Chapitre 9

Automates

Dans ce chapitre, nous présentons les bases de la théorie des automates finis. Nous montrons l'équivalence entre les automates finis et les automates finis déterministes, puis nous étudions des propriétés de fermeture. Nous prouvons le théorème de Kleene qui montre que les langages reconnaissables et les langages rationnels sont une seule et même famille de langages. Nous prouvons l'existence et l'unicité d'un automate déterministe minimal reconnaissant un langage donné, et nous présentons l'algorithme de Hopcroft de minimisation.

Introduction

Les automates finis constituent l'un des modèles de calcul les plus anciens en informatique. A l'origine, ils ont été conçus et employés comme une tentative de modélisation des neurones ; les premiers résultats théoriques, comme le théorème de Kleene, datent de cette époque. Parallèlement, ils ont été utilisés en tant qu'outils de développement des circuits logiques. Les applications les plus courantes sont à présent le traitement de texte, dans son sens le plus général. L'analyse lexicale, la première phase d'un compilateur, est réalisée par des algorithmes qui reproduisent le fonctionnement d'un automate fini. La spécification d'un analyseur lexical se fait d'ailleurs souvent en donnant les expressions rationnelles des mots à reconnaître. Dans le traitement de langues naturelles, on retrouve les automates finis sous le terme de réseau de transitions. Enfin, le traitement de texte proprement dit fait largement appel aux automates, que ce soit pour la reconnaissance de motifs – qui constitue l'objet du chapitre suivant – ou pour la description de chaînes de caractères, sous le vocable d'expressions régulières. De nombreuses primitives courantes dans les systèmes d'exploitation modernes font appel, implicitement ou explicitement, à ces concepts.

La théorie des automates a également connu de grands développements du point de vue mathématique, en liaison étroite avec la théorie des monoïdes finis, princi-

palement sous l'impulsion de M. P. Schützenberger. Elle a aussi des liens profonds avec les théories logiques.

9.1 Mots et langages

Soit A un ensemble. On appelle *mot* sur A toute suite finie

$$u = (a_1, \ldots, a_n)$$

où $n \geq 0$, et $a_i \in A$ pour $i = 1, \ldots, n$. L'entier n est la *longueur* de u, notée $|u|$. Si $n = 0$, le mot est appelé le *mot vide*, et est noté 1 ou ε. Si

$$v = (b_1, \ldots, b_m)$$

est un autre mot, le *produit de concaténation* de u et v est le mot

$$uv = (a_1, \ldots, a_n, b_1, \ldots, b_m)$$

Tout mot étant le produit de concaténation de mots de longueur 1, on identifie les mots de longueur 1 et les éléments de A. On appelle alors A l'*alphabet*, et les éléments de A des *lettres*. On note A^* l'ensemble des mots sur A.

Si u, v et w sont des mots et $w = uv$, alors u est un *préfixe* et v est un *suffixe* de w. Si de plus $u \neq w$ (resp. $v \neq w$), alors u est un *préfixe propre* (resp. un *suffixe propre*) de w. Le mot v est un *facteur* d'un mot w s'il existe des mots u et u' tels que $w = uvu'$.

Un *langage* (formel) sur A est une partie X de A^*. Si X et Y sont des langages, le *produit* XY est défini par

$$XY = \{xy \mid x \in X,\, y \in Y\}$$

Ce produit est associatif, et le langage $\{1\}$ est élément neutre pour le produit. On définit les puissances de X par $X^0 = \{1\}$, et $X^{n+1} = X^n X$ pour $n \geq 0$. L'*étoile* de X est le langage

$$X^* = \bigcup_{n \geq 0} X^n$$

C'est l'ensemble de tous les produits $x_1 \cdots x_n$, pour $n \geq 0$ et $x_1, \ldots, x_n \in X$. Le mot vide appartient toujours à X^*. Il est facile de vérifier que

$$X^* = \{1\} \cup XX^* = \{1\} \cup X^*X$$

Il est commode de noter X^+ l'ensemble $XX^* = X^*X$. Souvent, on omettra des accolades autour de singletons; ainsi, on écrira w au lieu de $\{w\}$.

9.2 Automates finis

9.2.1 Définition

Un *automate fini* sur un alphabet fini A est composé d'un ensemble fini Q d'*états*, d'un ensemble $I \subset Q$ d'états *initiaux* , d'un ensemble $T \subset Q$ d'états *terminaux* ou *finals* et d'un ensemble $\mathcal{F} \subset Q \times A \times Q$ de *flèches*. Un automate est habituellement noté

$$\mathcal{A} = (Q, I, T, \mathcal{F})$$

Parfois, on écrit plus simplement $\mathcal{A} = (Q, I, T)$, lorsque l'ensemble des flèches est sous-entendu, mais cette notation est critiquable car l'ensemble des flèches est essentiel. L'*étiquette* d'une flèche $f = (p, a, q)$ est la lettre a. Un *calcul* de longueur n dans $\mathcal{A}$ est une suite $c = f_1 \cdots f_n$ de flèches consécutives $f_i = (p_i, a_i, q_i)$, c'est-à-dire telles que $q_i = p_{i+1}$ pour $i = 1, \ldots, n-1$. L'*étiquette* du calcul c est $|c| = a_1 \cdots a_n$. On écrit également, si $w = |c|$,

$$c : p_1 \to q_n \quad \text{ou} \quad c : p_1 \xrightarrow{w} q_n$$

Par convention, il existe un calcul vide $1_q : q \to q$ d'étiquette ε ou 1 (le mot vide) pour chaque état q. Les calculs peuvent être composés. Etant donnés deux calculs $c : p \to q$ et $d : q \to r$, le calcul $cd : p \to r$ est défini par concaténation. On a bien entendu $|cd| = |c|\,|d|$.

Un calcul $c : i \to t$ est dit *réussi* si $i \in I$ et $t \in T$. Un mot est *reconnu* s'il est l'étiquette d'un calcul réussi. Le *langage reconnu* par l'automate $\mathcal{A}$ est l'ensemble des mots reconnus par $\mathcal{A}$, soit

$$L(\mathcal{A}) = \{w \in A^* \mid \exists c : i \to t,\ i \in I,\ t \in T,\ w = |c|\}$$

Une partie $X \subset A^*$ est *reconnaissable* s'il existe un automate fini $\mathcal{A}$ sur A telle que $X = L(\mathcal{A})$. La famille de toutes les parties reconnaissables de A^* est notée $\mathrm{Rec}(A^*)$.

La terminologie qui vient d'être introduite suggère d'elle-même une représentation graphique d'un automate fini par ce qui est appelé son *diagramme d'états* : les états sont représentés par les sommets d'un graphe (plus précisément d'un multigraphe). Chaque flèche (p, a, q) est représentée par un arc étiqueté qui relie l'état de départ p à l'état d'arrivée q, et qui est étiqueté par l'étiquette a de la flèche. Un calcul n'est autre qu'un chemin dans le multigraphe, et l'étiquette du calcul est la suite des étiquettes des arcs composant le chemin. Dans les figures, on attribue un signe distinctif aux états initiaux et terminaux. Un état initial est muni d'une flèche qui pointe sur lui, un état final est repéré par une flèche qui le quitte. Parfois, nous convenons de réunir en un seul arc plusieurs arcs étiquetés ayant les mêmes extrémités. L'arc porte alors l'ensemble de ces étiquettes. En particulier, s'il y a une flèche (p, a, q) pour toute lettre $a \in A$, on tracera une flèche unique d'étiquette A.

9.2.2 Exemples

L'automate de la figure 2.1 est défini sur l'alphabet $A = \{a, b\}$.

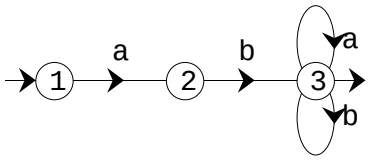

Figure 2.1: *Automate reconnaissant le langage abA^*.*

L'état initial est l'état 1, le seul état final est 3. Tout calcul réussi se factorise en

$$1 \xrightarrow{a} 2 \xrightarrow{b} 3 \xrightarrow{w} 3$$

avec $w \in A^*$. Le langage reconnu est donc bien abA^*. Toujours sur l'alphabet $A = \{a, b\}$, considérons l'automate de la figure 2.2.

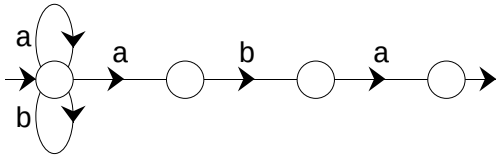

Figure 2.2: *Automate reconnaissant le langage A^*aba.*

Cet automate reconnaît l'ensemble A^*aba des mots qui se terminent par aba. L'automate de la figure 2.3 est défini sur l'alphabet $A = \{a, b\}$.

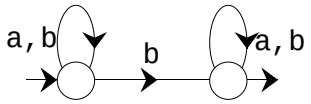

Figure 2.3: *Automate reconnaissant les mots contenant au moins un b.*

Tout calcul réussi contient exactement une fois la flèche $(1, b, 2)$. Un mot est donc reconnu si et seulement s'il contient au moins une fois la lettre b.

L'automate de la figure 2.4 reconnaît l'ensemble des mots contenant un nombre impair de a. Un autre exemple est l'automate vide, ne contenant pas d'états. Il reconnaît le langage vide. A l'inverse, l'automate de la figure 2.5 ayant un seul état qui est à la fois initial et terminal, et une flèche pour chaque lettre $a \in A$ reconnaît tous les mots.

Enfin, le premier des deux automates de la figure 2.6 ne reconnaît que le mot vide, alors que le deuxième reconnaît tous les mots sur A sauf le mot vide, c'est-à-dire le langage A^+.

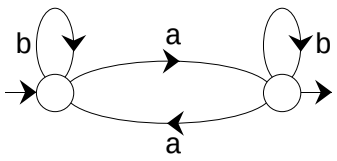

Figure 2.4: *Automate reconnaissant les mots contenant un nombre impair de a.*

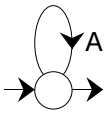

Figure 2.5: *Tous les mots sont reconnus.*

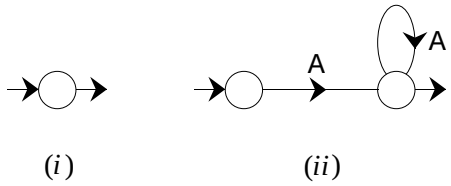

Figure 2.6: *Automates reconnaissant* (i) *le mot vide et* (ii) *tous les mots sauf le mot vide.*

9.2.3 Automates déterministes

Un état $q \in Q$ d'un automate $\mathcal{A} = (Q, I, T, \mathcal{F})$ est *accessible* s'il existe un calcul $c : i \to q$ avec $i \in I$. De même, l'état q est *coaccessible* s'il existe un calcul $c : q \to t$ avec $t \in T$. Un automate est *émondé* si tous ses états sont accessibles et coaccessibles. Soit P l'ensemble des états qui sont à la fois accessibles et coaccessibles, et soit $\mathcal{A}^0 = (P, I \cap P, T \cap P, \mathcal{F} \cap P \times A \times P)$. Il est clair que $\mathcal{A}^0$ est émondé. Comme tout calcul réussi de $\mathcal{A}$ ne passe que par des états accessibles et coaccessibles, on a $L(\mathcal{A}^0) = L(\mathcal{A})$.

Emonder un automate fini revient à calculer l'ensemble des états qui sont descendants d'un état initial et ascendants d'un état final. Cela peut se faire en temps linéaire en fonction du nombre de flèches (d'arcs), par les algorithmes du chapitre 4.

Dans la pratique, les automates déterministes que nous définissons maintenant sont les plus importants, notamment parce qu'ils sont faciles à implémenter.

Un automate $\mathcal{A} = (Q, I, T, \mathcal{F})$ est *déterministe* s'il possède un seul état initial (c'est-à-dire $|I| = 1$) et si

$$(p, a, q), (p, a, q') \in \mathcal{F} \Rightarrow q = q'$$

Ainsi, pour tout $p \in Q$ et tout $a \in A$, il existe au plus un état q dans Q tel que

$(p, a, q) \in \mathcal{F}$. On pose alors, pour $p \in Q$ et $a \in A$,

$$p \cdot a = \begin{cases} q & \text{si } (p, a, q) \in \mathcal{F}, \\ \emptyset & \text{sinon.} \end{cases}$$

On définit ainsi une fonction partielle

$$Q \times A \to Q$$

appelée la *fonction de transition* de l'automate déterministe. On l'étend aux mots en posant, pour $p \in Q$,

$$p \cdot 1 = p$$

et pour $w \in A^*$, et $a \in A$,

$$p \cdot wa = (p \cdot w) \cdot a$$

Cette notation signifie que $p \cdot wa$ est défini si et seulement si $p \cdot w$ et $(p \cdot w) \cdot a$ sont définis, et dans l'affirmative, $p \cdot wa$ prend la valeur indiquée. Avec cette notation on a, avec $I = \{i\}$,

$$L(\mathcal{A}) = \{w \in A^* \mid i \cdot w \in T\}.$$

Un automate est dit *complet* si, pour tout $p \in Q$ et $a \in A$, il existe au moins un état $q \in Q$ tel que $(p, a, q) \in \mathcal{F}$. Si un automate fini $\mathcal{A}$ n'est pas complet, on peut le compléter sans changer le langage reconnu en ajoutant un nouvel état non final s, et en ajoutant les flèches (p, a, s) pour tout couple (p, a) tel que $(p, a, q) \notin \mathcal{F}$ pour tout $q \in Q$. Rendre un automate déterministe, c'est-à-dire le *déterminiser*, est plus difficile, et donne lieu à un algorithme intéressant.

Théorème 2.1. *Pour tout automate fini $\mathcal{A}$, il existe un automate fini déterministe et complet $\mathcal{B}$ tel que*

$$L(\mathcal{A}) = L(\mathcal{B}).$$

Preuve. Soit $\mathcal{A} = (Q, I, T, \mathcal{F})$. On définit un automate déterministe $\mathcal{B}$ qui a pour ensemble d'états l'ensemble $\wp(Q)$ des parties de Q, pour état initial I, et pour ensemble d'états terminaux $V = \{S \subset Q \mid S \cap T \neq \emptyset\}$. On définit enfin la fonction de transition de $\mathcal{B}$ pour $S \in \wp(Q)$ et $a \in A$ par

$$S \cdot a = \{q \in Q \mid \exists s \in S : (s, a, q) \in \mathcal{F}\}.$$

Nous prouvons par récurrence sur la longueur d'un mot w que

$$S \cdot w = \{q \in Q \mid \exists s \in S : s \xrightarrow{w} q\}.$$

Ceci est clair si $w = 1$, et est vrai par définition si w est une lettre. Posons $w = va$, avec $v \in A^*$ et $a \in A$. Alors comme par définition $S \cdot w = (S \cdot v) \cdot a$, on a $q \in S \cdot w$ si et seulement s'il existe une flèche (p, a, q), avec $p \in S \cdot v$, donc telle qu'il existe un calcul $s \xrightarrow{v} p$ pour un $s \in S$. Ainsi $q \in S \cdot w$ si et seulement s'il existe un calcul $s \xrightarrow{w} q$ avec $s \in S$. Ceci prouve l'assertion.

Maintenant, $w \in L(\mathcal{A})$ si et seulement s'il existe un calcul réussi d'étiquette w, ce qui signifie donc que $I \cdot w$ contient au moins un état final de $\mathcal{A}$, en d'autres termes que $I \cdot w \cap T \neq \emptyset$. Ceci prouve l'égalité $L(\mathcal{A}) = L(\mathcal{B})$. ∎

La démonstration du théorème est constructive. Elle montre que pour un automate $\mathcal{A}$ à n états, on peut construire un automate fini déterministe reconnaissant le même langage et ayant 2^n états. La construction est appelée la *construction par sous-ensembles* (en anglais la «subset construction»).

Exemple. Sur l'alphabet $A = \{a, b\}$, considérons l'automate $\mathcal{A}$ reconnaissant le langage A^*ab des mots se terminant par ab (figure 2.7).

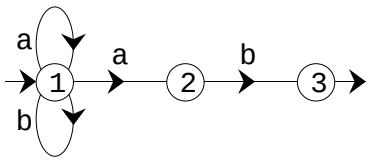

Figure 2.7: *Un automate reconnaissant le langage A^*ab.*

L'application stricte de la preuve du théorème conduit à l'automate déterministe à 8 états de la figure 2.8. Cet automate a beaucoup d'états inaccessibles. Il suffit

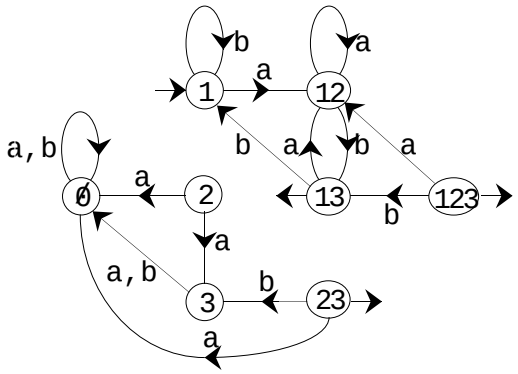

Figure 2.8: *L'automate déterminisé.*

de se contenter des états accessibles, et si l'on n'a pas besoin d'un automate complet, il suffit de considérer la partie émondée; dans notre exemple, on obtient l'automate complet à trois états de la figure 2.9.

En pratique, pour déterminiser un automate $\mathcal{A}$, on ne construit pas l'automate déterministe de la preuve en entier avant de l'émonder; on combine plutôt les deux étapes en une seule, en faisant une recherche des descendants de l'état initial de l'automate $\mathcal{B}$ pendant sa construction. On maintient pour cela un ensemble R d'états de $\mathcal{B}$ déjà construits, et un ensemble de *transitions* (S, a) qui restent à explorer et qui sont composées d'un état de $\mathcal{B}$ déjà construit et d'une lettre. A chaque étape, on choisit un couple (S, a) et on construit l'état $S \cdot a$ de $\mathcal{B}$ en se

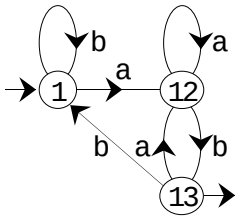

Figure 2.9: *L'automate déterministe et émondé.*

servant de l'automate de départ $\mathcal{A}$. Si l'état $S' = S \cdot a$ est connu parce qu'il figure dans R, on passe à la transition suivante; sinon, on l'ajoute à R et on ajoute à la liste des transitions à explorer les couples (S', a), pour $a \in A$.

Exemple. Reprenons l'exemple ci-dessus. Au départ, l'ensemble d'états de l'automate déterministe est réduit à $\{1\}$, et la liste E des transitions à explorer est $(\{1\}, a)$, $(\{1\}, b)$. On débute donc avec

$$R = \{\{1\}\} \qquad E = (\{1\}, a), (\{1\}, b)$$

Choisissons la première transition, ce qui donne le nouvel état $\{1, 2\}$ et les nouvelles transitions $(\{1, 2\}, a), (\{1, 2\}, b)$, d'où

$$R = \{\{1\}, \{1, 2\}\} \qquad E = (\{1\}, b), (\{1, 2\}, a), (\{1, 2\}, b)$$

Les étapes suivantes sont (noter que l'on a le choix de la transition à traiter, ici on les considère dans l'ordre d'arrivée) :

$$\begin{array}{ll} R = \{\{1\}, \{1, 2\}\} & E = (\{1, 2\}, a), (\{1, 2\}, b) \\ R = \{\{1\}, \{1, 2\}\} & E = (\{1, 2\}, b) \\ R = \{\{1\}, \{1, 2\}, \{1, 3\}\} & E = (\{1, 3\}, a), (\{1, 3\}, b) \\ R = \{\{1\}, \{1, 2\}, \{1, 3\}\} & E = (\{1, 3\}, b) \\ R = \{\{1\}, \{1, 2\}, \{1, 3\}\} & E = \emptyset \end{array}$$

Comme le suggère la construction, il existe des automates à n états pour lesquels tout automate déterministe équivalent possède de l'ordre de 2^n états. Voici, pour tout entier positif n fixé, un automate à $n + 1$ états (figure 2.10) qui reconnaît le langage $L_n = A^* a A^{n-1}$ sur l'alphabet $A = \{a, b\}$. Nous allons démontrer

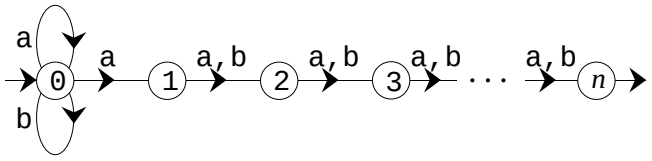

Figure 2.10: *Un automate à $n + 1$ états.*

que tout automate déterministe reconnaissant L_n a au moins 2^n états. Soit en effet $\mathcal{A} = (Q, i, T)$ un automate déterministe reconnaissant L_n. Alors on a, pour $v, v' \in A^n$:

$$i \cdot v = i \cdot v' \Rightarrow v = v' \tag{2.1}$$

En effet, supposons $v \neq v'$. Il existe alors des mots x, x', w tels que $v = xaw$ et $v' = x'bw$ ou vice-versa. Soit y un mot quelconque tel que wy est de longueur $n-1$. Alors $vy = xawy \in L_n$ et $v'y = x'bwy \notin L_n$, alors que l'égalité $i \cdot v = i \cdot v'$ implique que $i \cdot vy = i \cdot v'y$, et donc que vy et vy' soit sont tous deux dans L_n, soit sont tous deux dans le complément de L_n. D'où la contradiction. L'implication (2.1) montre que $\mathcal{A}$ a au moins 2^n états.

9.3 Opérations

9.3.1 Opérations booléennes

Proposition 3.1. *La famille des langages reconnaissables sur un alphabet A est fermée pour les opérations booléennes, c'est-à-dire pour l'union, l'intersection et la complémentation.*

Preuve. Soient X et X' deux langages reconnaissables de A^*, et soient $\mathcal{A} = (Q, i, T)$ et $\mathcal{A}' = (Q', i', T')$ deux automates finis déterministes complets tels que $X = L(\mathcal{A})$ et $X' = L(\mathcal{A}')$. Considérons l'automate déterministe complet

$$\mathcal{B} = (Q \times Q', (i, i'), S)$$

dont la fonction de transition est définie par

$$(p, p') \cdot a = (p \cdot a, p' \cdot a)$$

pour tout couple d'états (p, p') et toute lettre $a \in A$. Alors on a

$$(p, p') \cdot w = (p \cdot w, p' \cdot w)$$

pour tout mot w, comme on le vérifie immédiatement en raisonnant par récurrence sur la longueur de w. Il résulte de cette équation que pour $S = T \times T'$, on obtient $L(\mathcal{B}) = X \cap X'$, et pour $S = (T \times Q') \cup (Q \times T')$, on obtient $L(\mathcal{B}) = X \cup X'$. Enfin, pour $S = T \times (Q' - T')$, on a $L(\mathcal{B}) = X - X'$. ∎

Corollaire 3.2. *Un langage X est reconnaissable si et seulement si $X - \{1\}$ est reconnaissable.* ∎

9.3.2 Automates asynchrones

Nous introduisons maintenant une généralisation des automates finis, dont nous montrons qu'en fait ils reconnaissent les mêmes langages. L'extension réside dans le fait d'autoriser également le mot vide comme étiquette d'une flèche. L'avantage de cette convention est une bien plus grande souplesse dans la construction des

automates. Elle a en revanche l'inconvénient d'accroître le nondéterminisme des automates.

Un *automate fini asynchrone* $\mathcal{A} = (Q, I, T, \mathcal{F})$ est un automate dans lequel certaines flèches peuvent être étiquetées par le mot vide, donc tel que

$$\mathcal{F} \subset Q \times (A \cup 1) \times Q.$$

Les notions de calcul, d'étiquette, de mot et de langage reconnu s'étendent de manière évidente aux automates asynchrones. La terminologie provient de l'observation que, dans un automate asynchrone, la longueur d'un calcul réussi, donc la longueur d'un calcul, peut être supérieure à la longueur du mot qui est son étiquette. Dans un automate usuel en revanche, lecture d'une lettre et progression dans l'automate sont rigoureusement synchronisées.

Exemple. L'automate asynchrone de la figure 3.1 reconnaît le langage a^*b^*.

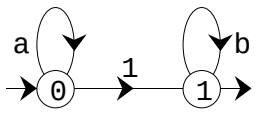

Figure 3.1: *Un automate asynchrone.*

Proposition 3.3. *Pour tout automate fini asynchrone $\mathcal{A}$, le langage $L(\mathcal{A})$ est reconnaissable.*

Preuve. Soit $\mathcal{A} = (Q, I, T)$ un automate fini asynchrone sur A. Soit $\mathcal{B} = (Q, I, T)$ l'automate fini dont les flèches sont les triplets (p, a, q) pour lesquels il existe un calcul $c : p \xrightarrow{a} q$ dans $\mathcal{A}$. On a

$$L(\mathcal{A}) \cap A^+ = L(\mathcal{B}) \cap A^+.$$

Si $I \cap T \neq \emptyset$, les deux langages $L(\mathcal{A})$ et $L(\mathcal{B})$ contiennent le mot vide et sont donc égaux. Dans le cas contraire, on a $L(\mathcal{A}) = L(\mathcal{B}) \cup 1$ ou $L(\mathcal{A}) = L(\mathcal{B})$, selon que le mot vide 1 est l'étiquette d'un calcul réussi dans $\mathcal{A}$ ou non. Ceci prouve que $L(\mathcal{A})$ est reconnaissable. ∎

Reprenons notre exemple de l'automate de la figure 3.1. La construction de la preuve conduit à l'automate «synchrone» de la figure 3.2.

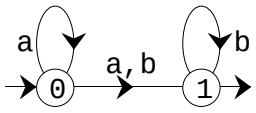

Figure 3.2: *L'automate précédent «synchronisé».*

En effet, il existe dans l'automate d'origine deux calculs, de longueur 2, de l'état 0 à l'état 1, l'un portant l'étiquette a, l'autre l'étiquette b. Notons que cet automate ne reconnaît pas le mot vide, donc ne reconnaît le langage a^*b^* qu'au mot vide près.

9.3.3 Produit et étoile

Proposition 3.4. *Si X est reconnaissable, alors X^* est reconnaissable; si X et Y sont reconnaissables, alors XY est reconnaissable.*

Preuve. Soit X un langage reconnaissable. Comme $X^* = (X-1)^*$, et que $X-1$ est reconnaissable, on peut supposer que X ne contient pas le mot vide. Soit $\mathcal{A} = (Q, I, T, \mathcal{F})$ un automate fini reconnaissant X, et soit $\mathcal{B} = (Q, I, T)$ l'automate asynchrone ayant pour flèches

$$\mathcal{F} \cup (T \times \{1\} \times I)$$

Notons que $T \cap I = \emptyset$ parce que le mot vide n'est pas dans X.

Montrons que l'on a $X^+ = L(\mathcal{B})$. Il est clair en effet que $X^+ \subset L(\mathcal{B})$. Réciproquement, soit $c : i \xrightarrow{w} t$ un calcul réussi dans $\mathcal{B}$. Ce calcul a la forme

$$c : i_1 \xrightarrow{w_1} t_1 \xrightarrow{1} i_2 \xrightarrow{w_2} t_2 \xrightarrow{1} \cdots i_n \xrightarrow{w_n} t_n$$

avec $i = i_1$, $t = t_n$, et où aucun des calculs $c_k : i_k \xrightarrow{w_k} t_k$ ne contient de flèche étiquetée par le mot vide. Alors $w_1, w_2, \ldots, w_n \in X$, et donc $w \in X^+$. Il en résulte évidemment que $X^* = X^+ \cup 1$ est reconnaissable.

Considérons maintenant deux automates finis $\mathcal{A} = (Q, I, T, \mathcal{F})$ et $\mathcal{B} = (P, J, R, \mathcal{G})$ reconnaissant respectivement les langages X et Y. On peut supposer les ensembles d'états Q et P disjoints. Soit alors $\mathcal{C} = (Q \cup P, I, R)$ l'automate dont les flèches sont

$$\mathcal{F} \cup \mathcal{G} \cup (T \times \{1\} \times J)$$

Alors on vérifie facilement que $L(\mathcal{C}) = XY$. ∎

9.4 Langages rationnels

Dans cette section, nous définissons une autre famille de langages, les langages rationnels, et nous prouvons qu'ils coïncident avec les langages reconnaissables. Grâce à ce résultat, dû à Kleene, on dispose de deux caractérisations très différentes d'une même famille de langages.

9.4.1 Langages rationnels : définitions

Soit A un alphabet. Les *opérations rationnelles* sur les parties de A^* sont les opérations suivantes :

union	$X \cup Y$;
produit	$XY = \{xy \mid x \in X,\ y \in Y\}$;
étoile	$X^* =$ le sous-monoïde engendré par X.

Une famille de parties de A^* est *rationnellement fermée* si elle est fermée pour les trois opérations rationnelles. Les *langages rationnels*indexrationnel, langage de A^* sont les éléments de la plus petite famille rationnellement fermée de A^* qui contient les singletons (c'est-à-dire les langages réduits à un seul mot) et le langage vide. Cette famille est notée $\mathrm{Rat}(A^*)$. Une expression d'un langage comme combinaison finie d'unions, de produits et d'étoiles de singletons est une *expression rationnelle*. Une étude plus formelle des expressions rationnelles est entreprise au paragraphe 3 de cette section. Considérons quelques exemples.

Le langage abA^*, avec $A = \{a, b\}$, est rationnel : il est le produit des singletons a et b par l'étoile de A qui est lui-même l'union des deux lettres a et b. De même, le langage A^*aba est rationnel. Toujours sur $A = \{a, b\}$, le langage L des mots qui contiennent un nombre pair de a est rationnel : c'est le langage $L = (ab^*a \cup b)^*$.

9.4.2 Le théorème de Kleene

Le théorème suivant est dû à Kleene :

Théorème 4.1 (de Kleene)**.** *Soit A un alphabet fini. Alors les langages rationnels et reconnaissables sur A coïncident :* $\mathrm{Rat}(A^*) = \mathrm{Rec}(A^*)$.

Cet énoncé est remarquable dans la mesure où il donne deux caractérisations très différentes d'une même famille de langages : l'automate fini est un moyen de calcul, et se donner un langage par un automate fini revient à se donner un algorithme pour vérifier l'appartenance de mots au langage. Au contraire, une expression rationnelle décrit la structure syntaxique du langage. Elle permet en particulier des manipulations, et des opérations entre langages.

La démonstration du théorème de Kleene est en deux parties. Une première partie consiste à montrer que tout langage rationnel est reconnaissable, c'est-à-dire à prouver l'inclusion $\mathrm{Rat}(A^*) \subset \mathrm{Rec}(A^*)$. On l'obtient en montrant que la famille $\mathrm{Rec}(A^*)$ est fermée pour les opérations rationnelles. L'inclusion opposée $\mathrm{Rec}(A^*) \subset \mathrm{Rat}(A^*)$ se montre en exhibant, pour tout automate fini, une expression rationnelle (au sens du paragraphe suivant) du langage qu'il reconnaît. Nous commençons par la proposition suivante :

Proposition 4.2. *Soit A un alphabet. Tout langage rationnel de A^* est reconnaissable :* $\mathrm{Rat}(A^*) \subset \mathrm{Rec}(A^*)$.

Preuve. La famille des langages sur A reconnus par des automates finis est fermée par union, produit et étoile. Par ailleurs, elle contient les singletons ; en effet, pour tout mot $w = a_1 a_2 \cdots a_n$ de longueur n, l'automate de la figure 4.1 reconnaît ce mot. Ainsi, $\mathrm{Rec}(A^*)$ est fermée pour les opérations rationnelles. Ceci montre que tout langage rationnel sur A est reconnaissable. ∎

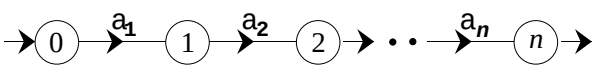

Figure 4.1: *Automate reconnaissant exactement le mot* $w = a_1 a_2 \cdots a_n$.

Proposition 4.3. *Soit A un alphabet fini. Tout langage reconnaissable de A^* est rationnel :* $\mathrm{Rec}(A^*) \subset \mathrm{Rat}(A^*)$.

Preuve. Soit $\mathcal{A} = (Q, I, T)$ un automate fini sur A, et soit X le langage reconnu. Numérotons les états de manière que $Q = \{1, \ldots, n\}$. Pour i, j dans Q, soit $X_{i,j} = \{w \mid i \xrightarrow{w} j\}$, et pour $k = 0, \ldots, n$, soit $X_{i,j}^{(k)}$ l'ensemble des étiquettes des calculs de longueur strictement positive de la forme

$$i \to p_1 \to \ldots \to p_s \to j, \qquad s \geq 0, \quad p_1, \ldots, p_s \leq k$$

Observons que $X_{i,j}^{(0)} \subset A$, et donc que chaque $X_{i,j}^{(0)}$ est une partie rationnelle, parce que l'alphabet est fini. Ensuite, on a

$$X_{i,j}^{(k+1)} = X_{i,j}^{(k)} \cup X_{i,k+1}^{(k)} \left(X_{k+1,k+1}^{(k)}\right)^* X_{k+1,j}^{(k)} \tag{4.1}$$

Cette formule montre, par récurrence sur k, que chacun des langages $X_{i,j}^{(k)}$ est rationnel. Or

$$X_{i,j} = \begin{cases} X_{i,j}^{(n)} & \text{si } i \neq j \\ 1 \cup X_{i,j}^{(n)} & \text{si } i = j \end{cases} \tag{4.2}$$

et

$$X = \bigcup_{\substack{i \in I \\ t \in T}} X_{i,t} \tag{4.3}$$

et par conséquent les langages $X_{i,j}$ sont rationnels. Donc X est également rationnel. ∎

Les formules (4.1)–(4.3) permettent de calculer effectivement une expression (expression rationnelle) pour le langage reconnu par un automate fini. Cette méthode est appelée l'algorithme de MacNaughton et Yamada, d'après ses créateurs. Il est très simple, mais pas très efficace dans la mesure où il faut calculer les $O(n^3)$ expressions $X_{i,j}^{(k)}$ pour un automate à n états. Une façon parfois plus commode — surtout pour les calculs à la main — de déterminer l'expression rationnelle du langage reconnu par un automate consiste à résoudre un système d'équations linéaires naturellement associé à tout automate fini.

Soit $\mathcal{A} = (Q, I, T, \mathcal{F})$ un automate fini sur A. Le *système d'équations* (linéaire droit) associé à $\mathcal{A}$ est

$$X_p = \bigcup_{q \in Q} E_{p,q} X_q \cup \delta_{p,T} \qquad p \in Q$$

où

$$E_{p,q} = \{a \in A \mid (p,a,q) \in \mathcal{F}\}, \qquad p,q \in Q$$
$$\delta_{p,T} = \begin{cases} \{1\} & \text{si } p \in T \\ \emptyset & \text{sinon} \end{cases}$$

Considérons par exemple l'automate de la figure 4.2.

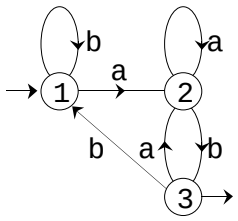

Figure 4.2: *Quel est le langage reconnu par cet automate?*

Le système d'équations associé à cet automate est

$$\begin{aligned} X_1 &= bX_1 \cup aX_2 \\ X_2 &= \qquad aX_2 \cup bX_3 \\ X_3 &= bX_1 \cup aX_2 \qquad \cup 1 \end{aligned}$$

Une *solution* du système d'équations est une famille $(L_p)_{p\in Q}$ de langages qui vérifie le système d'équations.

Proposition 4.4. *Soit $\mathcal{A} = (Q, I, T)$ un automate fini sur A, et posons*

$$L_p(\mathcal{A}) = \{w \in A^* \mid \exists c : p \to t \in T,\ |c| = w\}$$

Alors la famille $(L_p(\mathcal{A}))_{p\in Q}$ est l'unique solution du système d'équations associé à $\mathcal{A}$.

Le langage $L_p(\mathcal{A})$ est l'ensemble des mots reconnus par $\mathcal{A}$ en prenant p pour état initial, de sorte que

$$L(\mathcal{A}) = \bigcup_{p\in I} L_p(\mathcal{A})$$

Nous commençons par le cas particulier d'un système d'équations réduit à une seule équation. La démarche sera la même dans le cas général.

Lemme 4.5 (Lemme d'Arden)**.** *Soient E et F deux langages. Si $1 \notin E$, alors l'équation $X = EX \cup F$ possède une solution unique, à savoir le langage E^*F.*

Preuve. Posons $L = E^*F$. Alors $EL \cup F = EE^*F \cup F = (EE^* \cup 1)F = E^*F = L$, ce qui montre que L est bien solution de l'équation. Supposons qu'il y ait deux solutions L et L'. Comme

$$L - L' = (EL \cup F) - L' = EL - L' \subset EL - EL'$$

et que $EL - EL' \subset E(L - L')$, on a $L - L' \subset E(L - L') \subset E^n(L - L')$ pour tout $n > 0$. Mais comme le mot vide n'appartient pas à E, cela signifie qu'un mot de $L - L'$ a pour longueur au moins n pour tout n. Donc $L - L'$ est vide, et par conséquent $L \subset L'$ et de même $L' \subset L$. ∎

Preuve de la proposition. Pour montrer que les langages $L_p(\mathcal{A})$, $(p \in Q)$, constituent une solution, posons

$$L'_p = \bigcup_{q \in Q} E_{p,q} L_q(\mathcal{A}) \cup \delta_{p,T}$$

et vérifions que $L_p(\mathcal{A}) = L'_p$ pour $p \in Q$. Si $1 \in L_p(\mathcal{A})$, alors $p \in T$, donc $\delta_{p,T} = \{1\}$ et $1 \in L'_p$, et réciproquement. Soit $w \in L_p(\mathcal{A})$ de longueur > 0; il existe un calcul $c : p \to t$ pour un $t \in T$ d'étiquette w. En posant $w = av$, avec a une lettre et v un mot, le calcul c se factorise en $p \xrightarrow{a} q \xrightarrow{v} t$ pour un état q. Mais alors $a \in E_{p,q}$ et $v \in L_q(\mathcal{A})$, donc $w \in L'_p$. L'inclusion réciproque se montre de la même façon.

Pour prouver l'unicité de la solution, on procède comme dans la preuve du lemme d'Arden. Soient $(L_p)_{p \in Q}$ et $(L'_p)_{p \in Q}$ deux solutions du système d'équations, et posons $M_p = L_p - L'_p$ pour $p \in Q$. Alors

$$M_p = \bigcup_{q \in Q} E_{p,q} L_q \cup \delta_{p,T} - L'_p \subset \bigcup_{q \in Q} (E_{p,q} L_q - E_{p,q} L'_q) \subset \bigcup_{q \in Q} E_{p,q} M_q$$

A nouveau, ces inclusions impliquent que $M_p = \emptyset$ pour tout p. Supposons en effet le contraire, soit w un mot de longueur minimale dans

$$M' = \bigcup_{p \in Q} M_p$$

et soit p un état tel que $w \in M_p$. Alors

$$w \in \bigcup_{q \in Q} E_{p,q} M_q$$

donc w est le produit d'une lettre et d'un mot v de M'; mais alors v est plus court que w, une contradiction. ∎

Pour *résoudre* un système d'équations, on peut procéder par élimination de variables (c'est la méthode de Gauss). Dans notre exemple, on substitue la troisième équation dans la deuxième, ce qui donne

$$X_2 = (a \cup ba) X_2 \cup b^2 X_1 \cup b$$

qui, par le lemme d'Arden, équivaut à l'équation

$$X_2 = (a \cup ba)^* (b^2 X_1 \cup b)$$

Cette expression pour X_2 est substituée dans la première équation du système; on obtient

$$X_1 = (b \cup a(a \cup ba)^* b^2) X_1 \cup a(a \cup ba)^* b$$

d'où, à nouveau par le lemme d'Arden,

$$X_1 = (b \cup a(a \cup ba)^* b^2)^* a(a \cup ba)^* b$$

Il n'est pas du tout évident que cette dernière expression soit égale à $\{a, b\}^* ab$. Pour le montrer, observons d'abord que

$$(a \cup ba)^* = a^*(ba^+)^*$$

(rappelons que $X^+ = XX^* = X^*X$ pour tout X) en utilisant la règle $(U \cup V)^* = U^*(VU^*)^*$, d'où

$$a(a \cup ba)^* b = a^+(ba^+)^* b = (a^+ b)^+$$

Il en résulte que

$$b \cup a(a \cup ba)^* b^2 = [1 \cup (a^+ b)^+]b = (a^+ b)^* b$$

d'où

$$X_1 = [(a^+ b)^* b]^* (a^+ b)^* (a^+ b) = (a^+ b \cup b)^* (a^+ b) = (a^* b)^* a^* ab = \{a, b\}^* ab$$

9.4.3 Expressions rationnelles

Le théorème de Kleene, qui établit l'égalité entre langages rationnels et reconnaissables, possède des preuves constructives. L'une est l'algorithme de McNaughton et Yamada, la deuxième revient à résoudre un système d'équations linéaires. Dans les deux cas, la rationalité du langage obtenu est évidente parce qu'elle provient d'une *expression rationnelle* fournie pour le langage; plus précisément, le langage est exprimé à l'aide des opérations rationnelles.

Dans ce paragraphe, nous étudions les expressions rationnelles en elles-mêmes, en faisant une distinction entre une expression et le langage qu'elle décrit, un peu comme l'on fait la différence entre une expression arithmétique et la valeur numérique qu'elle représente. L'approche est donc purement syntaxique, et le langage représenté par une expression peut être considéré comme résultant d'une évaluation de l'expression.

Les manipulations (algébriques ou combinatoires) d'expressions permettent de construire des automates efficaces reconnaissant le langage représenté par des opérations qui sont proches de l'analyse syntaxique, et qui ne font pas intervenir le langage lui-même. Elles se prêtent donc bien à une implémentation effective. Une autre application est la recherche de motifs dans un texte, comme elle est réalisée dans un traitement de texte par exemple, et comme elle sera traitée au chapitre suivant.

Soit A un alphabet fini, soient 0 et 1 deux symboles ne figurant pas dans A, et soient $+, \cdot, ^*$ trois symboles de fonctions, les deux premiers binaires, le dernier unaire. Les *expressions* sur $A \cup \{0, 1, +, \cdot, ^*\}$ sont les termes bien formés sur cet ensemble de symboles.

Exemple. Sur $A = \{a, b\}$ le terme $a(a + a \cdot b)^* b$ est une expression. Comme souvent, on peut représenter une expression par un arbre qui, sur cet exemple, est l'arbre de la figure 4.3.

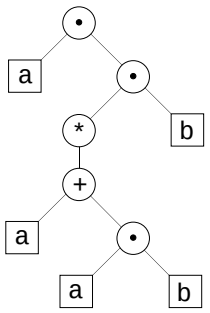

Figure 4.3: *L'expression rationnelle* $a(a + a \cdot b)^* b$.

Les parenthèses ne sont utilisées que lorsqu'elles sont nécessaires, en convenant que l'opération * a priorité sur $\cdot$ qui a priorité sur $+$; on omet également le signe de multiplication quand cela ne prête pas à confusion.

Les *expressions rationnelles* sur A sont les éléments de l'algèbre quotient obtenue en convenant que

(i) l'opération $+$ est idempotente, associative et commutative,

(ii) l'opération $\cdot$ est associative et distributive par rapport à l'opération $+$,

(iii) les équations suivantes sont vérifiées pour tout terme e :

$$\begin{aligned} 0 + e &= e = e + 0 \\ 1 \cdot e &= e = e \cdot 1 \\ 0 \cdot e &= 0 = e \cdot 0 \end{aligned}$$

(iv) on a :

$$0^* = 1^* = 1$$

Exemple. Par exemple, $(0 + a(1 + b \cdot 1))^*$ et $(ab + a)^*$ sont la *même* expression rationnelle, alors que $1 + a^* a$ et a^* sont deux expressions différentes (même si elles décrivent le même langage).

On note $\mathcal{E}(A)$ l'algèbre des expressions rationnelles ainsi obtenue. La convention d'identifier certaines expressions selon les règles que nous venons d'édicter allège très considérablement l'écriture et l'exposé; tout ce qui suit pourrait se faire également en ne raisonnant que sur les termes ou les arbres.

L'important est toutefois que l'*égalité* de deux expressions est *facilement décidable* sur les expressions elles-mêmes. Pour ce faire, on met une expression, donnée disons sous forme d'arbre, en «forme normale» en supprimant d'abord les feuilles 0 ou 1 quand c'est possible en appliquant les égalités (iii) ci-dessus, puis en distribuant le produit par rapport à l'addition pour faire «descendre» au maximum les produits. Lorsque les deux expressions sont en forme normale, on peut décider récursivement si elles sont égales : il faut et il suffit pour cela qu'elles aient même symbole aux racines, et si les *suites* d'expressions des fils obtenues en faisant jouer l'associativité de $\cdot$ et les *ensembles* d'expressions des fils modulo l'associativité, l'idempotence et la commutativité de $+$ sont égaux.

Exemple. En appliquant les règles de simplification à l'expression $(0 + a(1 + b \cdot 1))^*$, on obtient $(a(1+b))^*$. Par distributivité et simplification, elle donne $(a+ab)^*$, et les deux expressions fils de la racine sont les mêmes dans cette expression, et dans $(ab + a)^*$.

Remarque. Dans ce qui précède ne figure pas l'opération $e \mapsto e^+$. C'est pour ne pas alourdir l'exposé que nous considérons cette opération comme une abréviation de ee^* ; on pourrait aussi bien considérer cette opération comme opération de base.

Il y a une application naturelle de l'algèbre des expressions rationnelles sur A dans les langages rationnels sur A. C'est l'application

$$L : \mathcal{E}(A) \to \wp(A^*)$$

définie par récurrence comme suit :

$$\begin{array}{l} L(0) = \emptyset, \quad L(1) = \{1\}, \quad L(a) = \{a\}, \\ L(e + e') = L(e) \cup L(e'), \quad L(e \cdot e') = L(e)L(e'), \\ L(e^*) = L(e)^* \end{array}$$

Pour une expression e, le langage $L(e)$ est le langage *décrit* ou *dénoté* par e. Deux expressions e et e' sont dites *équivalentes* lorsqu'elles dénotent le même langage, c'est-à-dire lorsque $L(e) = L(e')$. On écrit alors $e \approx e'$.

Proposition 4.6. *Pour toutes expressions rationnelles e et f, on a les formules suivantes :*

$$\begin{aligned} (ef)^* &\approx 1 + e(fe)^* f \\ (e + f)^* &\approx e^*(fe^*)^* \\ e^* &\approx (1 + e + \ldots + e^{p-1})(e^p)^* \qquad p > 1 \end{aligned}$$

Cette proposition est un corollaire immédiat du lemme suivant :

Lemme 4.7. *Quelles que soient les parties X et Y de A^*, on a*

$$(XY)^* = 1 \cup X(YX)^*Y \tag{4.4}$$
$$(X \cup Y)^* = X^*(YX^*)^* \tag{4.5}$$
$$X^* = (1 \cup X \cup \cdots \cup X^{p-1})(X^p)^* \qquad p > 1 \tag{4.6}$$

Preuve. Pour établir (4.4), montrons d'abord que $(XY)^* \subset 1 \cup X(YX)^*Y$. Pour cela, soit $w \in (XY)^*$. Alors $w = x_1y_1 \cdots x_ny_n$ pour $n \geq 0$ et $x_i \in X$, $y_i \in Y$. Si $n = 0$, alors $w = 1$, sinon $w = x_1vy_n$, avec $v = y_1 \cdots x_n \in (YX)^*$. Ceci prouve l'inclusion. L'inclusion réciproque se montre de manière similaire.

Pour prouver (4.5), il suffit d'établir

$$(X \cup Y)^* \subset 1 \cup X^*(YX^*)^*$$

l'inclusion réciproque étant évidente. Soit $w \in (X \cup Y)^*$. Il existe $n \geq 0$ et $z_1, \ldots, z_n \in X \cup Y$ tels que $w = z_1 \cdots z_n$. En groupant les z_i consécutifs qui appartiennent à X, ce produit peut s'écrire

$$w = x_0y_1x_1 \cdots y_mx_m$$

avec $x_0, \ldots, x_m \in X^*$ et $y_0, \ldots, y_m \in Y$. Par conséquent $w \in 1 \cup X^*(YX^*)^*$.

Considérons enfin (4.6). On a

$$(1 \cup X \cup \cdots \cup X^{p-1})(X^p)^* = \bigcup_{k=0}^{p-1} \bigcup_{m \geq 0} X^{k+mp} = X^* \qquad \blacksquare$$

9.5 Automate minimal

Déterminer un automate ayant un nombre minimum d'états pour un langage rationnel donné est très intéressant du point de vue pratique. Il est tout à fait remarquable que tout langage rationnel possède un automate déterministe minimal unique, à une numérotation des états près. Ce résultat n'est plus vrai si l'on considère des automates qui ne sont pas nécessairement déterministes.

Il y a deux façons de définir l'automate déterministe minimal reconnaissant un langage rationnel donné. La première est intrinsèque; elle est définie à partir du langage par une opération appelée le *quotient*. La deuxième est plus opératoire; on part d'un automate déterministe donné, et on le réduit en identifiant des états appelés *inséparables*. Les algorithmes de minimisation utilisent la deuxième définition.

9.5.1 Quotients

Soit A un alphabet. Pour deux mots u et v, on pose

$$u^{-1}v = \{w \in A^* \mid uw = v\}, \qquad uv^{-1} = \{w \in A^* \mid u = wv\}$$

Un tel ensemble, appelé *quotient gauche* respectivement *droit* est, bien entendu, soit vide soit réduit à un seul mot qui, dans le premier cas est un suffixe de v, et dans le deuxième cas un préfixe de u.

La notation est étendue aux parties en posant, pour $X, Y \subset A^*$

$$X^{-1}Y = \bigcup_{x\in X}\bigcup_{y\in Y} x^{-1}y, \qquad XY^{-1} = \bigcup_{x\in X}\bigcup_{y\in Y} xy^{-1}$$

Les quotients sont un outil important pour l'étude des automates finis et des langages rationnels. On utilise principalement les quotients gauches par un mot, c'est-à-dire les ensembles

$$u^{-1}X = \{w \in A^* \mid uw \in X\}$$

Notons qu'en particulier, $1^{-1}X = X$ pour tout ensemble X, et que $(uv)^{-1}X = v^{-1}(u^{-1})X$. Pour toute partie X de A^*, on pose

$$Q(X) = \{u^{-1}X \mid u \in A^*\} \tag{5.1}$$

Exemple. Sur $A = \{a, b\}$, soit X l'ensemble des mots qui contiennent au moins une fois la lettre a. Alors on a :

$$1^{-1}X = X, \quad a^{-1}X = A^*, \quad b^{-1}X = X, \quad a^{-1}A^* = b^{-1}A^* = A^*$$

donc $Q(X) = \{X, A^*\}$.

Proposition 5.1. *Soit $X \subset A^*$ le langage reconnu par un automate fini déterministe, accessible et complet $\mathcal{A} = (Q, i, T)$; pour $q \in Q$, soit $L_q(\mathcal{A}) = \{w \in A^* \mid q \cdot w \in T\}$. Alors*

$$\{L_q(\mathcal{A}) \mid q \in Q\} = Q(X) \tag{5.2}$$

Preuve. Montrons d'abord que $Q(X)$ est contenu dans $\{L_q(\mathcal{A}) \mid q \in Q\}$. Soit $u \in A^*$, et soit $q = i \cdot u$ (cet état existe parce que $\mathcal{A}$ est complet). Alors $u^{-1}X = L_q(\mathcal{A})$, puisque

$$w \in u^{-1}X \iff uw \in X \iff i \cdot uw \in T \iff q \cdot w \in T \iff w \in L_q(\mathcal{A})$$

Pour montrer l'inclusion réciproque, soit $q \in Q$ et soit $u \in A^*$ tel que $q = i \cdot u$ (un tel mot existe parce que l'automate est accessible). Alors $L_q(\mathcal{A}) = u^{-1}X$. Ceci prouve la proposition. ∎

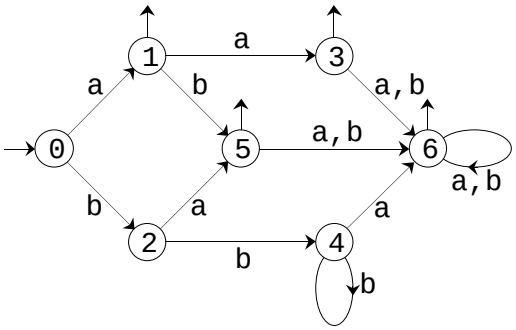

Figure 5.1: *Un automate reconnaissant* $X = b^*a\{a,b\}^*$.

Exemple. Considérons l'automate $\mathcal{A}$ donné dans la figure 5.1. Un calcul rapide montre que

$$L_1(\mathcal{A}) = L_3(\mathcal{A}) = L_5(\mathcal{A}) = L_6(\mathcal{A}) = \{a,b\}^*$$
$$L_0(\mathcal{A}) = L_2(\mathcal{A}) = L_4(\mathcal{A}) = b^*a\{a,b\}^*$$

L'équation (5.2) est bien vérifiée.

On déduit de cette proposition que, pour un langage reconnaissable X, l'ensemble des quotients gauches $Q(X)$ est fini; nous allons voir dans un instant que la réciproque est vraie également. L'équation (5.2) montre par ailleurs que tout automate déterministe, accessible et complet reconnaissant X possède au moins $|Q(X)|$ états. Nous allons voir que ce minimum est atteint, et même, au paragraphe suivant, qu'il existe un automate unique, à un isomorphisme près, qui a $|Q(X)|$ états et qui reconnaît X.

Soit $X \subset A^*$. On appelle *automate minimal* de X l'automate déterministe

$$\mathcal{A}(X) = (Q(X), X, T(X))$$

dont l'ensemble d'états est donné par (5.1), ayant X comme état initial, l'ensemble

$$T(X) = \{u^{-1}X \mid u \in X\} = \{u^{-1}X \mid 1 \in u^{-1}X\}$$

comme ensemble d'états terminaux, la fonction de transition étant définie, pour $Y \in Q$ et $a \in A$ par

$$Y \cdot a = a^{-1}Y$$

Notons que si $Y = u^{-1}X$, alors $Y \cdot a = a^{-1}(u^{-1}X) = (ua)^{-1}X$, donc la fonction de transition est bien définie, et l'automate est complet.

Exemple. L'automate $\mathcal{A}(X)$ pour le langage $X = b^*a\{a,b\}^*$ a les deux états X et $\{a,b\}^*$, le premier est initial, le deuxième est final. L'automate est donné dans la figure 5.2.

Proposition 5.2. *Le langage reconnu par l'automate* $\mathcal{A}(X)$ *est* X.

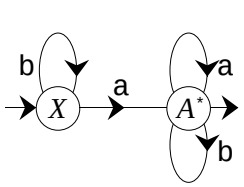

Figure 5.2: *Automate minimal pour* $X = b^*a\{a,b\}^*$

Preuve. Montrons d'abord que, pour tout $w \in A^*$ et pour tout $Y \in Q(X)$, on a $Y \cdot w = w^{-1}Y$. En effet, ceci est vrai si w est une lettre ou le mot vide. Si $w = ua$, avec a une lettre, alors $Y \cdot ua = (Y \cdot u) \cdot a = (u^{-1}Y) \cdot a = a^{-1}(u^{-1}Y) = (ua)^{-1}Y$. Ceci prouve la formule. Il en résulte que

$$w \in L(\mathcal{A}(X)) \iff X \cdot w \in T \iff w^{-1}X \in T \iff w \in X$$

Donc $\mathcal{A}(X)$ reconnaît X. ∎

Corollaire 5.3. *Une partie $X \subset A^*$ est reconnaissable si et seulement si l'ensemble $Q(X)$ est fini.*

Preuve. Si X est reconnaissable, alors (5.2) montre que $Q(X)$ est fini. La réciproque découle de la proposition précédente. ∎

On peut calculer l'ensemble $Q(X)$ pour un langage rationnel, à partir d'une expression rationnelle pour X, à l'aide des formules de la proposition suivante. Cela ne résout pas complètement le problème du calcul de l'automate minimal, parce qu'une difficulté majeure demeure, à savoir tester si deux expressions sont équivalentes. On en parlera plus loin.

Proposition 5.4. *Soit a une lettre, et soient X et Y des langages. Alors on a*

$$\begin{aligned} a^{-1}\emptyset &= a^{-1}1 = \emptyset \\ a^{-1}a &= 1 \\ a^{-1}b &= \emptyset \quad (b \neq a) \\ a^{-1}(X \cup Y) &= a^{-1}X \cup a^{-1}Y \\ a^{-1}(XY) &= (a^{-1}X)Y \cup (X \cap 1)a^{-1}Y \qquad (5.3) \\ a^{-1}X^* &= (a^{-1}X)X^* \qquad (5.4) \end{aligned}$$

Preuve. Seules les deux dernières formules demandent une vérification. Si $w \in a^{-1}(XY)$, alors $aw \in XY$; il existe donc $x \in X$, $y \in Y$ tels que $aw = xy$. Si $x = 1$, alors $aw = y$, donc $w \in (X \cap 1)a^{-1}Y$; sinon, $x = au$ pour un préfixe u de w, et $u \in a^{-1}X$, d'où $w \in (a^{-1}X)Y$. L'inclusion réciproque se montre de la même manière.

Considérons la dernière formule. Soit $w \in a^{-1}X^*$. Alors $aw \in X^*$, donc $aw = xx'$, avec $x \in X$, $x \neq 1$, et $x' \in X^*$. Mais alors $x = au$, avec $u \in a^{-1}X$, donc $w \in (a^{-1}X)X^*$. Réciproquement, on a par la formule (5.3), l'inclusion $(a^{-1}X)X^* \subset a^{-1}(XX^*)$, donc $(a^{-1}X)X^* \subset a^{-1}X^*$. ∎

Exemple. Calculons, à l'aide de ces formules, le langage $a^{-1}(b^*aA^*)$. On a

$$a^{-1}(b^*aA^*) = a^{-1}(b^*(aA^*)) = (a^{-1}b^*)aA^* \cup a^{-1}(aA^*)$$

par (5.3) ; or le premier terme de l'union est vide par (5.3), d'où

$$a^{-1}(b^*aA^*) = (a^{-1}a)A^* \cup (a \cap 1)a^{-1}A^* = (a^{-1}a)A^* = A^*$$

9.5.2 Equivalence de Nerode

Dans ce paragraphe, tous les automates considérés sont déterministes, accessibles et complets.

Soit $\mathcal{A} = (Q, i, T)$ un automate sur un alphabet A reconnaissant un langage X. Pour tout état q, on pose

$$L_q(\mathcal{A}) = \{w \in A^* \mid q \cdot w \in T\}$$

C'est donc l'ensemble des mots reconnus par l'automate $\mathcal{A}$ en prenant q comme état initial. Bien entendu, le langage X reconnu par $\mathcal{A}$ coïncide avec $L_i(\mathcal{A})$. On a vu, au paragraphe précédent, que

$$\{L_q(\mathcal{A}) \mid q \in Q\} = Q(X)$$

La correspondance s'établit par

$$L_q(\mathcal{A}) = u^{-1}X \qquad \text{si } i \cdot u = q$$

Notons que, plus généralement,

$$L_{q\cdot v}(\mathcal{A}) = v^{-1}L_q(\mathcal{A}) \tag{5.5}$$

En effet, $w \in L_{q\cdot v}(\mathcal{A})$ si et seulement si $(q \cdot v) \cdot w \in T$ donc si et seulement si $vw \in L_q(\mathcal{A})$. Lorsque l'automate est fixé, on écrira L_q au lieu de $L_q(\mathcal{A})$. Deux états $p, q \in Q$ sont dits *inséparables* si $L_p = L_q$, ils sont *séparables* sinon. Ainsi, p et q sont séparables si et seulement s'il existe un mot w tel que $p \cdot w \in T$ et $q \cdot w \notin T$ ou vice-versa. Si w est un mot ayant cette propriété, on dit qu'il *sépare* les états p et q. L'*équivalence de Nerode* sur Q (ou sur $\mathcal{A}$) est la relation $\sim$ définie par

$$p \sim q \iff p \text{ et } q \text{ sont inséparables}$$

Proposition 5.5. *Dans l'automate minimal, deux états distincts sont séparables, et l'équivalence de Nerode est l'égalité.*

Preuve. Soit $Y = u^{-1}X$ un état de l'automate minimal $\mathcal{A}(X)$ du langage X. Comme $Y = X \cdot u$, on a $L_Y = u^{-1}X = Y$. Par conséquent, deux états distincts ne sont pas équivalents. ∎

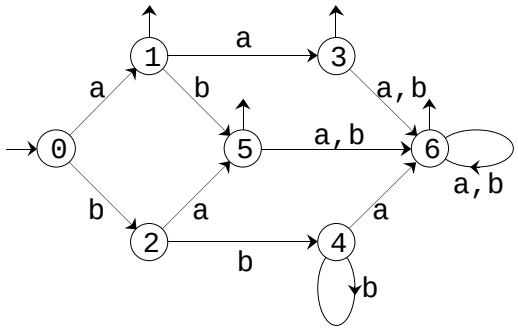

Figure 5.3: *Un automate qui n'est pas minimal.*

Exemple. Reprenons l'automate donné dans la figure 5.3 ci-dessous, et déjà considéré dans le paragraphe précédent. Puisque

$$L_1 = L_3 = L_5 = L_6 = \{a, b\}^*$$
$$L_0 = L_2 = L_4 = b^*a\{a, b\}^*$$

l'équivalence de Nerode a donc les deux classes $\{0, 2, 4\}$ et $\{1, 3, 5, 6\}$. Le mot vide sépare deux états pris dans des classes distinctes.

Proposition 5.6. *L'équivalence de Nerode est une relation d'équivalence régulière à droite, c'est-à-dire vérifiant*

$$p \sim q \Rightarrow p \cdot u \sim q \cdot u \qquad (u \in A^*)$$

De plus, une classe de l'équivalence ou bien ne contient pas d'états terminaux, ou bien ne contient que des états terminaux.

Preuve. Soit $\mathcal{A} = (Q, i, T)$ un automate. Il est clair que la relation $\sim$ est une relation d'équivalence. Pour montrer sa régularité, supposons $p \sim q$, et soit $u \in A^*$. En vue de (5.5), on a $L_{q\cdot u} = u^{-1}L_q = u^{-1}L_p = L_{p\cdot u}$, donc $p \cdot u \sim q \cdot u$. Supposons enfin $p \sim q$ et p terminal. Alors $1 \in L_p$, et comme $L_p = L_q$, on a $1 \in L_q$, donc q est terminal. ∎

L'équivalence de Nerode sur un automate $\mathcal{A} = (Q, i, T)$ étant régulière à droite, on peut définir un *automate quotient* en confondant les états d'une même classe. Plus précisément, notons $[q]$ la classe d'un état q dans l'équivalence. L'automate quotient est alors défini par

$$\mathcal{A}/\!\sim = (Q/\!\sim, [i], \{[t] : t \in T\})$$

avec la fonction de transition

$$[q \cdot a] = [q] \cdot a \tag{5.6}$$

qui justement est bien définie (indépendante du représentant choisi dans la classe de q) parce que l'équivalence est régulière à droite.

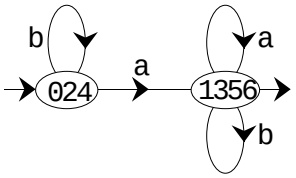

Figure 5.4: *Un automate quotient.*

Dans l'exemple ci-dessus, l'automate quotient a deux états : l' état $\{0, 2, 4\}$ est initial, et $\{1, 3, 5, 6\}$ est l'état final. La formule (5.6) permet de calculer les transitions. On obtient l'automate de la figure 5.4.

Comme le suggère cet exemple, l'automate quotient est en fait l'automate minimal, à une renumérotation des états près. Nous avons besoin, pour le prouver, de la notion d'isomorphisme d'automates. Soient $\mathcal{A} = (Q, i, T)$ et $\mathcal{A}' = (Q', i', T')$ deux automates sur A. Ils sont dits *isomorphes* s'il existe une bijection

$$\alpha : Q \to Q'$$

telle que $\alpha(i) = i'$, $\alpha(T) = T'$, et $\alpha(q \cdot a) = \alpha(q) \cdot a$ pour tout $q \in Q$ et $a \in A$.

Proposition 5.7. *Soit $\mathcal{A} = (Q, i, T)$ un automate sur un alphabet A reconnaissant un langage X. Si l'équivalence de Nerode de $\mathcal{A}$ est l'égalité, alors $\mathcal{A}$ et l'automate minimal $\mathcal{A}(X)$ sont isomorphes.*

Preuve. Soit $\alpha : Q \to Q(X)$ définie par $\alpha(q) = L_q(\mathcal{A})$. Comme l'équivalence de Nerode est l'égalité, α est une bijection. Par ailleurs, $\alpha(i) = L_i = X$, et $t \in T$ si et seulement si $1 \in L_t(\mathcal{A})$, donc si et seulement si $L_t(\mathcal{A})$ est état final de $\mathcal{A}(X)$. Enfin, on a

$$L_{q \cdot a}(\mathcal{A}) = L_q(\mathcal{A}) \cdot a$$

montrant que α est bien un morphisme. ∎

De cette propriété, on déduit une conséquence importante :

Théorème 5.8. *L'automate minimal d'un langage X est l'automate ayant le moins d'états parmi les automates déterministes complets qui reconnaissent X. Il est unique à un isomorphisme près.*

Preuve. Soit $\mathcal{B}$ un automate reconnaissant X et ayant un nombre minimal d'états. Alors son équivalence de Nerode est l'égalité, sinon on pourrait passer au quotient par son équivalence de Nerode et trouver un automate plus petit. Par la proposition précédente, $\mathcal{B}$ est isomorphe à $\mathcal{A}(X)$. ∎

L'unicité de l'automate minimal ne vaut que pour les automates déterministes. Il existe des automates non déterministes, non isomorphes, ayant un nombre minimal d'états, et reconnaissant un même langage (voir exercices).

9.6 Calcul de l'automate minimal

Le calcul de l'automate minimal peut se faire par un procédé appelé la construction de Moore et que nous exposons dans le premier paragraphe. Nous verrons dans la suite une implémentation sophistiquée de cette construction.

9.6.1 Construction de Moore

Soit $\mathcal{A} = (Q, i, T)$ un automate déterministe, accessible et complet sur un alphabet A. Pour calculer l'automate minimal, il suffit de calculer l'équivalence de Nerode définie, rappelons-le, par

$$p \sim q \iff L_p = L_q$$

où $L_p = \{w \in A^* \mid p \cdot w \in T\}$. Pour calculer cette équivalence, on procède par approximations successives. On considère pour ce faire l'équivalence suivante, où k est un entier naturel :

$$p \sim_k q \iff L_p^{(k)} = L_q^{(k)}$$

avec

$$L_p^{(k)} = \{w \in L_p \mid |w| \leq k\}$$

L'équivalence de Nerode est l'intersection de ces équivalences. La proposition suivante exprime l'équivalence $\sim_k$ au moyen de l'équivalence $\sim_{k-1}$. Elle permettra de prouver que l'on obtient bien «à la limite» l'équivalence de Nerode, et elle donne aussi un procédé de calcul.

Proposition 6.1. *Pour tout entier $k \geq 1$, on a*

$$p \sim_k q \iff p \sim_{k-1} q \quad \text{et} \quad (\forall\, a \in A, \quad p \cdot a \sim_{k-1} q \cdot a)$$

Preuve. On a

$$\begin{aligned} L_p^{(k)} &= \{w \mid p \cdot w \in T \text{ et } |w| \leq k\} \\ &= \{w \mid p \cdot w \in T \text{ et } |w| \leq k-1\} \\ &\qquad \cup \bigcup_{a \in A} a\{v \mid (p \cdot a) \cdot v \in T \text{ et } |v| \leq k-1\} \\ &= L_p^{(k-1)} \cup \bigcup_{a \in A} a L_{p \cdot a}^{(k-1)} \end{aligned}$$

L'observation n'est qu'une traduction de ces égalités. ∎

Corollaire 6.2. *Si les équivalences $\sim_k$ et $\sim_{k+1}$ coïncident, les équivalences $\sim_{k+l}$ ($l \geq 0$) sont toutes égales, et égales à l'équivalence de Nerode.*

Preuve. De la proposition, il résulte immédiatement que l'égalité des équivalences $\sim_k$ et $\sim_{k+1}$ entraîne celle des équivalences $\sim_{k+1}$ et $\sim_{k+2}$. D'où la première assertion. Comme $p \sim q$ si et seulement si $p \sim_k q$ pour tout $k \geq 0$, la deuxième assertion s'en déduit. ∎

Proposition 6.3. *Si $\mathcal{A}$ est un automate à n états, l'équivalence de Nerode de $\mathcal{A}$ est égale à $\sim_{n-2}$.*

Preuve. Si, pour un entier $k > 0$, les équivalences $\sim_{k-1}$ et $\sim_k$ sont distinctes, le nombre de classes de l'équivalence $\sim_k$ est $\leq k + 2$. ∎

9.6.2 Scinder une partition

La construction de l'automate minimal par la méthode de Moore appliquée directement à un automate à n états sur un alphabet à m lettres fournit un algorithme qui a un temps de calcul dans le pire des cas en $O(mn^2)$. Nous présentons ici un algorithme dû à Hopcroft, dont la preuve a été simplifiée par D. Gries, et dont la complexité en temps est $O(mn \log n)$.

Soient $\mathcal{R}$ et $\mathcal{R}'$ deux relations d'équivalence sur un ensemble Q. On dit que $\mathcal{R}$ *est plus fine que* $\mathcal{R}'$, ce que l'on note $\mathcal{R} \leq \mathcal{R}'$, si : $p\mathcal{R}q \Rightarrow p\mathcal{R}'q$.

Soient $\mathcal{P}$ et $\mathcal{P}'$ les partitions associées à $\mathcal{R}$ et $\mathcal{R}'$, on a alors :

$$\mathcal{R} \leq \mathcal{R}' \text{ si et seulement si } \forall P \in \mathcal{P},\ \exists P' \in \mathcal{P}',\ P \subset P'.$$

Soit A un ensemble opérant à droite sur Q par une application de $Q \times A$ dans Q où l'image du couple (q, a) est notée $q \cdot a$ (c'est le cas de la fonction de transition d'un automate déterministe complet). On note alors $a^{-1}S = \{q \in Q \mid q \cdot a \in S\}$. Pour toute partie S de Q, on note $\bar{S}$ la partie $Q - S$. Remarquons que $Q = a^{-1}S \cup a^{-1}\bar{S}$, et que cette union est disjointe.

On peut alors établir un certain nombre de propriétés :

Soient $\mathcal{P}$ une partition de Q, P et S deux éléments de $\mathcal{P}$ et a un élément de A.

On dit que P est *stable pour* (S, a) si on a :

$$P \cdot a \subset S \text{ ou } P \cdot a \subset \bar{S}$$

ou encore de manière équivalente si :

$$P \subset a^{-1}S \text{ ou } P \subset a^{-1}\bar{S}$$

Si P n'est pas stable pour (S, a), on dit que P est *brisée* par (S, a) ; cela signifie que les ensembles

$$P \cap a^{-1}S \text{ et } P \cap a^{-1}\bar{S}$$

sont tous les deux non vides. On définit une opération SCINDER de la manière suivante : SCINDER$(P,(S,a))$ encore noté $P \underset{a}{\lhd} S$ est défini par :

$$P \underset{a}{\lhd} S = \begin{cases} \{P\} & \text{si } P \text{ est stable pour } (S,a) \\ \{P \cap a^{-1}S, P \cap a^{-1}\bar{S}\} & \text{sinon} \end{cases}$$

On peut noter aussi que l'on a : $P \underset{a}{\lhd} S = P \underset{a}{\lhd} \bar{S}$.

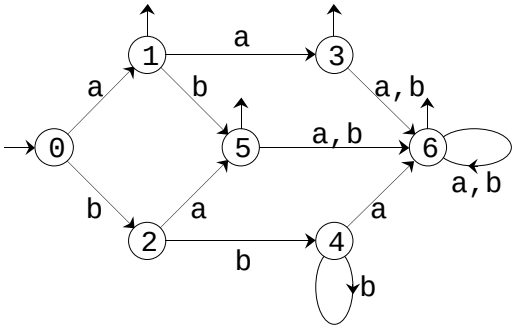

Figure 6.1: *Un exemple illustrant l'opération* $\lhd$.

Exemple. Considérons l'automate déterministe complet de la figure 6.1. La partie Q n'est pas stable pour (T,b) car $Q \cdot b = \{2,4,5,6\}$; or $2 \notin T$, et $5 \in T$. On a alors :

$$Q \underset{b}{\lhd} T = \{\{1,3,5,6\},\{0,2,4\}\}.$$

Par contre $\bar{T}$ est stable pour (T,a) et pour (T,b), car $\bar{T} \cdot a \subset T$ et $\bar{T} \cdot b \subset \bar{T}$.

Lemme 6.4. *Soit A un ensemble opérant à droite sur Q, soient P,S,T des parties de Q et $a,b \in A$.*

i) l'opération SCINDER *est «commutative», i.e. :*

$$P \underset{a}{\lhd} S \underset{b}{\lhd} T = P \underset{b}{\lhd} T \underset{a}{\lhd} S$$

ii) l'opération SCINDER *est «monotone» , i.e. si $\{S_1,S_2\}$ est une partition de S alors :*

$$P \underset{a}{\lhd} S \underset{a}{\lhd} S_1 = P \underset{a}{\lhd} S_2 \underset{a}{\lhd} S_1.$$

Preuve. i) $P \underset{a}{\lhd} S$ est formé des ensembles non vides parmi $P \cap a^{-1}S$, $P \cap a^{-1}\bar{S}$, et $P \underset{a}{\lhd} S \underset{b}{\lhd} T$ est composé des ensembles non vides parmi :

$$P \cap a^{-1}S \cap b^{-1}T, \quad P \cap a^{-1}\bar{S} \cap b^{-1}T, \quad P \cap a^{-1}S \cap b^{-1}\bar{T}, \quad P \cap a^{-1}\bar{S} \cap b^{-1}\bar{T}.$$

On constate aisément que la permutation de S et T et de a et b donne le même résultat.

ii) Comme l'opération SCINDER est commutative, il suffit de comparer $P \lhd_a S_1 \lhd_a S$ à $P \lhd_a S_1 \lhd_a S_2$. Il est facile de vérifier que les opérations $\bullet \lhd_a S_2$ et $\bullet \lhd_a S$ ne modifient pas la partition $P \lhd_a S_1$, d'où le résultat cherché. ∎

Une partition $\mathcal{P}$ est *stable pour* (S, a) si les classes qui la composent le sont. On définit SCINDER$(\mathcal{P}, (S, a))$ que l'on note aussi $\mathcal{P} \lhd_a S$ comme étant la partition $\mathcal{P}'$ obtenue en remplaçant dans $\mathcal{P}$ chaque classe P par SCINDER$(P, (S, a))$.

L'opération SCINDER agissant sur une partition a les propriétés suivantes :

Lemme 6.5. *i) Si une partition $\mathcal{P}$ de Q est stable pour (S, a), alors toute partition plus fine que $\mathcal{P}$ l'est également.*

ii) La partition $\mathcal{P}' = \mathcal{P} \lhd_a S$ est stable pour (S, a), plus fine que $\mathcal{P}$, et si $\mathcal{P}$ n'est pas stable pour (S, a), alors $\mathcal{P}'$ est strictement plus fine que $\mathcal{P}$.

Preuve. Les preuves sont faciles et laissées au lecteur à titre d'exercice. ∎

Remarques.

– Il est facile de vérifier que les propriétés énoncées au lemme 6.4 restent vraies si l'on remplace P par une partition $\mathcal{P}$.

– La notation $P \lhd_a S \lhd_b T$ peut s'étendre à une liste quelconque $\mathcal{L} = ((S_1, a_1), \ldots, ((S_k, a_k))$ de couples, et on écrira alors $P \lhd \mathcal{L}$ ou bien SCINDER$(P, \mathcal{L})$.

Notons que la régularité à droite d'une relation d'équivalence sur Q peut s'énoncer en termes de stabilité de la façon suivante :

Lemme 6.6. *Une relation d'équivalence sur Q est régulière à droite si et seulement si la partition $\mathcal{P}$ associée est stable pour tous les couples (P, a) où $P \in \mathcal{P}$ et $a \in A$.* ∎

9.6.3 Algorithme de Hopcroft

Revenons à l'équivalence de Nerode. Soit $\mathcal{A} = (Q, i, T)$ un automate déterministe complet à n états sur un alphabet A à m lettres. Rappelons que l'équivalence de Nerode sur Q est définie par :

$$p \sim q \Longleftrightarrow L_p = L_q$$

où L_p est l'ensemble des étiquettes des chemins de i à p.

Note 9.6.2

On appelle *partition de Nerode* la partition de Q associée à cette relation d'équivalence. On a vu (proposition 5.6) que l'équivalence de Nerode est régulière à droite.

Nous supposons désormais que l'ensemble T d'états terminaux de $\mathcal{A}$ est non vide et distinct de Q. Considérons la partition de Q, $\mathcal{P}_0 = \{T, \bar{T}\}$, et $\mathcal{R}_0$ la relation d'équivalence associée. Compte tenu des définitions données, on peut énoncer la proposition suivante :

Proposition 6.7. *L'équivalence de Nerode est l'équivalence régulière à droite la plus grossière qui soit plus fine que $\mathcal{R}_0$.*

Preuve. Soit $\mathcal{R}$ l'équivalence de Nerode. D'après la proposition 5.6, il est clair que $\mathcal{R}$ est régulière à droite et plus fine que $\mathcal{R}_0$.

Soit $\mathcal{R}'$ une relation d'équivalence régulière à droite et plus fine que $\mathcal{R}_0$. Supposons que $\mathcal{R}'$ ne soit pas plus fine que $\mathcal{R}$. Dans ce cas il existe $p, q \in Q$ tels que $p\mathcal{R}'q$ et $L_p \neq L_q$. Alors on peut supposer par exemple qu'il existe $u \in L_p$ et $u \notin L_q$. Soit $t = p \cdot u$ et $q' = q \cdot u$. On a $t \in T$, $q' \notin T$ et $t\mathcal{R}'q'$ puisque $p\mathcal{R}'q$. Ce qui induit une contradiction puisque $\mathcal{R}_0 \leq \mathcal{R}'$. ∎

Une partition $\mathcal{P}$ de l'ensemble Q sera dite *admissible* si la relation d'équivalence associée est plus fine que $\mathcal{R}_0$ et moins fine que l'équivalence de Nerode. La proposition 6.7 devient :

Lemme 6.8. *Une partition de l'ensemble Q est la partition de Nerode de l'automate $\mathcal{A}$ si et seulement si elle est admissible et stable.* ∎

Par ailleurs on déduit facilement de ce qui précède la propriété suivante :

Lemme 6.9. *Si $\mathcal{P}$ est une partition admissible, non stable pour le couple (P, a), alors $\mathcal{P} \underset{a}{\lhd} P$ est une partition admissible.* ∎

Pour calculer l'automate minimal de l'automate $\mathcal{A} = (Q, i, T)$, nous allons calculer son équivalence de Nerode et, pour cela, calculer la partition de Nerode associée à cette relation d'équivalence.

Le principe de l'algorithme est le suivant. On part de la partition $\mathcal{P}_0 = \{T, \bar{T}\}$ que l'on raffine par bris successifs; plus précisément, tant que $\mathcal{P}$ (au début $\mathcal{P} = \mathcal{P}_0$) n'est pas stable, on brise une classe de la partition.

Première écriture de l'algorithme

Soient $\mathcal{A} = (Q, i, T)$ un automate sur A et $\mathcal{P}_0 = \{T, \bar{T}\}$ la partition initiale de Q. Considérons l'algorithme suivant :

```
procédure MOORE1 (A);
  P := P_0;
  tantque ∃P ∈ P et a ∈ A tels que P n'est pas stable pour (P, a) faire
    P := P ◁_a P
  fintantque.
```

La boucle «tantque» est exécutée au plus $n-1$ fois, car à chaque exécution le nombre d'éléments de $\mathcal{P}$ augmente strictement et est majoré par n. La partition $\mathcal{P}$ obtenue est stable. Par ailleurs, «*P est admissible*» est un invariant de la boucle «tantque» en raison du lemme 6.9.

Comme $\mathcal{P}_0$ est admissible, cela implique que la partition obtenue à l'arrêt de la boucle est admissible, c'est donc la partition de Nerode. D'où :

Proposition 6.10. *La procédure* MOORE1 *calcule l'équivalence de Nerode.*

∎

Nous modifions la première procédure en transformant l'itération de la façon suivante : à chaque entrée dans la boucle on dresse la liste $\mathcal{L}$ des couples (P, a) pour lesquels $\mathcal{P}$ n'est pas stable, et on remplace $\mathcal{P}$ par $\text{SCINDER}(\mathcal{P}, \mathcal{L})$. Notons qu'on ne fait plus la même suite de scissions que dans la procédure MOORE1, car si $\mathcal{L} = ((P_1, a_1), \ldots, (P_k, a_k))$, et si l'on suppose que P_2 a été scindé lors de la scission relative à (P_1, a_1), alors dans la procédure MOORE1 on ne fera pas de scission relativement à (P_2, a_2) parce que P_2 n'appartient plus à la partition. Il est facile de voir que cette modification n'altère pas la validité de l'algorithme. On peut énoncer le résultat ainsi :

Lemme 6.11. *Si $\mathcal{P}$ est une partition admissible et $\mathcal{L}$ est la liste des couples (P, a) ($P \in \mathcal{P}$, $a \in A$) pour lesquels $\mathcal{P}$ n'est pas stable, alors* $\text{SCINDER}(\mathcal{P}, \mathcal{L})$ *est encore une partition admissible.*

Preuve. Il suffit de prouver que si $\mathcal{P}$ est admissible et que P est une union d'éléments de $\mathcal{P}$ alors $\mathcal{P} \lhd_a P$ est encore admissible. La preuve est similaire à celle de la Proposition 6.10 et est laissée au lecteur. ∎

On peut donc remplacer la procédure MOORE1 par la suivante : Note 9.6.3

```
procédure MOORE2 (A);
  P := P_0;
  tantque P n'est pas stable faire
    calculer la liste L des couples (P, a) qui brisent P;
    PP ◁ L
  fintantque.
```

Donnons un exemple pour montrer que la suite des scissions n'est pas la même dans les procédures MOORE1 et MOORE2.

Exemple. Considérons l'automate de la figure 6.2.

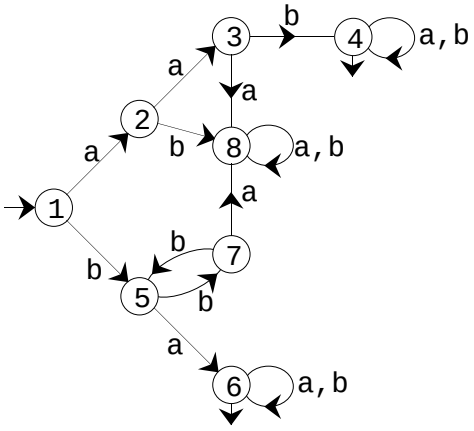

Figure 6.2: *Exemple.*

Note 9.6.4

La procédure MOORE1 donne la suite de scissions suivantes :

$\mathcal{P} : 46 - 123578$
Scission relative à $(46, a)$ — $\mathcal{P} : 46 - 12378 - 5$
Scission relative à $(5, b)$ — $\mathcal{P} : 46 - 17 - 28 - 3 - 5$
Scission relative à $(28, a)$ — $\mathcal{P} : 46 - 1 - 7 - 2 - 8 - 3 - 5$
Terminé

La procédure MOORE2 donne la suite de scissions suivantes :

$\mathcal{P} : 46 - 123578$
$\mathcal{L} = ((46, a), (46, b), (123578, a), (123578, b))$
Scission relative à $(46, a)$ — $\mathcal{P} : 46 - 12378 - 5$
Scission relative à $(46, b)$ — $\mathcal{P} : 46 - 1278 - 3 - 5$
Scission relative à $(123578, a)$ — $\mathcal{P}$: pas de changement
Scission relative à $(123578, b)$ — $\mathcal{P}$: pas de changement
$\mathcal{L} = ((1278, a)(1278, b), (3, a), (5, b))$
Scission relative à $(1278, a)$: — $\mathcal{P} : 46 - 178 - 2 - 3 - 5$
Scission relative à $(1278, b)$: — $\mathcal{P} : 46 - 17 - 8 - 2 - 3 - 5$
Scission relative à $(3, a)$: — $\mathcal{P}$: pas de changement
Scission relative à $(5, b)$: — $\mathcal{P} : 46 - 1 - 7 - 8 - 2 - 3 - 5$
$\mathcal{L} = \emptyset$
Terminé

L'automate minimal obtenu est donc le suivant et reconnaît le langage $\{a^2b, b(b^2)^* a\}\{a, b\}^*$:

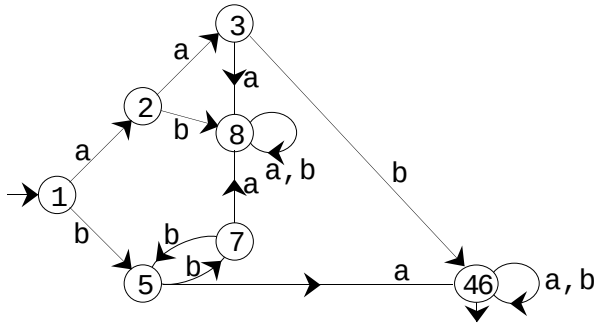

Figure 6.3: *Automate minimal obtenu à partir de celui de la figure 6.2.*

Version définitive de l'algorithme

Contrairement à ce que laisserait croire l'exemple traité, l'écriture de l'algorithme sous cette forme va nous permettre de mieux exploiter les propriétés de l'opération SCINDER pour améliorer les performances de l'algorithme.

Dans un premier temps, au lieu de calculer les couples (P, a) pour lesquels $\mathcal{P}$ n'est pas stable, on peut prendre $\mathcal{L} = \mathcal{P} \times A$; cela revient à ajouter des couples pour lesquels $\mathcal{P}$ est stable et pour lesquels l'opération SCINDER n'aura aucun effet. Ceci fait, en vertu du lemme 6.5 iv), on peut remplacer à la première itération $\mathcal{L} := \{\bar{T}, T\} \times A$ par $\mathcal{L} := \{T\} \times A$ ou $\mathcal{L} := \{\bar{T}\} \times A$. On choisira entre T et $\bar{T}$, la partie de plus faible cardinal, pour des raisons qui s'éclairciront plus tard mais dont on devine d'ores et déjà l'intérêt (réduction de la taille des ensembles relativement auxquels on fait la scission). Ceci nous amène à écrire la version quasiment définitive de l'algorithme. Ecrivons l'algorithme sous la forme suivante :

procédure HOPCROFT $(\mathcal{A})$;
  $\mathcal{P} := \{T, \bar{T}\}$;
  si Card$(T) <$ Card$(\bar{T})$ alors $\mathcal{L} := \{T\} \times A$ sinon $\mathcal{L} := \{\bar{T}\} \times A$;
  tantque $\mathcal{L} \neq \emptyset$ faire
  (a) enlever un couple (P, a) de $\mathcal{L}$;
  (b) déterminer l'ensemble $\mathcal{B}$ des éléments de $\mathcal{P}$ brisés par (P, a);
  (c) pour tout $B \in \mathcal{B}$ calculer $\{B', B''\} = B \underset{a}{\lhd} P$;
  (d) pour tout $B \in \mathcal{B}$ et pour tout $b \in A$ faire
        si $(B, b) \in \mathcal{L}$ alors
  (e)   enlever (B, b) et ajouter (B', b) et (B'', b) à $\mathcal{L}$
  (f)   sinon si Card$(B') <$ Card(B'') alors ajouter (B', b) à $\mathcal{L}$
              sinon ajouter (B'', b) à $\mathcal{L}$
        finsi
  fintantque;
  retourner $(\mathcal{P})$.

Dans cette version de l'algorithme, la liste $\mathcal{L}$ représente certains des couples (P, a) pour lesquels il faut scinder $\mathcal{P}$. Mais à un instant donné, $\mathcal{L}$ ne représente pas la liste de *tous* les couples $(P, a),\ P \in \mathcal{P}$ pour lesquels $\mathcal{P}$ n'est pas stable. Néanmoins la propriété suivante (I) est un invariant de la boucle «tantque» :

$$\text{si } P \in \mathcal{P} \text{ et } (P, a) \notin \mathcal{L} \text{ pour un certain } a, \text{ alors}$$

a) $\mathcal{P}$ est stable pour (P, a)

$$\text{ou} \qquad (I)$$

b) à l'arrêt de la boucle «tantque», la partition $\mathcal{P}$ est stable pour (P, a).

Prouvons donc que (I) est un invariant de la boucle «tantque» en utilisant les lemmes 6.4 et 6.5 .

Appelons $\mathcal{P}_k$ la partition obtenue avant la k-ième itération de la boucle, et $\mathcal{P}_N$ la partition finale.

Avant d'entrer dans la boucle «tantque» (I) est vrai : supposons que **a)** soit faux; comme $(P, a) \notin \mathcal{L}$, alors $(\bar{P}, a) \in \mathcal{L}$, et les lemmes 6.4 et 6.5 assurent que les transformations de $\mathcal{L}$ qui ont lieu en (a), (e), ou (f) sont telles que $\mathcal{P}_N$ est stable pour $(\bar{P}, a)$ et donc pour (P, a).

Supposons maintenant qu'avant la k-ième itération, (I) soit vrai et soient respectivement $\mathcal{L}_k$ et $\mathcal{L}_{k+1}$ l'état de $\mathcal{L}$ avant et après cette k-ième itération. Montrons que (I) est vrai pour $\mathcal{P}_{k+1}$ à la sortie de la k-ième itération de la boucle «tantque».

Soit un couple $(P, a) \notin \mathcal{L}_{k+1}$, avec $P \in \mathcal{P}_{k+1}$;

- ou bien $P \in \mathcal{P}_k$;

si $(P, a) \notin \mathcal{L}_j$, par hypothèse de récurrence $\mathcal{P}_k$ est stable pour (P, a) et donc aussi $\mathcal{P}_{k+1}$, ou bien $\mathcal{P}_N$ est stable pour (P, a) (cqfd);

si $(P, a) \in \mathcal{L}_j$, il est facile de voir que nécessairement la k-ième itération est une scission relativement à (P, a) et que P n'a pas été brisé lors de cette scission. La partition $\mathcal{P}_{k+1}$ est donc stable pour (P, a) ainsi que toutes les suivantes (cqfd);

- ou bien $P \notin \mathcal{P}_k$;

alors P a été obtenu lors de la k-ième itération par «bris» d'une partie R en deux parties P et P', avec $R \in \mathcal{P}_k$;

si $(R, a) \notin \mathcal{L}_k$, alors à la sortie de la boucle on a $(P, a) \notin \mathcal{L}_{k+1}$ par hypothèse, et donc $(P', a) \in \mathcal{L}_{k+1}$. Ainsi on est sûr que $\mathcal{P}_N$ sera stable pour (P', a) et par hypothèse de récurrence que $\mathcal{P}_k$ est stable pour (R, a) ou que $\mathcal{P}_N$ sera stable pour (R, a). Donc par application du lemme 6.4 ii), $\mathcal{P}_N$ sera stable pour (P, a) (cqfd);

si $(R, a) \in \mathcal{L}_k$, comme $(P, a) \notin \mathcal{L}_{k+1}$, c'est que la scission faite lors de la k-ième itération est une scission relative à (R, a) (sinon, puisque $(R, a) \in \mathcal{L}_k$, on aurait (P, a) et $(P', a) \in \mathcal{L}_{k+1}$, ce qui contredit $(P, a) \notin \mathcal{L}_{k+1}$). Donc $\mathcal{P}_N$ sera stable pour (R, a), et puisque $(P, a) \notin \mathcal{L}_{k+1}$ c'est que $(P', a) \in \mathcal{L}_{k+1}$, et donc $\mathcal{P}_N$ sera stable pour (P', a). D'où l'on déduit que $\mathcal{P}_N$ sera stable pour (P, a) (cqfd).

On peut donc énoncer le lemme :

Lemme 6.12. *L'algorithme* HOPCROFT *s'arrête et calcule la partition de Nerode.*

Preuve. Il reste uniquement à vérifier que la boucle «tantque» s'arrête. Or, on ne peut ajouter deux fois un même couple (P, a) dans $\mathcal{L}$, car chaque couple (P, a) ajouté dans la boucle en (e) ou (f) est constitué d'une partie P strictement contenue dans une classe de la partition courante, et comme chaque passage dans la boucle supprime un élément de $\mathcal{L}$, la boucle «tantque» s'arrête. ∎

Nous énonçons un dernier lemme utile à l'étude de la complexité de l'algorithme :

Lemme 6.13. *Le nombre d'itérations de la boucle «tantque» est majoré par* $2mn$.

Preuve. Il suffit de prouver que le nombre de couples (P, a) introduits dans $\mathcal{L}$ est au plus $2mn$, puisqu'à chaque itération on enlève un couple de $\mathcal{L}$. Pour chaque couple (P, a) introduit dans $\mathcal{L}$, P est élément de la partition courante. Considérons l'arbre binaire représentant les scissions successives opérées sur $\mathcal{P}$. La racine est la partie Q toute entière et ses fils sont T et $\bar{T}$ (si $T = Q$, on part de T), et chaque nœud P admet pour fils P' et P'', résultats d'une scission appliquée à P. Un tel arbre binaire complet a au plus n feuilles et donc au plus $2n - 1$ sommets. D'où le résultat. ∎

9.6.4 Complexité de l'algorithme

Avant de décrire les structures de données nécessaires à l'implémentation de l'algorithme, nous allons détailler l'écriture des instructions (b), (c) et (d) de la procédure HOPCROFT.

Détaillons l'écriture des instructions (b), (c) et (d) de la procédure HOPCROFT :

On remplace (b) par :

> (b) calculer $a^{-1}P$;
> dresser la liste CLASSES-RENCONTRÉES des classes $B \in \mathcal{P}$ telles que $B \cap a^{-1}P \neq \emptyset$;

Notons que cette liste contient toutes les classes susceptibles d'être brisées, mais certaines d'entre elles peuvent être stables pour (P, a). C'est à l'étape suivante que l'on détermine les classes devant être brisées. Ainsi (c-f) est remplacé par :

```
(c)  pour tout B ∈Classes-Rencontrées faire
       si B · a ⊄ P alors
         créer une nouvelle classe Jumeau(B);
         enlever de B et mettre dans Jumeau(B)
         tous les états p tels que p · a ∈ P;
(d)      pour tout b ∈ A faire
           si avant scission (B, b) ∈ L alors
(e)          ajouter (Jumeau(B),b) à L
(f)        sinon si après scission Card(B) ≤ Card(Jumeau(B)) alors
                     ajouter (B, b) à L
                   sinon ajouter (Jumeau(B),b) à L
           finsi
         finpour
       finsi
     finpour.
```

Considérons une itération de la boucle (c) pour laquelle $B \cdot a \not\subset P$. A la fin de cette itération, B et Jumeau(B) sont exactement les ensembles B' et B''. D'où l'algorithme complet :

```
procédure HOPCROFT (A);
P := {T, T̄};
si Card(T) < Card(T̄) alors L := {T} × A sinon L := {T̄} × A;
tantque L ≠ ∅ faire
(a)  enlever un couple (P, a) de L;
(b)  INVERSE:= a⁻¹P;
     dresser la liste CLASSES-RENCONTRÉES des classes
     B ∈ P telles que B∩INVERSE≠ ∅;
(c)  pour tout B ∈CLASSES-RENCONTRÉES faire
       si B · a ⊄ P alors
         créer une nouvelle classe JUMEAU(B);
         déplacer de B dans JUMEAU(B) tous les états p | p · a ∈ P;
(d)      pour tout b ∈ A faire
           si avant scission (B, b) ∈ L alors
             ajouter (JUMEAU(B),b) à L
           sinon si après scission Card(B) ≤ Card(JUMEAU(B)) alors
                     ajouter (B, b) à L
                   sinon ajouter (JUMEAU(B),b) à L
           finsi
         finpour
       finsi
     finpour
fintantque.
```

Structures de données

• Les *états* de l'automate sont les entiers de 1 à n. Les lettres sont les entiers de 1 à m. Une partition ayant au plus n éléments, un élément d'une partition aura pour nom un entier entre 1 et n, ainsi un couple (P, a) sera représenté comme un élément de $\{1, \ldots, n\} \times \{1, \ldots, m\}$, où la première composante est le nom de la partie P. Comme le nombre de classes grossit au cours du temps, une variable COMPT initialisée à 2 (ou à 1 si $T = Q$) indique quels sont les entiers déjà utilisés pour nommer les classes, à savoir les entiers de 1 à COMPT.

Une partition de Q sera gérée grâce à quatre tableaux indicés par les entiers de 1 à n : CLASSE, PART, CARD, PLACE.

Pour tout état i, CLASSE[i] est le nom de la classe contenant i. Pour toute classe de nom P, PART[P] est un pointeur sur une liste doublement chaînée des éléments de P, et CARD[P] est le nombre d'éléments de la classe P. Enfin, pour tout état i appartenant à la classe de nom P, PLACE[i] est un pointeur sur la «place» de i dans la liste doublement chaînée PART[P] des éléments de la classe de nom P. Cela permet en temps $O(1)$ de supprimer un élément d'une classe et de l'insérer dans une autre.

Exemple. $Q = \{1, \cdots, 6\}$, $\mathcal{P}$:$35 - 1 - 246$

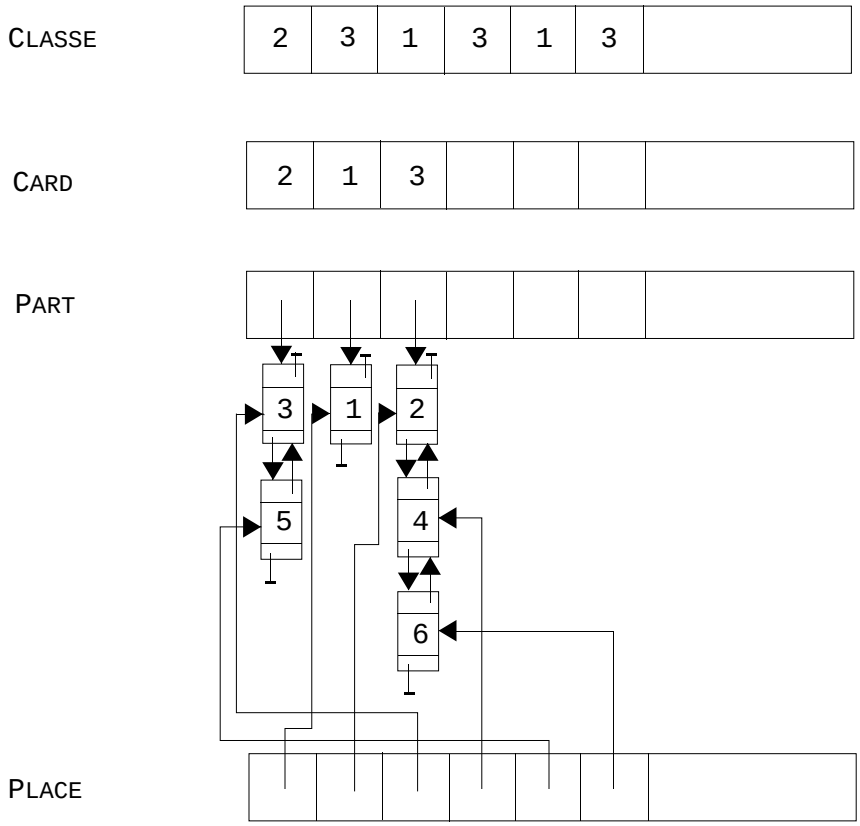

Figure 6.4: *Un exemple.*

• Pour calculer INVERSE:= $a^{-1}P$, on dispose d'un tableau INV indicé par $\{1, \ldots, n\} \times \{1, \ldots, m\}$ tel que INV$[p, a]$ est un pointeur sur une liste chaînée des éléments q tels que $q \cdot a = p$. Ainsi INVERSE sera non pas construit «physiquement», mais «lu» en temps proportionnel à sa taille en parcourant la liste PART$[P]$ et, pour chaque élément p de cette liste, en parcourantINV$[p, a]$. Ainsi l'expression «pour tout q dans INVERSE» est programmée par : pour tout $p \in P$ et pour tout $q \in$INV$[p, a]$.

• La liste $\mathcal{L}$ doit être gérée de telle sorte que les opérations d'adjonction, de recherche, et de suppression d'un élément *arbitraire* soient aisées (arbitraire fait référence ici à la ligne (1) de la procédure HOPCROFT, c'est-à-dire qu'on veut supprimer un élément de $\mathcal{L}$, peu importe lequel). Comme la taille de $\mathcal{L}$ est majorée par mn, on la représentera par un tableau de booléens LISTE indicé par $\{1, \ldots, n\} \times \{1, \ldots, m\}$ tel que LISTE$[P, a]$ est vrai si $(P, a) \in \mathcal{L}$, et simultanément par une liste chaînée. Ainsi, l'adjonction se fait en temps $O(1)$, (la mise à jour dans le tableau et dans la liste chaînée prenant un temps constant) ; la recherche se fait en temps $O(1)$ en utilisant le tableau ; enfin la suppression d'un élément arbitraire (ligne (a) de la procédure) se fait en deux temps, on enlève le premier élément de la liste chaînée puis on met à jour le tableau, ce qui prend un temps

$O(1)$, de même le test $\mathcal{L} \neq \emptyset$ se fait en temps $O(1)$ grâce à la structure de liste chaînée.

• Le traitement de Classes-Rencontrées et de Jumeau est relativement délicat.

Pour dresser la liste de Classes-Rencontrées, et pour gérer les scissions, on utilise trois structures de données :

une liste chaînée Classes-Rencontrées des noms des classes à scinder,

un tableau d'entiers Partage indicé par $\{1, \ldots, n\}$, tel que pour toute classe B de nom i, Partage$[i]$ est le cardinal de $B \cap a^{-1}P$ (le tableau Partage doît être à 0 à chaque entrée dans la boucle «tantque»)

et un tableau Jumeau indicé par $\{1, \ldots, n\}$. L'élément Jumeau$[i]$ est le nom de la nouvelle classe créée pour briser la classe de nom i. Ce tableau est initialisé à 0 au début de l'algorithme.

Analyse de la complexité

Analysons le temps d'exécution de la procédure Hopcroft avec les structures de données proposées ci-dessus. Notons au passage que pour obtenir une faible complexité en temps il a été nécessaire de ne pas lésiner sur l'espace mémoire utilisé.

L'initialisation qui précède la boucle «tantque» prend un temps $O(mn)$. Le nombre d'itérations de la boucle «tantque» étant majoré par $2mn$, le temps global d'exécution de la ligne (a) est $O(mn)$.

On a vu dans la preuve du lemme 6.13 que le nombre de classes créées est majoré par $2n-1$, donc le temps global d'exécution du bloc (d) prend un temps $O(mn)$.

Il reste à évaluer le temps global d'exécution des blocs (b) et (c). Soit N le nombre de passages dans la boucle «tantque». Dans un premier temps, évaluons la somme des tailles des N listes «abstraites» Inverse parcourues.

Proposition 6.14. *Soient $a \in A$, $p \in Q$. Le nombre de fois où l'on supprime de la liste $\mathcal{L}$ un couple (P, a) tel que $p \in P$, est majoré par $\log_2 n$.*

Preuve. On dira qu'un couple (P, a) est *marqué* si $p \in P$ (a et p sont fixés). Avant d'entrer dans la boucle «tantque», $\mathcal{L}$ contient au plus un couple marqué, et ceci reste vrai à tout instant. Soit (P, a) le premier couple marqué qui soit supprimé de $\mathcal{L}$. Le prochain couple (P'', a) marqué qui sera introduit dans $\mathcal{L}$ dans la boucle «tantque» est tel que P'' est une classe résultat de Scinder(P', b), pour un $b \in A$ où P' est une partie de P et $\mathrm{Card}(P'') \leq \mathrm{Card}(P')/2$. Tant qu'il n'y a pas de suppression dans $\mathcal{L}$ d'un couple marqué, $\mathcal{L}$ contient un unique couple marqué dont la première composante est un sous-ensemble de P'', donc sa taille est inférieure ou égale à celle de P''. Ainsi, le prochain couple marqué qui sera supprimé dans $\mathcal{L}$

aura une taille inférieure ou égale à $\mathrm{Card}(P)/2$. En itérant ce processus, et tenant compte du fait que le premier couple marqué introduit (éventuellement) dans $\mathcal{L}$ vérifie $\mathrm{Card}(P) \leq n/2$, il s'ensuit qu' on ne peut supprimer de $\mathcal{L}$ plus de $\log_2 n$ fois un tel couple. ∎

Corollaire 6.15. *La somme des tailles des N listes* INVERSE *parcourues est majorée par $mn \log_2 n$.*

Preuve. Soit un triplet fixé (p, a, q) tel que $q = p \cdot a$. Le nombre de fois où p est «lu» dans la liste INVERSE parce que $q = p \cdot a$, est exactement égal au nombre de couples (P, a) tels que $q \in P$, qui sont supprimés de la liste $\mathcal{L}$. Or par la proposition 6.14 ce nombre est majoré par $\log_2 n$. D'où le résultat. ∎

Il reste à constater que le temps global d'exécution des blocs (b) et (c) est proportionnel à la somme des tailles des N listes INVERSE.

En réalité, les blocs (b) et (c) s'exécutent en parcourant deux fois la liste INVERSE de la manière suivante :

```
(b) créer une liste vide CLASSES-RENCONTRÉES;
    pour tout p ∈ P et pour tout q ∈INV[p, a] faire
      i :=CLASSE[q];
      si PARTAGE[i] = 0 alors
        PARTAGE[i] := 1;
        ajouter i à CLASSES-RENCONTRÉES
      sinon
        PARTAGE[i] :=PARTAGE[i] + 1
      finsi
    finpour;
```

```
(c) pour tout p ∈ P et pour tout q ∈INV[p, a] faire
      i :=CLASSE[q];
      si PARTAGE[i] <CARD[i] alors
        si JUMEAU[i] = 0 alors
          COMPT:=COMPT+1;JUMEAU[i] :=COMPT
        finsi;
        supprimer q de sa classe et l'insérer dans
        la classe de nom JUMEAU[i]
      finsi
    finpour;
    pour tout j appartenant à CLASSES-RENCONTRÉES faire
      PARTAGE[j] := 0;JUMEAU[j] := 0;
```

En écrivant ainsi les blocs (b) et (c) et compte-tenu des structures de données choisies, il est clair que le temps d'exécution de chacun de ces blocs prend un temps proportionnel à la somme des tailles des N listes INVERSE et donc un temps $O(mn \log n)$.

En conclusion, on peut énoncer le :

Théorème 6.16. *Si A est un alphabet à m lettres et $\mathcal{A}$ un automate à n états sur cet alphabet, la procédure* HOPCROFT *calcule, dans le pire des cas, en temps $O(mn \log n)$ l'automate minimal de $\mathcal{A}$.*

Notes

La théorie des automates, dont nous avons décrit ici les bases, a connu des développements considérables. Un traité substantiel est l'ouvrage de S. Eilenberg :

S. Eilenberg, *Automata, Languages, and Machines*, Vol. A Academic Press, 1974.

Pour les développements récents, notamment en rapport avec l'algorithmique des mots, voir :

D. Perrin, Finite Automata, in : J. van Leeuwen, ed., *Handbook of Theoretical Computer Science, Vol. B*, North-Holland, 1990, 1–57.

Une théorie des langages rationnels et reconnaissables de mots infinis a vu le jour en même temps que la théorie portant sur les mots finis. On pourra consulter le chapitre de W. Thomas dans ce même volume.

Exercices

9.1. Montrer les égalités suivantes pour $X, Y, Z \subset A^*$:

$$\begin{aligned} (XY)^{-1}Z &= Y^{-1}(X^{-1}Z) \\ X^{-1}(YZ^{-1}) &= (X^{-1}Y)Z^{-1} \\ (XZ^{-1})Y^{-1} &= X(YZ)^{-1} \end{aligned}$$

9.2. Décrire un algorithme qui permet de tester si le langage reconnu par un automate donné est vide, fini non vide, ou infini.

9.3. (Lemme de l'étoile) Démontrer que pour tout langage reconnaissable L, il existe un entier N tel que tout mot w de L de longueur $|w| \geq N$ admet une factorisation $w = uxv$ avec $0 < |x| \leq N$ et vérifiant $ux^*v \subset L$.

9.4. Utiliser le lemme de l'étoile pour prouver que les langages $\{a^n b^n \mid n \geq 0\}$, $\{a^n b^p \mid n \geq p \geq 0\}$, $\{a^n b^p \mid n \neq p\}$ ne sont pas reconnaissables.

9.5. Démontrer que si L est un langage reconnaissable, alors l'ensemble des facteurs des mots de L est encore un langage reconnaissable. (Même question pour les préfixes, suffixes.)

9.6. Soient A et B deux alphabets. Un *morphisme* de A^* dans B^* est une application $f : A^* \to B^*$ vérifiant $f(uv) = f(u)f(v)$ pour $u, v \in A^*$.

a) Démontrer que si K est un langage rationnel sur A, alors $f(K)$ est un langage rationnel sur B.

b) Démontrer que $f^{-1}(L)$ est un langage rationnel sur A pour tout langage rationnel L sur B.

9.7. Une *substitution* de A^* dans B^* est une application s de A^* dans l'ensemble des parties de B^* qui vérifie $s(uv) = s(u)s(v)$ pour $u, v \in A^*$, et $s(1) = \{1\}$. Démontrer que si $s(a)$ est un langage rationnel pour tout $a \in A$, alors $s(K)$ est un langage rationnel pour tout langage rationnel K sur A.

9.8. Montrer que les deux automates de la figure 6.5 reconnaissent le même langage.

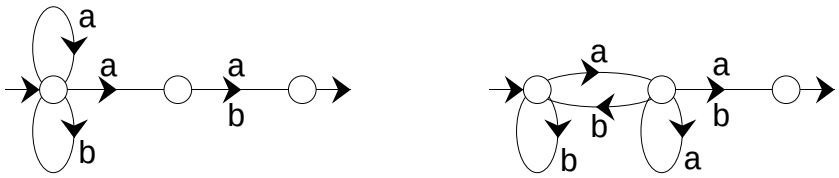

Figure 6.5: *Deux automates non déterministes reconnaissant le même langage.*

Quel est l'automate minimal déterministe reconnaissant ce langage? Plus généralement, montrer que les langages A^*wA^n, avec $A = \{a, b\}$, $n > 0$ sont reconnus par au moins $|w| + 1$ automates non déterministes non isomorphes ayant strictement moins d'états que l'automate (déterministe) minimal de ce langage.

9.9. Soient $\mathcal{A} = (Q, i, T)$ et $\mathcal{A}' = (Q', i', T')$ deux automates déterministes sur un alphaber A. On se propose de tester si ces deux automates sont *équivalents*, c'est-à-dire reconnaissent le même langage.

1) Donner un algorithme résolvant ce problème d'équivalence en utilisant la minimisation des automates déterministes. Quelle est sa complexité?

2) a) Soit $\mathcal{A}'' = (Q'', \{i, i'\}, T \cup T')$ l'automate union des automates $\mathcal{A}$ et $\mathcal{A}'$ (les ensembles Q et Q' sont supposés disjoints). L'automate $\mathcal{A}''$ est déterministe à ceci près qu'il a deux états initiaux. Pour tout $a \in A$, on note δ''_a la restriction à $Q'' \times \{a\}$ de la fonction de transition de l'automate $\mathcal{A}''$. Soit $\mathcal{P}$ la partition la plus fine de l'ensemble Q'' telle que i et i' soient dans la même classe, et qui soit compatible avec la fonction de transition de l'automate $\mathcal{A}''$ i.e. vérifiant $p \sim q \Rightarrow \delta''_a.p \sim \delta''_b.q$ pour tout $p, q \in Q''$ et tout $a \in A$, où $\sim$ désigne la relation d'équivalence associée à la partition considérée.

b) Montrer que les automates $\mathcal{A}$ et $\mathcal{A}'$ sont équivalents si et seulement si l'ensemble $T \cup T'$ est saturé pour la relation d'équivalence associée à la partition $\mathcal{P}$.

c) En déduire un algorithme résolvant le problème d'équivalence de deux automates déterministes. On utilisera pour ce faire l'algorithme « union-find ». Calculer sa complexité

Cet algorithme est dû à Hopcroft et Karp.

9.10. Soit $\mathcal{A} = (Q, F, I, T)$ un un automate fini sur un alphabet A reconnaissant un langage L. Soit $\mathcal{A}^r = (Q, F^r, T, I)$ l'automate obtenu en renversant les flèches de $\mathcal{A}$ (i.e. $(p, a, q) \in F^r \Leftrightarrow (q, a, p) \in F$) et en échangeant états initiaux et terminaux.

1) Montrer que $\mathcal{A}^r$ reconnaît l'image miroir de L notée L^r, i.e. l'ensemble :

$$L^r = \{a_1 \cdots a_n \in A^* \mid \text{ pour tout } \ a_i \in A \text{ et } a_n \cdots a_1 \in L\}$$

2) Soit $\mathcal{A}^r_d$ l'automate déterminste émondé obtenu à partir de $\mathcal{A}^r$ par la construction donnée dans ce chapitre. Montrer que $\mathcal{A}^r_d$ est l'automate déterministe *minimal* reconnaissant L^r.

La preuve de ce résultat est due à J. Brzozowski.

3) En déduire un algorithme de calcul de l'automate déterministe minimal d'un automate donné, et en donner sa complexité.