UMONS

Faculté
des Sciences

Automates

Étudiant : Benjamin André
Directrice :Véronique Bruyère
5 mai 2020

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

Table des matières

1 Introduction

Ce document a pour but d'amener à la compréhension de l'algorithme d'Angluin. Pour ce faire, des bases théoriques seront posées dans la section 2. Ensuite, quelques algorithmes utilisés par la méthode d'Angluin sont présentés et analysés dans la section 4. Dans la section 3, certaines notions théoriques supplémentaires sont apportées, se reposant sur les différents algorithmes du point précédant. Finalement, dans la section **??**, l'algorithme d'Angluin est expliqué et illustré avec un exemple.

2 Bases théoriques

Cette section pose les bases théoriques et les conventions nécessaires à la compréhension des sections suivantes. De plus, certaines propriétés importantes sont énoncées et démontrées.

2.1 Alphabet

Un alphabet, nommé par convention Σ est un ensemble fini et non vide de symboles. Voici certains exemples d'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

Comme Σ est un ensemble, on peut définir Σ^k, qui donne des k-uples de symboles, appartenant tous à Σ.

Un mot w de taille $|w| = k$ est un ensemble de symboles provenant de Σ^k. Dans le cas particulier où $k = 0$, on note le mot vide (sans symbole) $w = \epsilon$.

De façon générale, w est un mot sur Σ si il existe k tel que $w \in \Sigma^k$. Par convention, les mots sont nommés par une lettre minuscule, souvent w, x, y, z.

L'ensemble de tous ces mots possible sur Σ est noté Σ^*. Cet ensemble est infini.

2.3 Langage

Un langage L est défini sur un alphabet Σ. L est un ensemble de mots sur cet alphabet : $L \subseteq \Sigma^*$. Comme Σ^* contient une infinité de mots, L est susceptible de ne pas être fini non plus.

Par exemple, par rapport à tous les mots disponibles avec les lettres de l'alphabet latin (Σ^*), seulement certains font partie de la langue française (L).

L peut être défini :

— en énumérant les mots en faisant partie : $L = \{12, 35, 42, 7, 0\}$
— via un notation ensembliste : $L = \{0^k 1^j | k + j = 7\}$ ou $L = \{w | w \text{ est un mot français}\}$

Dans ces deux cas, Σ est souvent implicite.

2.3.1 Problème

Une notion liée aux langages est celle de problème. Ce que nous appelons couramment problème peut être exprimé en terme d'appartenance d'un mot w à un langage L sur un alphabet Σ.

Par exemple, prenons l'alphabet $\Sigma = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$. Considérons ensuite le langage $L = \{w | \text{le nombre représenté par } w \text{ est pair}\}$. Demander si un mot w appartient à L revient à résoudre le problème sur le test de parité de w. (Le cas échéant, soit w ne représente pas un nombre, soit il représente un nombre impair).

2.4 Expression régulière

Une expression régulière est une façon efficace de définir un langage. La construction de l'expression se fait de façon inductive.

Le cas de base est l'expression $e = a, a \in \Sigma$. Dès lors $L_e = a$ où L_e est le langage donné par l'expression e.

Les autres règles sont inductives. Par ordre de priorité :

— $e = (e_0)$ La mise en évidence. Ici, $L_e = \{w | w \in L_{e_0}\}$

— $e = e_0 a, a \in \Sigma$. La concaténation. Ici, $L_e = \{wa | w \in L_{e_0}\}$

— $e = e_0 + e_1$. L'union. Ici, $L_e = L_{e_0} \cup L_{e_1}$

— $e = e_0^*$. La fermeture. Intuitivement, il s'agit de tous les mots qui peuvent être formé par une concaténation de mots définis par e_0, éventuellement aucun. Ici, $L_e = \{w_0 w_1 ... w_k | k \in \mathbb{N}, w_0, w_1, ..., w_k \in L_{e_0}\}$

— $e = e_0^+$. La fermeture non nulle. Il s'agit de la fermeture mais avec toujours au moins un mot venant du langage défini par e_0. Ici, $L_e = \{w_0 w_1 ... w_k | k \in \mathbb{N}^0, w_0, w_1, ..., w_k \in L_{e_0}\}$

Par exemple, on peut écrire l'expression $e_B = (1 + 01)1^*0(0 + 1)^*$.

Les mots suivant en feraient partie :
- 10
- 010
- 0110
- 0111110
- 0101101

Ceux-ci n'en feraient pas partie
- 00
- 1
- 01
- 0101
- 11

Un langage pouvant être écrit sous la forme d'une expression régulière est appelé langage régulier. Un des conséquences de cette propriété est qu'il peut être représenté par un automate déterministe fini. ***TODO** : a prouver*

2.5 Automate déterministe fini

Soit un ensemble de symboles Σ. Soient $\Sigma^* = \{a_1 a_2 a_3 ... a_n | a_1, a_2, a_3, ..., a_n \in \Sigma\}$, l'ensemble des mots de taille arbitraire qu'il est possible de former à partir de Σ et $|w|, w \in \Sigma$ la longueur de w, le nombre de symboles utilisés. Si $|w| = 0$, on note $w = \epsilon$.

Un automate est défini par $A = (Q, \Sigma, q_0, \delta, F)$ où

— Q est un ensemble d'états, différenciés par leur indice $q_1, q_2, ..., q_n$ ou $n = |Q|$.

— Σ est un ensemble de symboles

— $q_0 \in Q$ est l'état initial
— $\delta : Q \times \Sigma \to Q$ est la fonction de transition. A partir d'un état de Q, en fonction d'un symbole, elle retourne un nouvel état faisant partie de Q.
— $F \subseteq Q$ est un ensemble d'état finaux.

Exemple : Soient
— $\Sigma = \{0, 1\}$
— $Q = \{q_a, q_b, q_c, q_d, q_e, q_f, q_g\}$
— $q_0 = q_a$
— $F = \{q_d\}$

Pour obtenir un automate, il manque la description de δ. Une façon efficace de noter cette fonction de transition est à l'aide d'une table dont les lignes reprennent des éléments de Q, les colonnes des symboles de Σ, et les cases des éléments de Q. Ainsi, on peut lire la relation $\delta : Q \times \Sigma \to Q$ en prenant une ligne et une colonne. De plus, via cette notation, Q et Σ sont explicites. En dénotant l'état initial par $\to$ et les états acceptants par $*$, on obtient une définition complète d'un automate.

	0	1
$\to q_a$	q_c	q_b
q_b	q_d	q_f
q_c	q_e	q_f
q_d	q_d	q_d
q_e	q_e	q_e
q_f	q_d	q_b
q_g	q_e	q_f

FIGURE 1: La table de transitions δ_B

L'automate peut aussi être représenté graphiquement :

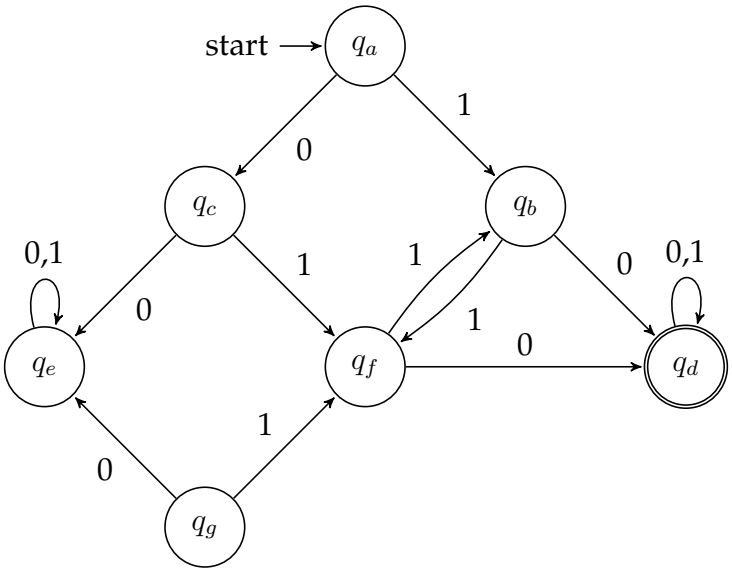

FIGURE 2: Automate A_B, exemple personnel

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. q_0 et F sont respectivement représenté par la flèche entrante et le double cercle.

2.5.1 Chemin

Pour faire le lien avec la notion de langage, il doit exister une façon pour l'automate de représenter quels mots sont acceptés ou non.

Pour ce faire, la fonction δ peut être étendue à un chemin w :

— Si $|w| \leq 1$, $\hat{\delta}(x, w) = \delta(x, w)$

— Sinon, c'est que w peut s'écrire $au, u \in \Sigma^*$. Alors, $\hat{\delta}(x, w) = \hat{\delta}(x, au) = \hat{\delta}(\delta(x, a), u)$

2.5.2 Langage défini par un automate

Le langage représenté par un automate A peut alors se définir comme les mots menant de l'état initial a un état acceptant :

$$\{w \in \Sigma^* | \hat{\delta}(q_O, w) \in F_A\}$$

Ainsi, pour tester l'appartenance d'un mot w à un langage L défini par l'automate A, il suffit de test $\hat{\delta}(q_O, w) \in F_A$.

2.5.3 La relation R_M

Soit un automate M. 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. Il s'agit d'une relation d'équivalence. En effet, cette relation est :

— **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.

— De plus, la relation est **congruente à droite :** 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,

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

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

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

Deux informations importantes découlent des ces caractéristiques :

1. Les états forment des classes d'équivalence.
2. Tout état dans une classe d'équivalence mène, pour un même symbole, à une même classe d'équivalence.

3 Preuves

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.

3.1 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 3.1 *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 3.1.1 (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.*

3.2 Théorème de Myhill-Nerode

Théorème 3.2 *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 3.2.1 *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 2.5.3. 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 3.2.1 *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.*

4 Algorithmes

Cette section repose sur les bases théoriques, principalement sur l'automate et la relation R_M, ainsi que le théorème de Myhill-Nérode, pour donner l'algorithme de remplissage de table (Table Filling Algorithme). A partir de celui-ci, les questions de minimisation, équivalence et constructions d'automates sont explorées.

4.1 Table Filling Algorithm

Le *Table Filling Algorithm* permet, pour un automate, de déterminer quels états sont équivalents. Il repose sur la définition de la relation R_M définie en 2.5.3

4.1.1 Construction de la table

L'idée est de construire, par induction, une table nous disant pour chaque paire d'état si ceux-ci sont équivalents où non, suivant la relation R_M définie précédemment :

Cas de base : Si p est un état final et que q ne l'est pas, alors la paire $\{p, q\}$ est différentiable.

Induction : Soient p, q des états tels qu'il existe un symbole a qui donne $\delta(p, a) = r$ et $\delta(q, a) = s$. Si r et s sont différentiables, 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 permet de différencier p et q.

Théorème 4.1 *Si deux états ne sont pas distingués par l'algorithme de remplissage de table, les états sont équivalents.*

Preuve 4.1.1 *Considérons un automate déterministe fini quelconque $A = (Q, \Sigma, \delta, q_0, F)$, et faisons une preuve par l'absurde.*

Supposons 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, c'est à dire différentiables.

L'hypothèse deux implique qu'il existe un mot $w \in \Sigma^*$ tel que de $\hat{\delta}(p, w)$ et $\hat{\delta}(q, w)$ un et un seul soit un état final.

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 $w = a_1 a_2 \dots a_n$ le mot témoin le plus court qui les distingue. Encore une fois, un seul de $\hat{\delta}(p, w)$ et $\hat{\delta}(q, w)$ est acceptant.*

Ce mot w ne peut pas être ϵ. Auquel cas, la table aurait été remplie dès l'étape d'induction de l'algorithme.

Ce mot w doit forcément être de taille ≥ 1 s'il n'est pas ϵ. Considérons $r = \delta(p, a_1)$ et $s = \delta(q, a_1)$. Ces états sont différenciés par $a_2 a_3 \dots a_n$ puisque cette chaîne mène aux mêmes états que $\hat{\delta}(p, w)$ et $\hat{\delta}(q, w)$. Mais dans ce cas, cela signifie qu'il existe un mot plus petit que w qui différencie deux états. 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 bad pair. Donc, l'algorithme a du découvrir qu'ils sont différentiables.

Mais le pas d'induction stipule clairement que comme $\delta(p, a_1)$ et $\delta(q, a_1)$ mènent à deux états différentiables, p et q le sont aussi. On a une contradiction sur l'existence des mauvaises paires.

Ainsi, s'il n'y en a pas, c'est que chaque paire différentiable est reconnue par l'algorithme.

4.1.2 Exemple

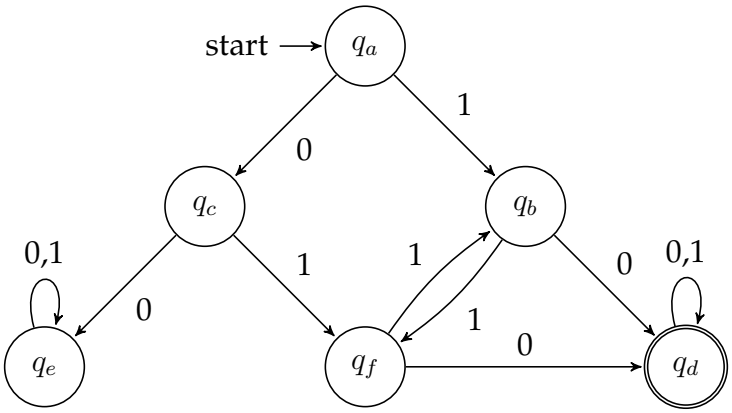

FIGURE 3: Automate A_2

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

B	x				
C	x	x			
D	x	x	x		
E	x	x	x	x	
F	x		x	x	x
	A	B	C	D	E

FIGURE 4: Table filling pour A_2, décelant des équivalences d'états

4.1.3 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 $O(k)$ pour tester chaque successeurs potentiel (en fonction du symbole lu). Ainsi, une itération sur la table se fait en $O(kn^2)$. Si une itération ne découvre pas de nouveaux état différentiable s'arrête. Comme la table a une taille en $O(n^2)$ et qu'à chaque étape un élément au minimum doit y être coché, la complexité totale de l'algorithme est en $O(kn^4)$.

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 $k.O(n^2) = O(kn^2)$. (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 $O(n^2)$, ce qui donne $O(kn^2)$ pour cette exploration.

La complexité totale revient à $O(kn^2)$.

4.2 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 au TFA déifni au point 4.1, construire un nouvel automate.

Ces étapes vont être accompagnées d'un exemple, à savoir l'automate A_1 représenté à la figure 5.

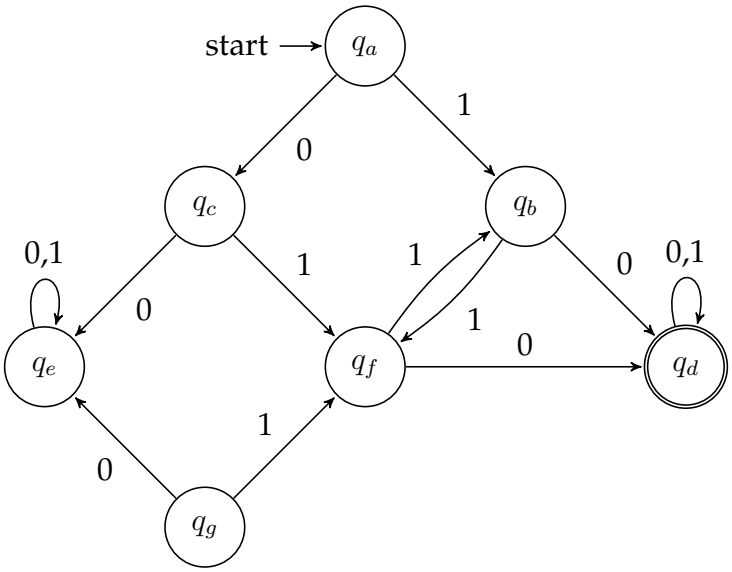

FIGURE 5: Automate A_1

L'état q_g n'est pas atteignable : il peut être simplement supprimé. On obtient ainsi l'automate A_2 qui a servi d'exemple pour le TFA, représenté à la figure 3.

4.2.1 Minimisation par table de différenciation

Pour minimiser l'automate $A_2 = (Q, \Sigma, \delta, q_0, F)$, il faut :

1. Générer la table de différenciation (qui, pour cet exemple, est à la figure 4)
2. Séparer Q en classes d'équivalences
3. Construire l'automate canonique A_3 :
   — Soit S une des classes d'équivalence
   — Soit γ la fonction de transition sur S. Pour un symbole $a \in \Sigma$, alors il doit exister une classe d'équivalence T tel que pour chaque état q dans S, $\delta(q, a) \in T$. Sinon, c'est que deux états p et q dans S menant à différentes classes d'équivalences. Ces deux états sont différenciables, et ne pourraient pas appartenir tous deux à S par construction. On peut écrire $\gamma(S, a) = T$.
4. L'état initial de A_3 est la classe d'équivalence contenant l'état initial de A_2 (dans notre exemple, l'état s'y trouve seul)
5. Les états finaux (F) de A_3 sont les classes d'équivalences qui contenaient des états acceptants de A_2.

La table de la figure 4. Peut servir de base à la construction du nouvel automate suivant cet algorithme.

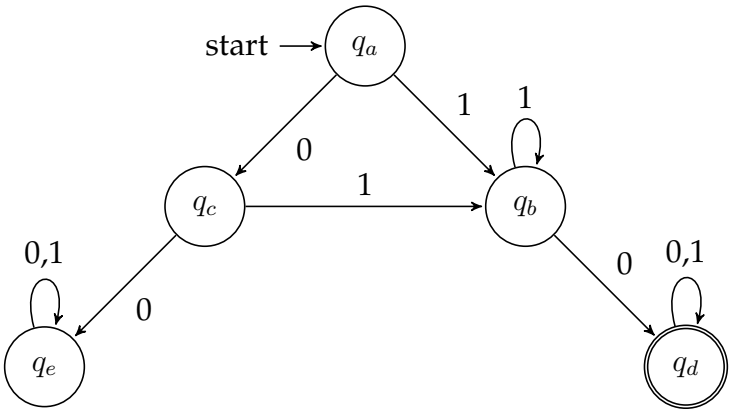

FIGURE 6: Automate A_3

Une expression régulière ($(1+01)1^*0(0+1)^*$) peut être déduite pour L grâce à cet automate.

4.3 Équivalence d'automates

Considérons les automates A_H et A_I donnés dans les figures 7 et 8

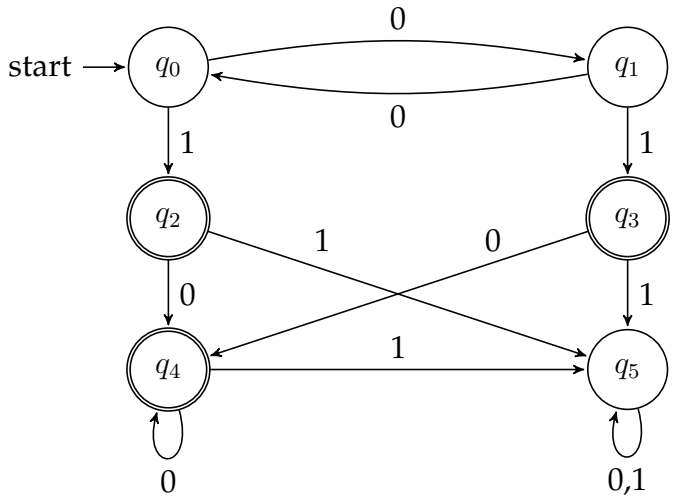

FIGURE 7: Automate A_H, exemple d'un livre de référence[1] (Fig3.2)

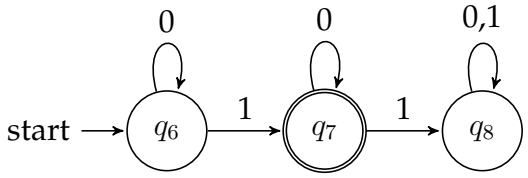

FIGURE 8: Automate A_I, provenant également de [1]

Ces deux automates sont-ils équivalents ? Grâce à l'algorithme de remplissage de table précédant, il est possible de tester l'équivalence d'états. Pour cela, il suffit de considérer les deux automates précédents comme un seul.

	q_0	q_1	q_2	q_3	q_4	q_5	q_6	q_7
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

FIGURE 9: Table Filling, décelant des équivalences d'états entre les deux automates

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 de cet état sera soit accepté dans les deux automates, soit refusé dans les deux. A_H et A_I définissent donc le même langage.

4.4 Construction d'automate depuis un langage

Soit le langage $A_N = \{w | w$ 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 10

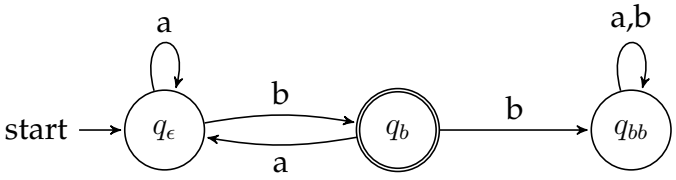

FIGURE 10: Automate A_N, exemple d'une thèse[2]

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.

Références

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

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