mots n'ayant pas encore reçu une valeur dans T sont soumis à l'enseignant pour un test d'appartenance.

O_1	ϵ
ϵ	0
b	0
ba	1
a	0
bb	0
baa	1
bab	1

O_2	ϵ	a
ϵ	0	0
b	0	1
ba	1	1
a	0	0
bb	0	1
baa	1	1
bab	1	1



Automate O_2

Comme pour la première itération, la *fermeture* est testée, suivie de la *cohérence*. Celle-ci n'est pas respectée : si on considère les mots ϵ et b, qui ont la même ligne dans la table T ($\epsilon R_O b$), le symbole a, on obtient les mots a et ba qui n'ont pas la même ligne : ($a \not R_O ba$). Le symbole a est alors ajouté à S et une nouvelle table O_2 est calculée.

La fermeture étant déjà vérifiée, la question de la cohérence est reposée, et cette fois-ci elle est vérifiée ; l'automate est construit et proposé à l'enseignant.

Sur cette itération, l'algorithme a reçu le mot ba comme étant accepté. Il a du ajouter a à S pour permettre de différencier certains états. L'automate se voit ajouter les états $[[b]]$ et $[[ba]]$.

7.5.3 Troisième itération

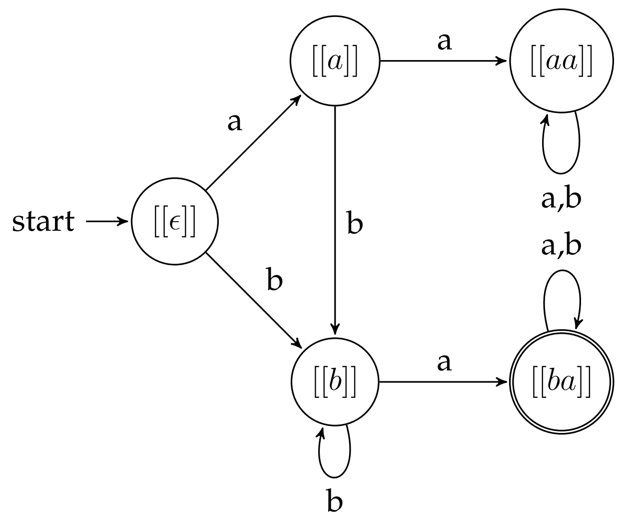
Suivant toujours l'algorithme de comparaison d'automates détaillé dans la section **??**, l'enseignant découvre qu'ils sont différents.

Il sort le contre-exemple $aaba$. Si c'est un contre-exemple et qu'il est accepté par O_2, c'est qu'il ne l'est pas (0) par A_4. Une nouvelle table O_3 doit être construite.

O_3	ϵ	a
ϵ	0	0
a	0	0
b	0	1
aa	0	0
ba	1	1
aab	0	0
$aaba$	0	0
ab	0	1
bb	0	1
aaa	0	0
baa	1	1
bab	1	1
$aabb$	0	0
$aabaa$	0	0
$aabab$	0	0

O_4	ϵ	a
ϵ	0	0
a	0	0
b	0	1
aa	0	0
ab	0	1
ba	1	1
aab	0	0
$aaba$	0	0
bb	0	1
aaa	0	0
aba	1	1
abb	0	1
baa	1	1
bab	1	1
$aabb$	0	0
$aabaa$	0	0
$aabab$	0	0

O_5	ϵ	a
ϵ	0	0
a	0	0
b	0	1
aa	0	0
ab	0	1
ba	1	1
aab	0	0
aba	1	1
$aaba$	0	0
bb	0	1
aaa	0	0
abb	0	1
baa	1	1
bab	1	1
$aabb$	0	0
$abaa$	1	1
$abab$	1	1
$aabaa$	0	0
$aabab$	0	0



Automate O_5

Ayant reçu $aaba$, ce mot et tous ses préfixes sont ajoutés à la table. L'extension $R.\Sigma$ est recalculée et la table O_3 est construite.

Ensuite, la question de la *fermeture* est posée. Un manquement est détecté : le mot a. En effet, en lui ajoutant le symbole b, on obtient ab qui n'est ni dans R ni en relation R_O avec a. ab est alors ajouté à R, et $R.\Sigma$ est étendu. La nouvelle table, O_4 est de nouveau testée.

O_4 ne respecte pas la fermeture : le mot ab, agrémenté du symbole a donne le mot aba, qui n'est ni dans R ni en relation avec ab. Le mot est ajouté à R, et la table est étendue. La nouvelle table, O_5 est à la fois fermée et cohérente.

L'automate O_5 est alors proposé à l'enseignant pour vérification. Celui-ci est accepté (isomorphe à A_4). L'algorithme s'arrête et un automate minimal pour le langage a été construit.

7.6 Algorithme

7.7 Preuve

7.8 Complexité

Références

[1] J. E. HOPCROFT, *Introduction to Automata Theory, Languages, and Computation (2nd Edition)*, Addison Wesley, nov 2000.

[2] J. E. HOPCROFT AND J. D. ULLMAN, *Introduction to automata theory, languages and computation. adison-wesley*, Reading, Mass, (1979).

[3] D. NEIDER, *Applications of automata learning in verification and synthesis*, PhD thesis, Hochschulbibliothek der Rheinisch-Westfälischen Technischen Hochschule Aachen, 2014.