Chapitre 3

Automates finis

Nous introduisons ici très succinctement les automates finis. Pour les lecteurs intéressés par les aspects formels de la théorie des automates finis, nous recommandons particulièrement la lecture de quelques chapitres de (Hopcroft and Ullman, 1979), ou en français, des chapitres initiaux de (Sakarovitch, 2003). L'exposé nécessairement limité présenté dans les sections qui suivent reprend pour l'essentiel le contenu de (Sudkamp, 1997).

3.1 Automates finis

3.1.1 Bases

Dans cette section, nous introduisons le modèle le plus simple d'automate fini : l'automate déterministe complet. Ce modèle nous permet de définir les notions de calcul et de langage associé à un automate. Nous terminons cette section en définissant la notion d´équivalence entre automates, ainsi que la notion d'utilité d'un état.

Définition 3.1 (Automate fini). *Un* automate fini *(DFA) est défini par un quintuplet* $A = (\Sigma, Q, q_0, F, \delta)$, *où :*
– Σ *est un ensemble fini de symboles (l'alphabet)*
– Q *est un ensemble fini d'états*
– $q_0 \in Q$ *est l'état* initial
– $F \subset Q$ *sont les états* finaux
– δ *est une fonction* totale *de* $(Q \times \Sigma)$ *dans* Q, *appelée* fonction de transition.

La terminologie anglaise correspondante parle de *finite-state automaton* ; comme il apparaîtra plus tard, les automates finis de cette section possèdent la propriété d'être *déterministes* : d'où l'abréviation DFA pour *Deterministic Finite-state Automaton*.

Un automate fini correspond à un graphe orienté, dans lequel certains des nœuds (états) sont distingués et marqués comme initial ou finaux et dans lequel les arcs (transitions) sont étiquetés par des symboles de Σ. Si $\delta(q, a) = r$, on dit que a est *l'étiquette* de la transition (q, r). Les automates admettent une représentation graphique, comme celle de la Figure 3.1.

Dans cette représentation, l'état initial 0 est marqué par un arc entrant sans origine et les états finaux (ici l'unique état final est 2) par un arc sortant sans destination. La fonction de transition correspondant à se graphe s'exprime matriciellement par :

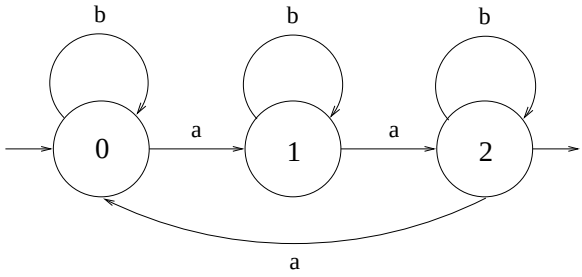

FIG. 3.1 – Un automate fini

δ	a	b
0	1	0
1	2	1
2	0	2

Un *calcul* dans A est une séquence d'états $q_1 \dots q_n$ de A, tel qu'il existe toujours au moins une transition entre deux états successifs q_i et q_{i+1}. L'*étiquette du calcul* est le mot construit par concaténation des étiquettes de chacune des transitions. Un calcul dans A est réussi si l'état d'origine est l'état initial, et si l'état terminal est un des états finaux. Le langage *reconnu* par l'automate A, noté $L(A)$, est l'ensemble des étiquettes des calculs réussis. Dans l'exemple précédent, le mot *baab* appartient au langage reconnu, puisqu'il étiquette le calcul : 00122.

La notation $\vdash_A$ permet de formaliser cette notion. Ainsi on écrira, pour a dans Σ et v dans $\Sigma^\star$: $(q, av) \vdash_A (\delta(q, a), v)$ pour noter une étape de calcul. Cette notation s'étend en $(q, uv) \vdash_A^\star (p, v)$ s'il existe une suite d´états $q = q_1 \dots q_n = p$ tels que $(q_1, u_1 \dots u_n v) \vdash_A (q_2, u_2 \dots u_n v) \dots \vdash_A (q_n, v)$. Avec ces notations, on a :

$$L(A) = \{u \in \Sigma^\star \mid (q_0, u) \vdash_A^\star (q, \varepsilon), \text{ avec } q \in F\}$$

Cette notation met en évidence l'automate comme une machine permettant de *reconnaître* des mots : tout parcours partant de q_0 permet de «consommer» un à un les symboles du mot à reconnaître ; ce processus stoppe lorsque le mot est entièrement consommé : si l'état ainsi atteint est un état final, alors le mot appartient au langage reconnu par l'automate.

On dérive un algorithme permettant de tester si un mot appartient au langage reconnu par un automate fini.

La complexité de cet algorithme découle de l'observation que chaque étape de calcul correspond à une application de la fonction $\delta()$, qui elle-même se réduit à la lecture d'une case d'un tableau et une affectation, deux opérations qui s'effectuent en temps constant. La reconnaissance d'un mot u se calcule en exactement $|u|$ étapes.

Définition 3.2. *Un langage est* reconnaissable *s'il existe un automate fini qui le reconnaît.*

Un second exemple d'automate, très similaire au premier, est représenté à la Figure 3.2. Dans cet exemple, 0 est à la fois initial et final. A reconnaît le langage correspondant aux mots u tels que $|u|_a$ est divisible par 3 : chaque état correspond à une valeur du reste dans la division par 3 : un calcul réussi correspond nécessairement à un reste égal à 0 et réciproquement.

Par un raisonnement similaire à celui utilisé pour définir un calcul de longueur quelconque, il est possible d´étendre récursivement la fonction de transition δ en une fonction $\delta^\star$ de $Q \times \Sigma^\star \to Q$ par :

Algorithm 1 – Reconnaissance par un DFA

// $u = u_1 \dots u_n$ est le mot à reconnaître
// $A = (Q, q_0, \delta, F)$ est le DFA
$q := q_0$
$i := 1$
while $(i \leq n)$ **do**
    $q := \delta(q, u_i)$
    $i := i + 1$
od
if $(q \in F)$ **then** *return*(**true**) **else** *return*(**false**)

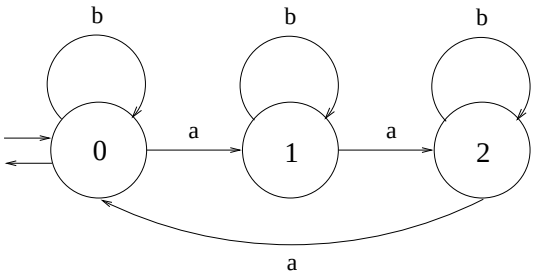

FIG. 3.2 – Un automate fini comptant les a (modulo 3)

– $\delta^\star(q, u) = \delta(q, u)$ si $|u| = 1$
– $\delta^\star(q, au) = \delta^\star(\delta(q, a), u)$
On notera que, puisque δ est une fonction totale, $\delta^\star$ est également une fonction totale : l'image d'un mot quelconque de $\Sigma^\star$ par $\delta^\star$ est toujours bien définie, ie. existe et est unique. Cette nouvelle notation permet de donner une notation alternative pour le langage reconnu par un automate A :

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

Nous avons pour l'instant plutôt vu l'automate comme une machine permettant de reconnaître des mots. Il est également possible de le voir comme un système de *production* : partant de l'état initial, tout parcours conduisant à un état final construit itérativement une séquence d'étiquettes par concaténation des étiquettes rencontrées le long des arcs.

Si chaque automate fini reconnaît un seul langage, la réciproque n'est pas vraie : plusieurs automates peuvent reconnaître le même langage. Comme pour les expressions rationnelles, on dira dans ce cas que les automates sont *équivalents*.

Définition 3.3 (Équivalence entre automates). *Deux automates finis A_1 et A_2 sont* équivalents *si et seulement s'ils reconnaissent le même langage.*

Ainsi, par exemple, l'automate de la Figure 3.3 est-il équivalent à celui de la Figure 3.1 : tous deux reconnaissent le langage de tous les mots qui contiennent un nombre de a congru à 2 modulo 3.

Nous l'avons noté plus haut, $\delta^\star$ est définie pour tout mot de $\Sigma^\star$. Ceci implique que l'algorithme de reconnaissance (1) demande exactement $|u|$ étapes de calcul, correspondant à une exécution complète de la boucle. Ceci peut s'avérer particulièrement inefficace, comme dans l'exemple de l'automate de la Figure 3.4, qui reconnaît le langage $ab\{a, b\}^\star$. Dans ce cas en effet, il est en fait possible d'accepter ou de rejeter des mots en ne considérant que les deux premiers symboles.

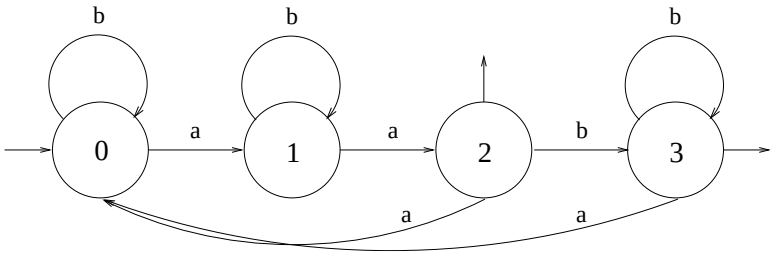

FIG. 3.3 – Un automate fini équivalent à celui de la Figure 3.1

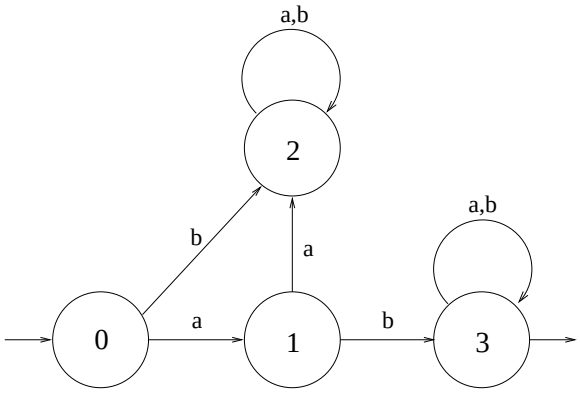

FIG. 3.4 – Un automate fini pour $ab(a+b)^\star$

3.1.2 Spécification partielle

Pour contourner ce problème et pouvoir arrêter le calcul aussi tôt que possible, nous introduisons dans cette section des définitions alternatives, mais qui s'avèrent en fait équivalentes, des notions d'automate et de calcul.

Définition 3.4 (Automate fini). *Un automate fini est défini par un quintuplet* $A = (\Sigma, Q, q_0, F, \delta)$*, où :*
– Σ *est un ensemble fini de symboles (l'alphabet)*
– Q *est un ensemble fini d'états*
– $q_0 \in Q$ *est l'état initial*
– $F \subset Q$ *sont les états finaux*
– δ *est une fonction* partielle *de* $(Q \times \Sigma)$ *dans* Q

La différence avec la définition 3.1 est que δ est ici définie comme une fonction partielle. Son domaine de définition est un sous-ensemble de $Q \times \Sigma$. Selon cette nouvelle définition, il est possible de se trouver dans une situation où un calcul s'arrête avant d'avoir atteint la fin de l'entrée. Ceci se produit dès que l'automate évolue dans une configuration (q, au), telle qu'il n'existe pas de transition étiquetée par a et sortant de l'état q.

La définition (3.4) est en fait strictement équivalente à la précédente, dans la mesure où les automates partiellement spécifiés peuvent être complétés par ajout d'un état «puits» absorbant les transitions absentes de l'automate original, sans pour autant changer le langage reconnu. Formellement, soit $A = (\Sigma, Q, q_0, F, \delta)$ un automate partiellement spécifié, on définit $A' = (\Sigma, Q', q'_0, F', \delta')$ avec :
– $Q' = Q \cup \{q_p\}$
– $q'_0 = q_0$
– $F' = F$
– $\forall q \in Q, a \in \Sigma, \delta'(q,a) = \delta(q,a)$ si $\delta(q,a)$ existe, $\delta'(q,a) = q_p$ sinon.

– $\forall a \in \Sigma, \delta'(q_p, a) = q_p$

L'état puits, q_p, est donc celui dans lequel on aboutit dans A' en cas d'échec dans A ; une fois dans q_p, il est impossible d'atteindre les autres états de A et donc de rejoindre un état final. Cette transformation est illustrée pour l'automate de la Figure 3.5, dont le transformé est précisément l'automate de la Figure 3.4, l´état 2 jouant le rôle de puits.

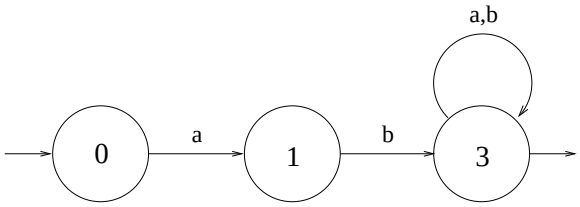

FIG. 3.5 – Un automate partiellement spécifié

A' reconnaît le même langage que A puisque :

– si $u \in L(A)$, $\delta^\star(q_0, u)$ existe et appartient à F. Dans ce cas, le même calcul existe dans A' et aboutit également dans un état final
– si $u \notin L(A)$, deux cas sont possibles : soit $\delta^\star(q_0, u)$ existe mais n'est pas final, et la même chose se produit dans A' ; soit le calcul s'arrête dans A après le préfixe v : on a alors $u = va$ et $\delta(\delta^\star(q_0, v), a)$ n'existe pas. Or, le calcul correspondant dans A' conduit au même état, à partir duquel une transition existe vers q_p. Dès lors, l'automate est voué à rester dans cet état jusqu'à la fin du calcul ; cet état n´étant pas final, le calcul échoue et la chaîne est rejetée.

Pour tout automate (au sens de la définition 3.4), il existe donc un automate complètement spécifié (ou automate *complet*) équivalent.

3.1.3 États utiles

Un second résultat concernant l'équivalence entre automates demande l'introduction des quelques définitions complémentaires suivantes.

Définition 3.5 (Accessibilité, co-accessibilité). *Un état q de A est dit* accessible *s'il existe u dans $\Sigma^\star$ tel que $\delta^\star(q_0, u) = q$. q_0 est trivialement accessible (par $u = \varepsilon$). Un automate dont tous les états sont accessibles est lui-même dit accessible.*

Un état q de A est dit co-accessible *s'il existe u dans $\Sigma^\star$ tel que $\delta^\star(q, u) \in F$. Tout état final est trivialement co-accessible (par $u = \varepsilon$). Un automate dont tous les états sont co-accessibles est lui-même dit co-accessible.*

*Un état q de A est dit*utile *s'il est à la fois accessible et co-accessible. D'un automate dont tous les états sont utiles on dit qu'il est* émondé *(en anglais* trim*).*

Les états utiles sont donc les états qui servent dans au moins un calcul réussi : on peut les atteindre depuis l'état initial, et les quitter pour un état final. Les autres états, les états inutiles, ne servent pas à grand-chose, en tout cas pas à la spécification de $L(A)$. C'est précisément ce que montre le théorème suivant.

Théorème 3.1 (Émondage). *Si $L(A) \neq \emptyset$ est un langage reconnaissable, alors il est également reconnu par un automate émondé.*

Preuve : Soit A un automate reconnaissant L, et $Q_u \subset Q$ l'ensemble de ses états utiles ; Q_u n'est pas vide dès lors que $L(A) \neq \emptyset$. La restriction δ' de δ à Q_u permet de définir un automate $A' = (\Sigma, Q_u, q_0, F, \delta')$. A' est équivalent à A. Q_u étant un sous-ensemble de Q, on a en effet immédiatement

$L(A') \subset L(A)$. Soit u dans $L(A)$, tous les états du calcul qui le reconnaît étant par définition utiles, ce calcul existe aussi dans A' et aboutit dans un état final : u est donc aussi reconnu par A'.

3.1.4 Automates non-déterministes

Dans cette section, nous augmentons le modèle d'automate défini en (3.4), en autorisant plusieurs transitions sortantes d'un état q à porter le même symbole : les automates ainsi spécifiés sont dits *non-déterministes*. Nous montrons que cette généralisation n'augmente toutefois pas l'expressivité du modèle : les langages reconnus par les automates non-déterministes sont les mêmes que ceux reconnus pas les automates déterministes.

Non-déterminisme

Définition 3.6 (Automate fini non-déterministe). *Un automate fini* non-déterministe *(NFA) est défini par un quintuplet $A = (\Sigma, Q, q_0, F, \delta)$, où :*
– *Σ est un ensemble fini de symboles (l'alphabet)*
– *Q est un ensemble fini d'états*
– *$q_0 \in Q$ est l'état* initial
– *$F \subset Q$ sont les états* finaux
– *δ est une fonction (partielle) de $(\Sigma \times Q)$ dans 2^Q : l'image par δ d'un couple (q, a) est un sous-ensemble de Q.*

La nouveauté introduite par cette définition est l'indétermination qui porte sur les transitions : pour une paire (q, a), il peut exister dans A plusieurs transitions possibles. On parle, dans ce cas, de *non-déterminisme*, signifiant qu'il existe des états dans lesquels la lecture d'un symbole a dans l'entrée provoque un choix (ou une indétermination) et que plusieurs transitions alternatives sont possibles. Notez que cette définition généralise proprement la notion d'automate fini : la définition (3.4) est un cas particulier de la définition (3.6), avec pour tout (q, a), l'ensemble $\delta(q, a)$ ne contient qu'un seul élément : on parle alors d'automate déterministe.

Les notions de calcul et de calcul réussi se définissent exactement comme dans dans le cas déterministe. On définit également la fonction de transition étendue $\delta^\star$ de $Q \times \Sigma^\star$ dans 2^Q par :
– $\delta^\star(q, u) = \delta(q, u)$ si $|u| = 1$
– $\delta^\star(q, au) = \bigcup_{r \in \delta(q,a)} \delta^\star(r, u)$
Ces notions sont illustrées sur l'automate non-déterministe de la Figure 3.6 : deux transitions sortantes de 1 sont étiquetées par a : $\delta(1, a) = \{1, 2\}$. aa donne lieu à un calcul réussi passant successivement par 1, 2 et 4, qui est final ; aa donne aussi lieu à un calcul $(1, 1, 1)$, qui n'est pas un calcul réussi.

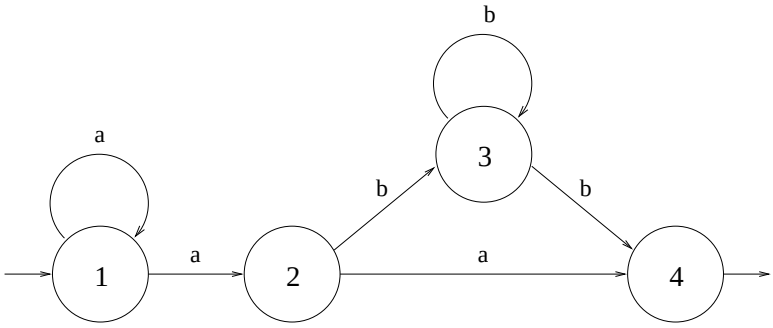

FIG. 3.6 – Un automate non-déterministe

Le langage reconnu par un automate non-déterministe est défini par :

$$L(A) = \{u \in \Sigma^\star, \delta^\star(q_0, u) \cap F \neq \emptyset\}$$

Pour qu'un mot appartienne au langage reconnu par l'automate, il suffit qu'il existe, parmi tous les calculs possibles, un calcul réussi, c'est-à-dire un calcul qui consomme tous les symboles de u entre q_0 et un état final ; la reconnaissance n'échoue donc que si *tous les calculs* aboutissent à une des situations d'échec. Ceci implique que pour calculer l'appartenance d'un mot à un langage, il faut examiner successivement tous les chemins possibles, et donc éventuellement revenir en arrière dans l'exploration des parcours de l'automate lorsque l'on rencontre une impasse. Dans ce nouveau modèle, le temps de reconnaissance d'un mot n'est plus linéaire, mais proportionnel au nombre de chemins dont ce mot est l'étiquette, qui peut être exponentiel en fonction de la taille de l'entrée.

Le non-déterminisme ne paye pas

La généralisation du modèle d'automate fini liée à l'introduction de transitions non-déterministes est, du point de vue des langages reconnus, sans effet : tout langage reconnu par un automate fini non-déterministe est aussi reconnu par un automate déterministe.

Théorème 3.2. *Pour tout NFA A défini sur Σ, il existe un DFA A' équivalent à A. Si A a n états, alors A' a au plus 2^n états.*

Preuve : on pose $A = (\Sigma, Q, q_0, F, \delta)$ et on considère A' défini par : $A' = (\Sigma, 2^Q, \{q_0\}, F', \delta')$ avec :
– $F' = \{G \subset Q, F \cap G \neq \emptyset\}$.
– $\delta'(G, a) = H$, avec $H = \cup_{q \in G} \delta(q, a)$
Les états de A' sont donc associés de manière biunivoque à des sous-ensembles de Q (il y en a un nombre fini) : l'état initial est le singleton $\{q_0\}$; chaque partie contenant un état final de A donne lieu à un état final de A' ; la transition sortante d'un sous-ensemble E, étiquetée par a, atteint l'ensemble de tous les états de Q atteignables depuis un état de E par une transition étiquetée par a. A' est le *déterminisé* de A. Illustrons cette construction sur l'automate de la Figure 3.7.

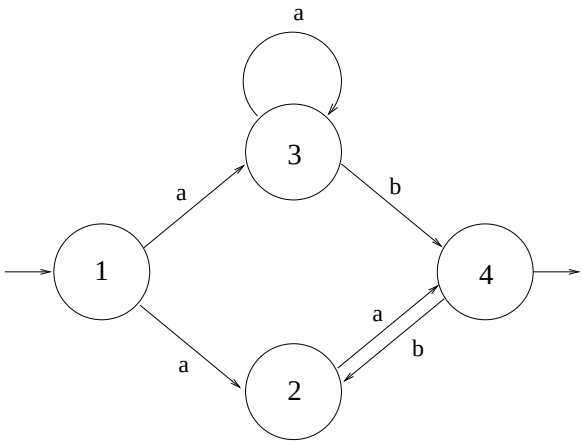

FIG. 3.7 – Un automate à déterminiser

L'automate de la Figure 3.7 ayant 4 états, son déterminisé en aura donc 16, correspondant au nombre de sous-ensembles de $\{1, 2, 3, 4\}$. Son état initial est le singleton $\{1\}$, et ses états finaux tous les sous-ensembles contenant 4 : il y en a exactement 8, qui sont : $\{4\}$, $\{1, 4\}$, $\{2, 4\}$, $\{3, 4\}$, $\{1, 2, 4\}$,

$\{1,3,4\}$, $\{2,3,4\}$,$\{1,2,3,4\}$. Considérons par exemple les transitions sortantes de l'état initial : 1 ayant deux transitions sortantes sur le symbole a, $\{1\}$ aura une transition depuis a vers l'état correspondant au doubleton $\{2,3\}$. Le déterminisé est représenté à la Figure 3.8. On notera que cette figure ne représente que les états *utiles* du déterminisé : ainsi $\{1,2\}$ n'est pas représenté, puisqu'il n'existe aucun moyen d'atteindre cet état.

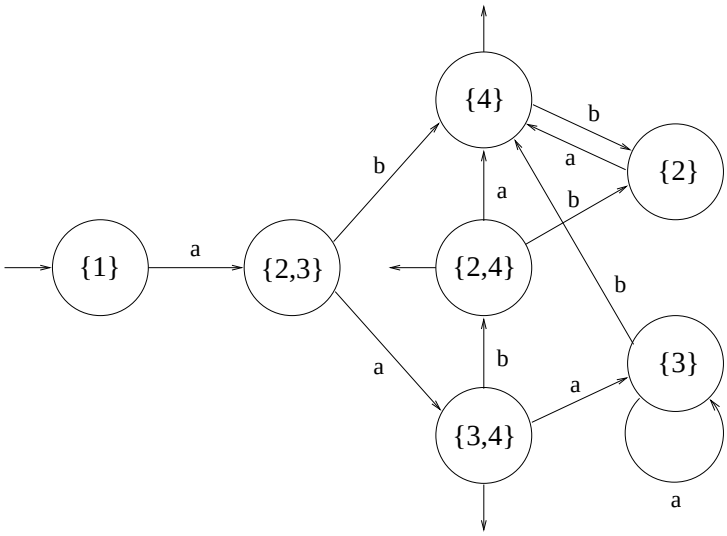

FIG. 3.8 – Le résultat de la déterminisation

Que se passerait-il si l'on ajoutait à l'automate de la Figure 3.7 une transition supplémentaire bouclant dans l'état 1 sur le symbole a ? Construisez le déterminisé de ce nouvel automate.

Démontrons maintenant le théorème 3.2 ; et pour saisir le sens de la démonstration, reportons nous à la Figure 3.7, et considérons les calculs des mots préfixés par aaa : le premier a conduit à une indétermination entre 2 et 3 ; suivant les cas, le second a conduit donc en 4 (si on a choisit d'aller initialement en 2) ou en 3 (si on a choisi d'aller initialement en 3). La lecture du troisième a lève l'ambiguïté, puisqu'il n'y a pas de transition sortante pour 4 : le seul état possible après aaa est 3. C'est ce qui se lit sur la Figure 3.8 : les ambiguïtés initiales correspondent aux états $\{2,3\}$ (atteint après le premier a) et $\{3,4\}$ (atteint après le second a) ; après le troisième le doute n'est plus permis et l'état atteint correspond au singleton $\{3\}$. Formalisons maintenant ces idées pour démontrer le résultat qui nous intéresse.

Première remarque : A' est un automate fini déterministe, puisque l'image par δ' d'un couple (H,a) est uniquement définie. Nous allons montrer que tout calcul dans A correspond à exactement un calcul dans A', soit formellement que :

$$\text{si } (q_0,u) \vdash_A^\star (p,\varepsilon) \text{ alors } \exists G, p \in G, (\{q_0\},u) \vdash_{A'}^\star (G,\varepsilon) \text{si } (\{q_0\},u) \vdash_{A'}^\star (G,\varepsilon) \text{ alors } \forall p \in G, (q_0\,u) \vdash_A^\star (G,\varepsilon)$$

Opérons une récurrence sur la longueur de u : si u est égal à εle résultat est vrai par définition de l'état initial dans A'. Supposons que le résultat est également vrai pour tout mot de longueur strictement inférieure à u, et considérons $u = va$. Soit $(q_0,va) \vdash_A^\star (p,a) \vdash_A (q,\varepsilon)$ un calcul dans A : par l'hypothèse de récurrence, il existe un calcul dans A' tel que : $(\{q_0\},v) \vdash_{A'}^\star (G,\varepsilon)$, avec $p \in G$. Ceci implique, en vertu de la définition même de δ', que q appartient à $H = \delta'(G,a)$, et donc que $(\{q_0\},u = va) \vdash_{A'}^\star (H,\varepsilon)$, avec $q \in H$. Inversement, soit $(\{q_0\},u = va) \vdash_{A'}^\star (G,a) \vdash_{A'} (H,\varepsilon)$: pour tout p dans G il existe un calcul un calcul dans A tel que $(q_0,v) \vdash_A^\star (p,\varepsilon)$. G ayant une transition étiquetée par a, il existe également dans G un état p tel que $\delta(p,a) = q$, avec $q \in H$, puis que $(q_0,u = va) \vdash_A^\star (q,\varepsilon)$, avec $q \in H$. On déduit alors que l'on a $(q_0,u = va) \vdash_A^\star (q,\varepsilon)$, avec $q \in F$ si

et seulement si $(\{q_0\}, u) \vdash_{A'}^{\star} (G, \varepsilon)$ avec $q \in G$, donc avec $F \cap G \neq \emptyset$, soit encore $G \in F'$. Il s'ensuit directement que $L(A) = L(A')$.

La construction utilisée pour construire le NFA équivalent à un DFA A s'appelle la *construction des sous-ensembles* : elle se traduit directement dans un algorithme permettant de construire le déterminisé d'un automate quelconque. On notera que cette construction peut s'organiser de telle façon à ne considérer *que les états réellement utiles* du déterminisé. Il suffit, pour cela, de construire de proche en proche depuis $\{q_0\}$, les états accessibles, résultant en général à des automates (complets) ayant moins que 2^n états.

Il existe toutefois des automates pour lesquels l'explosion combinatoire annoncée a lieu, comme celui qui est représenté à la Figure 3.9. Sauriez-vous expliquer d'où vient cette difficulté ? Quel est le langage reconnu par cet automate ?

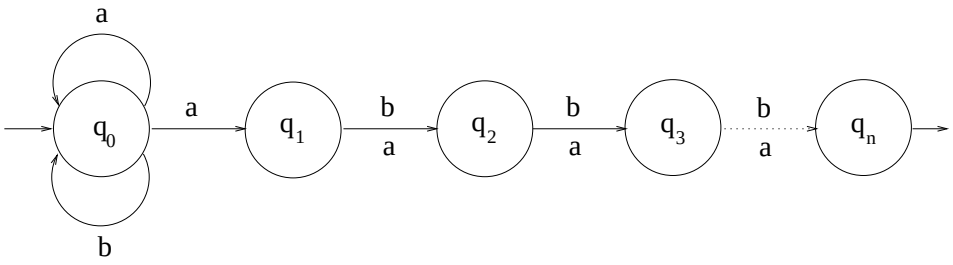

FIG. 3.9 – Un automate difficile à déterminiser

Dans la mesure où ils n'apportent aucun gain en expressivité, il est permis de se demander à quoi servent les automates non-déterministes ? Au moins à deux choses : ils sont plus faciles à construire à partir d'autres représentations des langages et sont donc utiles pour certaines preuves (voir plus loin) ou algorithmes. Ils fournissent également des machines bien plus (dans certains cas exponentiellement plus) 'compactes' que les DFA, ce qui n'est pas une propriété négligeable.

Transitions spontanées

Il est commode, dans la pratique, de disposer d'une définition encore plus plastique de la notion d'automate fini, en autorisant des transitions étiquetées par le mot vide, qui sont appelées les *transitions spontanées* .

Formellement, un automate non-déterministe avec transitions spontanées (en abrégé un ε-NFA) se définit comme un *NFA*, à la différence près que δ a maintenant comme domaine de définition $Q \times (\Sigma \cup \{\varepsilon\})$. Les transitions spontanées permettent d'étendre la notion de calcul : $(q, u) \vdash_A (p, v)$ si (i) $u = av$ et $p \in \delta(q, a)$ ou bien (ii) $u = v$ et $p \in \delta(q, \varepsilon)$. En d'autres termes, dans un ε-NFA, il est possible de changer d'état sans consommer de symbole, en empruntant une transition étiquetée par le mot vide. Le langage reconnu par un ε-NFA A est, comme précédemment, défini par $L(A) = \{u, (q_0, u) \vdash_A^{\star} (q, \varepsilon) \text{ avec } q \in F\}$.

La Figure 3.10 représente un exemple de ε-NFA, correspondant au langage $a^{\star}b^{\star}c^{\star}$.

Cette nouvelle extension n'ajoute rien de plus à l'expressivité du modèle, puisqu'il est possible de transformer chaque ε-NFA A en un NFA équivalent. Pour cela, nous introduisons tout d'abord la notion de ε-fermeture d'un état q, correspondant à tous les états accessibles depuis q par une ou plusieurs transition spontanée. Formellement :

Définition 3.7. *Soit q un état de Q. On appelle ε-fermeture (en anglais* closure*) de q l'ensemble $\varepsilon - closure(q) = \{p, (q, \varepsilon) \vdash_A^{\star} (p, \varepsilon)\}$. Par construction, $q \in \varepsilon - closure(q)$.*

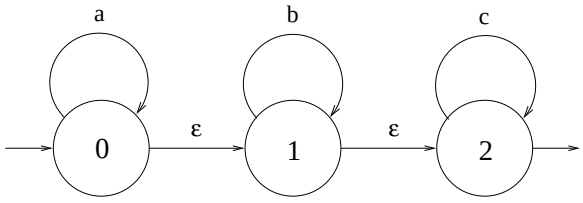

FIG. 3.10 – Un automate avec transitions spontanées

Intuitivement, la fermeture d'un état q contient tous les états qu'il est possible d'atteindre depuis q sans consommer de symboles. Ainsi, la fermeture de l'état 0 de l'automate de la Figure 3.10 est-elle égale à $\{0, 1, 2\}$.

Théorème 3.3. *Pour tout ε-NFA A, il existe un NFA A' tel que $L(A) = L(A')$.*

Preuve. En posant $A = (\Sigma, Q, q_0, F, \delta)$, on définit A' comme suit : $A' = (\Sigma, Q, q_0, F', \delta')$ avec :
– $F' = \{q, \varepsilon - closure(q) \cap F \neq \emptyset\}$
– $\delta'(q, a) = \bigcup_{p \in \varepsilon - closure(q)} \delta(p, a)$
Par une récurrence similaire à la précédente, on montre alors que tout calcul $(q_0, u) \vdash_A^\star p$ est équivalent à un calcul $(q_0, u) \vdash_{A'}^\star p'$, avec $p \in \varepsilon - closure(p')$, puis que $L(A) = L(A')$. On déduit directement un algorithme constructif pour supprimer, *à nombre d'états constant*, les transitions spontanées. Appliqué à l'automate de la Figure 3.10, cet algorithme construit l'automate représenté à la Figure 3.11.

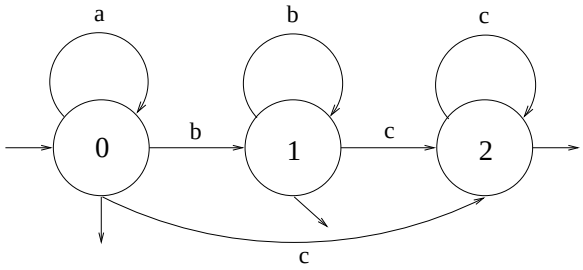

FIG. 3.11 – Un automate débarrassé de ses transitions spontanées

Les transitions spontanées introduisent une plasticité supplémentaire à l'objet automate. Par exemple, il est facile de voir que l'on peut, en les utilisant, transformer un automate fini quelconque en un automate équivalent doté d'un unique état final n'ayant que des transitions entrantes.

3.2 Reconnaissables

Nous avons défini à la section 3.1 les langages reconnaissables comme étant les langages reconnus par un automate fini déterministe. Les sections précédentes nous ont montré que nous aurions tout aussi bien les définir comme les langages reconnus par un automate fini non-déterministe ou encore par un automate fini non-déterministe avec transitions spontanées.

Dans cette section, nous montrons dans un premier temps que l'ensemble des langages reconnaissables est clos pour toutes les opérations «classiques», en produisant des constructions portant directement sur les automates. Nous montrons ensuite que les reconnaissables sont exactement les langages rationnels (présentés à la section 2.1.1), puis nous présentons un ensemble de résultats classiques permettant de caractériser les langages reconnaissables.

3.2.1 Opérations sur les reconnaissables

Théorème 3.4 (Clôture par complémentation). *Les langages reconnaissables sont clos par complémentation.*

Preuve : Soit L un reconnaissable et A un DFA complet reconnaissant L. On construit alors un automate A' pour $\overline{L}$ en prenant $A' = (\Sigma, Q, q_0, F', \delta)$, avec $F' = Q \backslash F$. Tout calcul réussi de A se termine dans un état de F, entraînant son échec dans A'. Inversement, tout calcul échouant dans A aboutit dans un état non-final de A, ce qui implique qu'il réussit dans A'.

Théorème 3.5 (Clôture par union ensembliste). *Les langages reconnaissables sont clos par union ensembliste.*

Preuve : Soit L^1 et L^2 deux langages reconnaissables, reconnus respectivement par A^1 et A^2, deux DFA complets. On construit un automate $A = (\Sigma, Q = Q^1 \times Q^2, q_0, F, \delta)$ pour $L_1 \cup L_2$ de la manière suivante :
– $q_0 = (q_0^1, q_0^2)$
– $F = (F^1 \times Q^2) \cup (Q^1 \times F^2)$
– $\delta((q_1, q_2), a) = (\delta^1(q^1, a), \delta^2(q^2, a))$
La construction de A est destinée à faire fonctionner A^1 et A^2 «en parallèle» : pour tout symbole d'entrée a, on transite par δ dans la paire d'états résultant d'une transition dans A^1 et d'une transition dans A^2. Un calcul réussi dans A est un calcul réussi dans A^1 (arrêt dans un état de $F^1 \times Q_2$) ou dans A^2 (arrêt dans un état de $Q^1 \times F_2$). Nous verrons un peu plus loin une autre construction, plus simple, pour cette opération, mais qui à l'inverse de la précédente, ne préserve pas le déterminisme de la machine réalisant l'union.

Théorème 3.6 (Clôture par intersection ensembliste). *Les langages reconnaissables sont clos par intersection ensembliste.*

Preuve (constructive[1]) : soient L^1 et L^2 deux reconnaissables, reconnus respectivement par A^1 et A^2, deux DFA complets. On construit un automate $A = (\Sigma, Q = Q^1 \times Q^2, q_0, F, \delta)$ pour $L_1 \cap L_2$ de la manière suivante :
– $q_0 = (q_0^1, q_0^2)$
– $F = (F^1, F^2)$
– $\delta((q_1, q_2), a) = (\delta^1(q^1, a), \delta^2(q^2, a))$
La construction de l'automate intersection est identique à celle de l'automate réalisant l'union, à la différence près qu'un calcul réussi dans A doit ici réussir simultanément dans les deux automates A^1 et A^2. Ceci s'exprime dans la nouvelle définition de l'ensemble F des états finaux comme : $F = (F^1, F^2)$.

Théorème 3.7 (Clôture par miroir). *Les langages reconnaissables sont clos par miroir.*

Preuve : Soit L un langage reconnaissable, reconnu par $A = (\Sigma, Q, q_0, F, \delta)$, et n'ayant qu'un unique état final, noté q_F. A', défini par $A' = (\Sigma, Q, q_F, \{q_0\}, \delta')$, où $\delta'(q, a) = p$ si et seulement si $\delta(p, a) = q$ reconnaît exactement le langage miroir de L. A' est en fait obtenu en inversant l'orientation des arcs de A : tout calcul de A énumérant les symboles de u entre q_0 et q_F correspond à un calcul de A' énumérant u^R entre q_F et q_0. On notera que même si A est déterministe, la construction ici proposée n'aboutit pas nécessairement à un automate déterministe pour le miroir.

[1]Une preuve plus directe utilise les deux résultats précédents et la loi de Morgan $L_1 \cap L_2 = \overline{(\overline{L_1} \cup \overline{L_2})}$.

Théorème 3.8 (Clôture par concaténation). *Les langages reconnaissables sont clos par concaténation*

Preuve : Soient L^1 et L^2 deux langages reconnus respectivement par A^1 et A^2, où l'on suppose que les ensembles d'états Q^1 et Q^2 sont disjoints et que A^1 a un unique état final de degré extérieur nul. On construit l'automate A pour L^1L^2 en identifiant l'état final de A^1 avec l´état initial de A^2. Formellement on a $A = (\Sigma, Q^1 \cup Q^2 \backslash \{q_0^2\}, q_0^1, F^2, \delta)$, où δ est défini par :
– $\forall q \in Q^1, q \neq q_f, a \in \Sigma, \delta(q,a) = \delta^1(q,a)$
– $\delta(q,a) = \delta^2(q,a)$ si $q \in Q^2$.
– $\delta(q_F^1, a) = \delta^1(q_F, a) \cup \delta^2(q_0^2, a)$
Tout calcul réussi dans A doit nécessairement atteindre un état final de A^2 et pour cela préalablement atteindre l´état final de A^1, seul point de passage vers les états de A^2. De surcroît, le calcul n'emprunte, après le premier passage dans q_F^1, que des états de A_2 : il se décompose donc en un calcul réussi dans chacun des automates. Réciproquement, un mot de L^1L^2 se factorise sous la forme $u = vw$, $v \in L^1$ et $w \in L^2$. Chaque facteur correspond à un calcul réussi respectivement dans A^1 et dans A^2, desquels se déduit immédiatement un calcul réussi dans A.

Théorème 3.9 (Clôture par étoile). *Les langages reconnaissables sont clos par étoile.*

Preuve : La construction de A' reconnaissant $L^\star$ à partir de A reconnaissant L est immédiate : il suffit de rajouter une transition spontanée depuis tout état final de A vers l'état initial q_0. Cette nouvelle transition permet l'itération dans A' de mots de L. Pour compléter la construction, on vérifie si εappartient à $L(A)$: si ce n'est pas le cas, alors il faudra marquer l'état initial de A comme état final de A'.

En application de cette section, vous pourrez montrer (en construisant les automates correspondants) que les langages reconnaissables sont aussi clos pour les opérations de préfixation, suffixation, pour les facteurs, les sous-mots...

3.2.2 Reconnaissables et rationnels

Les propriétés de clôture démontrées pour les reconnaissables (pour l'union, la concaténation et l´étoile) à la section précédente, complétées par la remarque que tous les langages finis sont reconnaissables, nous permettent d'affirmer que tout langage rationnel est reconnaissable. L'ensemble des langages rationnels étant en effet le plus petit ensemble contenant tous les ensembles finis et clos pour les opérations rationnelles, il est nécessairement inclus dans l'ensemble des reconnaissables. Nous montrons dans un premier temps comment exploiter les constructions précédentes pour construire simplement un automate correspondant à une expression rationnelle donnée. Nous montrons ensuite la réciproque, à savoir que tout reconnaissable est également rationnel : les langages reconnus par les automates finis sont précisément ceux qui sont décrits par des expressions rationnelles.

Des expressions rationnelles vers les automates

Les constructions de la section précédente ont montré comment construire les automates réalisant des opérations élémentaires sur les langages. Nous allons nous inspirer de ces constructions pour dériver un algorithme permettant de convertir une expression rationnelle en un automate fini reconnaissant le même langage.

Les expressions rationnelles sont formellement définies de manière récursive à partir des «briques»

de base que sont $\emptyset$, εet les symboles de Σ. Nous commençons donc par présenter les automates finis pour les langages dénotés par ces trois expressions rationnelles à la Figure 3.12.

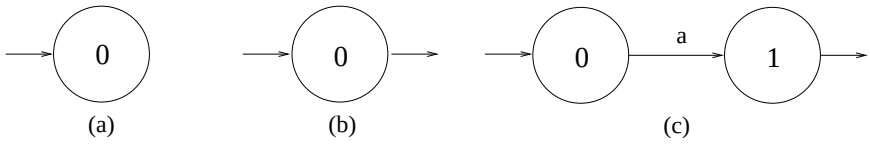

FIG. 3.12 – Machines élémentaires pour $\emptyset$, $\{\varepsilon\}$ et $\{a\}$

À partir de ces automates élémentaires, nous allons construire de manière itérative des automates pour des expressions rationnelles plus complexes. Pour cette construction, nous n'utiliserons que des automates qui ont un unique état final n'admettant aucune transition sortante, que nous noterons q_F. C'est bien le cas des automates élémentaires pour εet a ; pour $\emptyset$ il faudrait rajouter un état final distinct de (et non relié à) l'état initial. Si e_1 et e_2 dénotent les langages reconnus respectivement par A_1 et A_2, alors l'automate de la Figure 3.13 reconnaît le langage dénoté par l'expression $e_1 + e_2$.

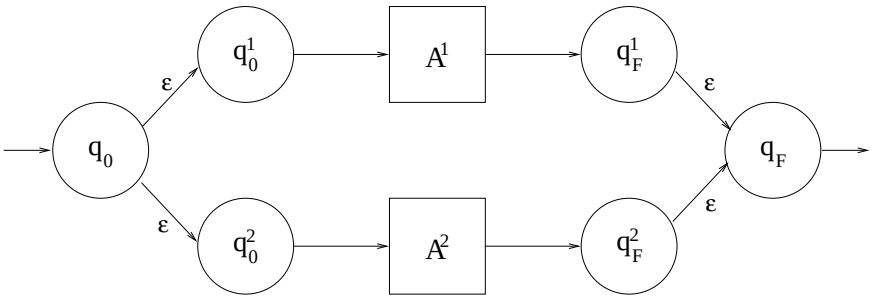

FIG. 3.13 – Machine réalisant $e_1 + e_2$

L'union correspond donc à une mise en parallèle «physique» de A_1 et de A_2 : un calcul dans cette machine est réussi si et seulement s'il est réussi dans l'une des deux machines A_1 ou A_2. On note, par ailleurs, que la machine résultant de l'union conserve la propriété de n'avoir qu'un seul état final de degré sortant nul.

La machine reconnaissant le langage dénoté par concaténation de deux expressions e_1 et e_2 correspond à une mise en série des deux machines A_1 et A_2, où l'état final de A_1 est connecté à l'état initial de A_2 par une transition spontanée comme sur la Figure 3.14.

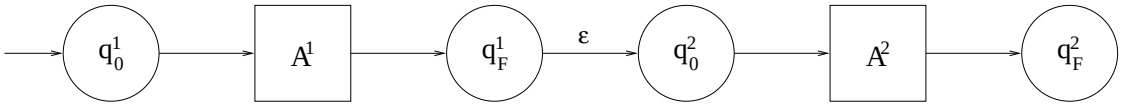

FIG. 3.14 – Machine réalisant e_1e_2.

La machine réalisant l'étoile est, comme précédemment, construite en rajoutant une possibilité de reboucler depuis l'état final vers l'état initial de la machine, ainsi qu'un arc permettant de reconnaître ε, comme représenté sur la Figure 3.15.

A partir de ces constructions simples, il est possible de dériver un algorithme permettant de construire un automate reconnaissant le langage dénoté par une expression régulière quelconque : il suffit de décomposer l'expression en ses composants élémentaires, puis d'appliquer les constructions précédentes pour construire l'automate correspondant. Cet algorithme est connu sous le nom

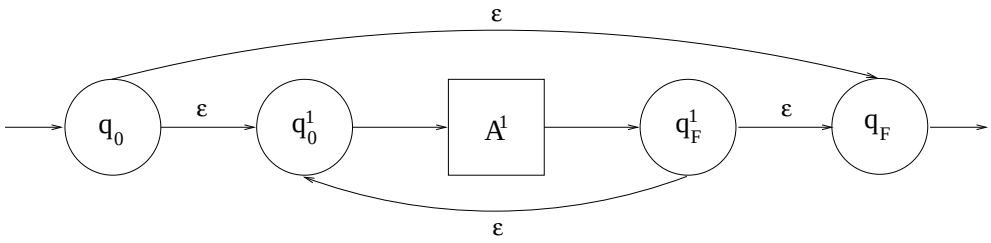

FIG. 3.15 – Machine réalisant $e_1^\star$

d'algorithme de Thompson.

En guise d'illustration de cette construction, la Figure 3.16 représente l'automate correspondant à l'expression : $e = (a + b)^\star b$.

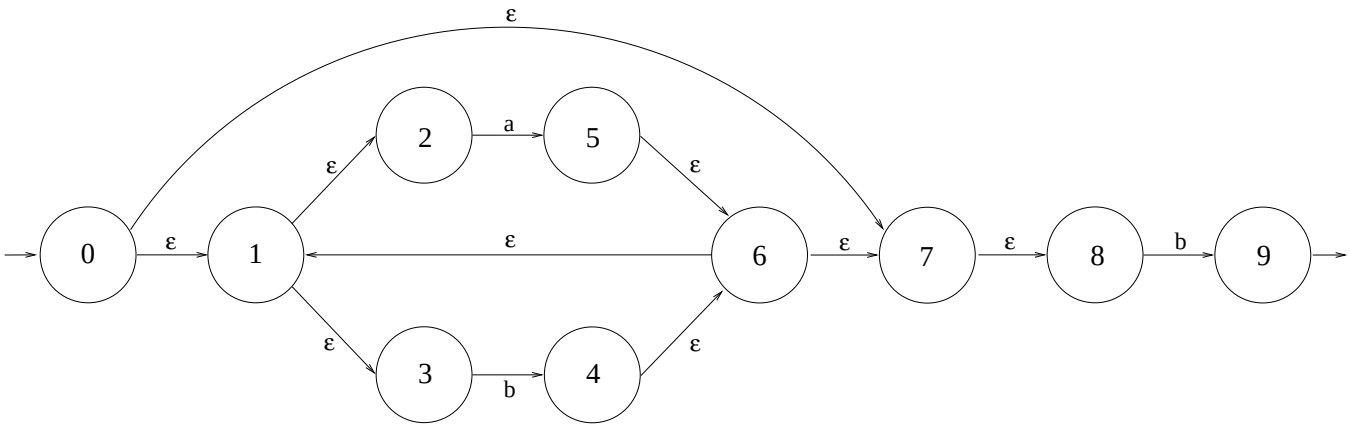

FIG. 3.16 – Machine pour $(a + b)^\star b$

Cette construction simple produit un automate qui a au plus $2n$ états pour une expression formée par n opérations rationnelles (car chaque opération rajoute exactement deux états) ; chaque état de l'automate possède au plus deux transitions sortantes. Cependant, cette construction a le désavantage de produire un automate non-déterministe, contenant de multiples transitions spontanées. Il existe d'autres procédures permettant de traiter de manière plus efficace les expressions ne contenant pas le symbole ε(par exemple la construction de *Gloushkow*) ou pour construire directement un automate déterministe.

Des automates vers les expressions rationnelles

La construction d'une expression rationnelle dénotant le langage reconnu par un automate A demande l'introduction d'une nouvelle variété d'automates, que nous appellerons *généralisés* . Les automates généralisés diffèrent des automates finis en ceci que leurs transitions sont étiquetées par des sous-ensembles rationnels de $\Sigma^\star$. Dans un automate généralisé, l'étiquette d'un calcul se construit par concaténation des étiquettes rencontrées le long des transitions ; le langage reconnu par un automate généralisé est l'union des langages correspondants aux calculs réussis. Les automates généralisés reconnaissent exactement les mêmes langages que les automates finis « standard ».

L'idée générale de la transformation que nous allons étudier consiste à partir d'un automate fini standard et de supprimer un par un les états, tout en s'assurant que cette suppression ne modifie

pas le langage reconnu par l'automate. Ceci revient à construire de proche en proche une série d'automates généralisés qui sont tous équivalents à l'automate de départ. La procédure se termine lorsqu'il ne reste plus que l'unique état initial et l'unique état final : en lisant l'étiquette des transitions correspondantes, on déduit une expression rationnelle dénotant le langage reconnu par l'automate originel.

Pour se simplifier la tâche commençons par introduire deux nouveaux états, q_I et q_F, qui joueront le rôle d'unique état respectivement initial et final. Ces nouveaux états sont connectés aux états initial et finaux par des transitions spontanées. On s'assure ainsi qu'à la fin de l'algorithme, il ne reste plus qu'une seule et unique transition, celle qui relie q_I à q_F.

L'opération cruciale de cette méthode est celle qui consiste à supprimer l'état q_i, où q_i n'est ni initial, ni final. On suppose qu'il y a au plus une transition entre deux états : si ça n'est pas le cas, il est possible de se ramener à cette configuration en prenant l'union des transitions existantes. On note e_{ii} l'étiquette de la transition de q_i vers q_i si celle ci existe ; si elle n'existe pas on a simplement $e_{ii} = \varepsilon$.

La procédure de suppression de q_i comprend alors les étapes suivantes :
- pour chaque paire d'état (q_j, q_k) avec $j \neq i, k \neq i$, tels qu'il existe une transition $q_j \to q_i$ étiquetée e_{ji} et une transition $q_i \to q_k$ étiquetée e_{ik}, ajouter la transition $q_j \to q_k$, portant l'étiquette $e_{ji}e_{ii}^{\star}e_{ik}$. Si la transition $q_j \to q_k$ existe déjà avec l'étiquette e_{jk}, alors il faut additionnellement faire : $e_{jk} = (e_{jk} + e_{ji}e_{ii}^{\star}e_{ik})$. Cette transformation doit être opérée pour chaque paire d'états (y compris pour $q_j = q_k$!) avant que q_i puisse être supprimé. Une illustration graphique de ce mécanisme est reproduit sur la Figure 3.17.
- supprimer q_i, ainsi que tous les arcs incidents

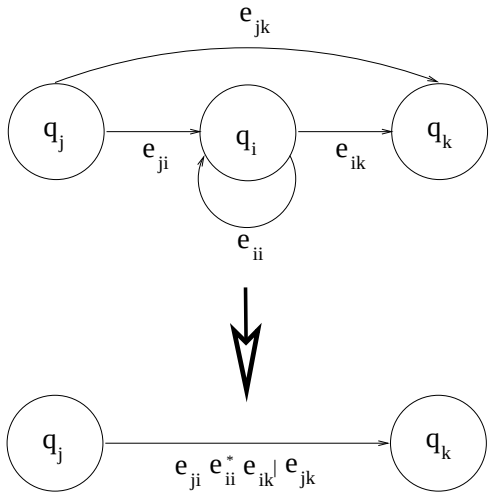

FIG. 3.17 – Illustration de BMC : élimination de l'état i

La preuve de la correction de cet algorithme réside en la vérification qu'à chaque itération le langage reconnu ne change pas. Cet invariant se vérifie très simplement : tout calcul réussi passant par q_i avant que q_i soit supprimé contient une séquence $q_j q_i q_k$. L'étiquette de ce sous-calcul étant copiée lors de la suppression de q_i sur l'arc $q_j \to q_k$, un calcul équivalent existe dans l'automate réduit.

Cette méthode de construction de l'expression équivalente à un automate est appelée *méthode d'élimination des états*, connue également sous le nom d'algorithme de Brzozowski et McCluskey. Vous noterez que le résultat obtenu dépend de l'ordre selon lequel les états sont examinés.

En guise d'application, il est recommandé de déterminer quelle est l'expression rationnelle corres-

pondant aux automates des figures 3.7 et 3.8.

Un corollaire fondamental de ce résultat est que pour chaque langage reconnaissable, il existe (au moins) une expression rationnelle qui le dénote : tout langage reconnaissable est rationnel.

Kleene

Nous avons montré comment construire un automate correspondant à une expression rationnelle et comment dériver une expression rationnelle dénotant un langage équivalent à celui reconnu par un automate fini quelconque. Ces deux résultats permettent d'énoncer un des résultats majeurs de ce chapitre.

Théorème 3.10 (Kleene). *Un langage est reconnaissable (reconnu par un automate fini) si et seulement si il est rationnel (dénoté par une expression rationnelle).*

Ce résultat permet de tirer quelques conséquences non encore envisagées : par exemple que les langages rationnels sont clos par complémentation et par intersection finie : ce résultat, loin d'être évident si on s'en tient aux notions d'opérations rationnelles, tombe simplement lorsque l'on utilise l'équivalence entre rationnels et reconnaissables.

De manière similaire, les méthodes de transformation d'une représentation à l'autre sont bien pratiques : s'il est immédiat de dériver de l'automate de la Figure 3.2 une expression rationnelle pour le langage contenant un nombre de a multiple de 3, il est beaucoup moins facile d'écrire directement l'expression rationnelle correspondante. En particulier, le passage par la représentation sous forme d'automates permet de déduire une méthode pour vérifier si deux expressions sont équivalentes : construire les deux automates correspondants, et vérifier qu'ils sont équivalents. Nous savons déjà comment réaliser la première partie de ce programme ; une procédure pour tester l'équivalence de deux automates est présentée à la section 3.3.2.

3.3 Quelques propriétés des langages reconnaissables

L'équivalence entre langages rationnels et langages reconnaissables a enrichi singulièrement notre palette d'outils concernant les langages rationnels : pour montrer qu'un langage est rationnel, on peut au choix exhiber un automate fini pour ce langage ou bien encore une expression rationnelle. Pour montrer qu'un langage *n'est pas rationnel*, il est utile de disposer de la propriété présentée dans la section 3.3.1, qui est connue sous le nom de *lemme de pompage* (en anglais *pumping lemma*). Intuitivement, cette propriété pose des limitations intrinsèques concernant la diversité des mots appartenant à un langage rationnel infini : au delà d'une certaine longueur, les mots d'un langage rationnel sont en fait construits par itération de motifs apparaissant dans des mots plus courts.

3.3.1 Lemme de pompage

Théorème 3.11 (Lemme de pompage). *Soit L un langage rationnel infini. Il existe un entier k tel que pour tout mot x de L plus long que k, x se factorise en $x = uvw$, avec (i) $|v| \geq 1$ (ii) $|uv| \leq k$ et (iii) pour tout $i \geq 0$, uv^iw est également un mot de L.*

Si un langage est infini, alors il contient des mots de longueur arbitrairement grande. Ce que dit ce lemme, c'est essentiellement que dès qu'un mot de L est assez long, il contient un facteur différent

de εqui peut être itéré à volonté tout en restant dans L. En d'autres termes, les mots « longs» de L sont construits par répétition d'un facteur s'insérant à l'intérieur de mots plus courts.

Preuve : Soit A un DFA de k états reconnaissant L et soit x un mot de L, de longueur supérieure ou égale à k. La reconnaissance de x dans A correspond à un calcul $q_0 \dots q_n$ impliquant $|x| + 1$ états. A n'ayant que k états, le préfixe de longueur $k+1$ de cette séquence contient nécessairement deux fois le même état q, aux indices i et j, avec $0 \leq i < j \leq k$. Si l'on note u le préfixe de x tel que $\delta^\star(q_0, u) = q$ et v le facteur tel que $\delta^\star(q, v) = q$, alors on a bien (i) car au moins un symbole est consommé le long du cycle $q \dots q$; (ii) car $j \leq k$; (iii) en court-circuitant ou en itérant les parcours le long du circuit $q \dots q$.

Attention : le lemme est aussi vérifié pour de nombreux langages non-rationnels : il n'exprime donc qu'une condition nécessaire (mais pas suffisante) de rationalité.

Ce lemme permet, par exemple, de prouver que le langage des carrés parfaits défini par $L = \{u \in \Sigma^\star, \exists v \text{ tel que } u = v^2\}$ n'est pas un langage rationnel. En effet, soit k l'entier spécifié par le lemme de pompage et x un mot plus grand que $2k$: $x = yy$ avec $|y| \geq k$. Il est alors possible d'écrire $x = uvw$, avec $|uv| \leq k$. Ceci implique que uv est un préfixe de y, et y un suffixe de w. Pourtant, uv^iw doit également être dans L, alors qu'un seul des y est affecté par l'itération : ceci est manifestement impossible.

Vous montrerez de même que le langage ne contenant que les mots dont la longueur est un carré parfait et que le langage des mots dont la longueur est un nombre premier ne sont pas non plus des langages reconnaissables.

Une manière simple d'exprimer cette limitation intrinsèque des langages rationnels se fonde sur l'observation suivante : dans un automate, le choix de l'état successeur pour un état q ne dépend que de q, et pas de la manière dont le calcul s'est déroulé avant q. En conséquence, *un automate fini ne peut gérer qu'un nombre fini de configurations différentes*, ou, dit autrement, *possède une mémoire bornée*. C'est insuffisant pour un langage tel que le langage des carrés parfaits pour lequel l'action à conduire (le langage à reconnaître) après un suffixe u dépend de u tout entier : reconnaître un tel langage demanderait en fait un nombre infini d'états.

3.3.2 Quelques conséquences

Dans cette section, nous établissons quelques résultats complémentaires portant sur la décidabilité, c'est-à-dire sur l'existence d'algorithmes permettant de résoudre quelques problèmes classiques portant sur les langages rationnels. Nous connaissons déjà un algorithme pour décider si un mot appartient à un langage rationnel (l'algorithme 1) ; cette section montre en fait que la plupart des problèmes classiques pour les langages rationnels ont des solutions algorithmiques.

Théorème 3.12. *Si A est un automate fini contenant k états :*
(i) $L(A)$ est non vide si et seulement si A reconnaît un mot de longueur strictement inférieure à k.
(ii) $L(A)$ est infini si et seulement si A reconnaît un mot u tel que $k \leq |u| < 2k$

Preuve (i) : un sens de l'implication est trivial : si A reconnaît un mot de longueur inférieure à k, $L(A)$ est non-vide. Supposons $L(A)$ non vide et soit u le plus petit mot de $L(A)$; supposons que la longueur de u soit strictement supérieure à k. Le calcul $(q_0, u) \vdash_A^\star (q, \varepsilon)$ contient au moins k étapes, impliquant qu'un état au moins est visité deux fois et est donc impliqué dans un circuit C. En court-circuitant C, on déduit un mot de $L(A)$ de longueur strictement inférieure à la longueur de u, ce qui contredit l'hypothèse de départ. On a donc bien $|u| < k$.

(ii) : un raisonnement analogue à celui utilisé pour montrer le lemme de pompage nous assure qu'un sens de l'implication est vrai. Si maintenant $L(A)$ est infini, il doit au moins contenir un mot plus long que k. Soit u le plus petit mot de longueur au moins k : soit il est de longueur strictement inférieure à $2k$, et le résultat est prouvé. Soit il est au moins égal à $2k$, mais par le lemme de pompage on peut court-circuiter un facteur de taille au plus k et donc exhiber un mot strictement plus court et de longueur au moins $2k$, ce qui est impossible. C'est donc que le plus petit mot de longueur au moins k a une longueur inférieure à $2k$.

Théorème 3.13. *Soit A un automate fini, il existe un algorithme permettant de décider si :*
– *$L(A)$ est vide*
– *$L(A)$ est fini / infini*

Ce résultat découle directement des précédents : il existe, en effet, un algorithme pour déterminer si un mot u est reconnu par A. Le résultat précédent nous assure qu'il suffit de tester $|\Sigma|^k$ mots pour décider si le langage d'un automate A est vide. De même, $|\Sigma|^{2k} - |\Sigma|^k$ vérifications suffisent pour prouver qu'un automate reconnaît un langage infini.

On en déduit un résultat concernant l'équivalence :

Théorème 3.14. *Soient A^1 et A^2 deux automates finis. Il existe une procédure permettant de décider si A^1 et A^2 sont équivalents.*

Il suffit en effet pour cela de former l'automate reconnaissant $(L(A^1) \cap \overline{L(A^2)}) \cup (\overline{L(A^1)} \cap L(A^2))$ (par exemple en utilisant les procédures décrites à la section 3.2.1) et de tester si le langage reconnu par cet automate est vide. Si c'est le cas, alors les deux automates sont effectivement équivalents.

3.4 L'automate canonique

Dans cette section, nous donnons une nouvelle caractérisation des langages reconnaissables, à partir de laquelle nous introduisons la notion *d'automate canonique d'un langage*. Nous présentons ensuite un algorithme pour construire l'automate canonique d'un langage reconnaissable représenté par un DFA quelconque.

3.4.1 Une nouvelle caractérisation des reconnaissables

Commençons par une nouvelle définition : celle d'indistinguabilité.

Définition 3.8. *Soit L un langage de $\Sigma^\star$. Deux mots de u et v sont dits* indistinguables *dans L si pour tout w dans $\Sigma^\star$, soit uw et vw sont tous deux dans L, soit uw et vw sont tous deux dans $\overline{L}$.*

En d'autres termes, deux mots u et v sont distinguables dans L s'il existe un mot w de L tel que uw soit dans L, mais pas vw. La relation d'indistinguabilité dans L est une relation réflexive, symétrique et transitive : c'est une relation d'équivalence que nous noterons $\equiv_L$.

Considérons, à titre d'illustration, le langage $L = a(a+b)(bb)^\star$. Pour ce langage, $u = aab$ et $v = abb$ sont indistinguables : pour tout mot de L $x = uw$, ayant pour préfixe u, $y = vw$ est en effet un autre mot de L. $u = a$ et $v = aa$ sont par contre distinguables : en concaténant $w = abb$ à u, on obtient $aabb$ qui est dans L ; en revanche, $aaabb$ n'est pas un mot de L.

De manière similaire, on définit la notion d'indistinguabilité dans un automate déterministe A par :

Définition 3.9. *Soit $A = (\Sigma, Q, q_0, F, \delta)$ un automate fini déterministe. Deux mots de u et v sont dits* indistinguables *dans A si et seulement si $\delta^\star(q_0, u) = \delta^\star(q_0, v)$. On notera d'indistinguabilité dans A par : $u \equiv_A v$.*

Autrement dit, deux mots u et v sont indistinguables pour A si le calcul par A de u depuis q_0 aboutit dans le même état q que le calcul de v. Cette notion rejoint bien la précédente, puisque tout mot w tel que $\delta^\star(q, w)$ aboutisse à dans un état final est une continuation valide à la fois de u et de v dans L ; inversement tout mot conduisant à un échec depuis q est une continuation invalide à la fois de u et de v.

Pour continuer, rappelons la notion de congruence droite, déjà introduite à la section 1.4.2 :

Définition 3.10 (Invariance droite). *Une relation d´équivalence $\mathcal{R}$ sur $\Sigma^\star$ est dite* invariante à droite *si et seulement si : $u\mathcal{R}v \Rightarrow \forall w, uw\mathcal{R}vw$. Une relation invariante à droite est appelée une congruence droite.*

Par définition, les deux relations d'indistinguabilité définies ci-dessus sont invariantes à droite.

Nous sommes maintenant en mesure d'exposer le résultat principal de cette section.

Théorème 3.15 (Myhill-Nerode). *Soit L un langage sur Σ, les trois assertions suivantes sont équivalentes :*
- *(i) L est un langage rationnel*
- *(ii) il existe une relation d´équivalence $\equiv$ sur $\Sigma^\star$, invariante à droite, ayant un nombre fini de classes d'équivalence et telle que L est égal à l'union de classes d'équivalence de $\equiv$*
- *(iii) $\equiv_L$ possède un nombre fini de classes d'équivalences*

Preuve : (i) $\Rightarrow$ (ii) : A étant rationnel, il existe un DFA A qui le reconnaît. La relation d'équivalence $\equiv_A$ ayant autant de classes d'équivalences qu'il y a d'états, ce nombre est nécessairement fini. Cette relation est bien invariante à droite, et L, défini comme $\{u \in \Sigma^\star, \delta^\star(q_0, u) \in F\}$ est simplement l'union des classes d'équivalences associées aux états finaux de A.

(ii) $\Rightarrow$ (iii) : Soit $\equiv$ la relation satisfaisant la propriété (*ii*), et u et v tels que $u \equiv v$. Par la propriété d'invariance droite, on a pour tout mot w dans $\Sigma^\star$ $uw \equiv vw$. Ceci entraîne que soit uw et vw sont simultanément dans L (si leur classe d'équivalence commune est un sous-ensemble de L), soit tout deux hors de L (dans le cas contraire). Il s'ensuit que $u \equiv_L v$: toute classe d'équivalence pour $\equiv$ est incluse dans une classe d'équivalence de $\equiv_L$; il y a donc moins de classes d'équivalence pour $\equiv_L$ que pour $\equiv$, ce qui entraîne que le nombre de classes d'équivalence de $\equiv_L$ est fini.

(iii) $\Rightarrow$ (i) : construisons l'automate $A = (\Sigma, Q, q_0, F, \delta)$ suivant :
- chaque état de Q correspond à une classe d'équivalence $[u]_L$ de $\equiv_L$; d'où il s'ensuit que Q est fini.
- $q_0 = [\varepsilon]_L$, classe d'équivalence de ε
- $F = \{[u]_L, u \in L\}$
- $\delta([u]_L, a) = [ua]_L$. Cette définition de δ est indépendante du choix d'un représentant de $[u]_L$: si u et v sont dans la même classe pour $\equiv_L$, par invariance droite de $\equiv_L$, il en ira de même pour ua et va.

A ainsi défini est un automate fini déterministe et complet. Montrons maintenant que A reconnaît L et pour cela, montrons par induction que $(q_0, u) \vdash_A^\star [u]_L$. Cette propriété est vraie pour $u = \varepsilon$, supposons la vraie pour tout mot de taille inférieure à k et soit $u = va$ de taille $k + 1$. On a $(q_0, ua) \vdash_A^\star (p, a) \vdash_A (q, \varepsilon)$. Or, par l'hypothèse de récurrence on sait que $p = [u]_L$; et comme $q = \delta([u]_L, a)$, alors $q = [ua]_L$, ce qui est bien le résultat recherché. On déduit que si u est dans $L(A)$, un calcul sur l'entrée u aboutit dans un état final de A, et donc que u est dans L. Réciproquement, si u est dans L, un calcul sur l'entrée u aboutit dans un état $[u]_L$, qui est, par définition de A, final.

Ce résultat fournit une nouvelle caractérisation des langages reconnaissables et peut donc être utilisé pour montrer qu'un langage est, ou n'est pas, reconnaissable. Ainsi, par exemple, $L = \{u, \exists a \in \Sigma, i \in N \text{ tq. } u = a^{2^i}\}$ n'est pas reconnaissable. En effet, pour tout i, j, a^{2^i} et a^{2^j} sont distingués par a^{2^i}. Il n'y a donc pas un nombre fini de classes d'équivalence pour $\equiv_L$, et L ne peut en conséquence être reconnu par un automate fini. L n'est donc pas un langage rationnel.

3.4.2 Automate canonique

La principale conséquence du résultat précédent concerne l'existence d'un représentant unique (à une renumérotation des états près) et minimal (en nombre d'états) pour les classes de la relation d'équivalence sur les automates finis.

Théorème 3.16. *L'automate A_L, fondé sur la relation d'équivalence $\equiv_L$, est le plus petit automate déterministe complet reconnaissant L. Cet automate est unique (à une renumérotation des états près) et appelé automate* canonique *de L.*

Preuve : soit A un automate fini déterministe reconnaissant L. $\equiv_A$ définit une relation d´équivalence satisfaisant les conditions de l'alinéa (ii) de la preuve du théorème 3.15 présenté ci-dessus. Nous avons montré que chaque état de A correspond à une classe d'équivalence (pour $\equiv_A$) incluse dans une classe d'équivalence pour $\equiv_L$. Le nombre d'états de A est donc nécessairement plus grand que celui de A_L. Le cas où A et A_L ont le même nombre d'états correspond au cas où les classes d'équivalences sont toutes semblables, permettant de définir une correspondance biunivoque entre les états des deux machines.

L'existence de A_L étant garantie, reste à savoir comment le construire : la construction directe des classes d'équivalence de $\equiv_L$ n'est en effet pas nécessairement immédiate. Nous allons présenter un algorithme permettant de construire A_L à partir d'un automate déterministe quelconque reconnaissant L. Comme préalable, nous définissons une troisième relation d'indistinguabilité, portant cette fois sur les états :

Définition 3.11. *Deux états q et p d'un automate fini déterministe A sont* distinguables *s'il existe un mot w tel que le calcul (q, w) termine dans un état final alors que le calcul (p, w) échoue. Si deux états ne sont pas distinguables, ils sont* indistinguables.

Comme les relations d'indistinguabilité précédentes, cette relation est une relation d'équivalence, notée $\equiv_\nu$ sur les états de Q. L'ensemble des classes d'équivalence $[q]_\nu$ est notée Q_ν. Pour un automate fini déterministe $A = (\Sigma, Q, q_0, F, \delta)$, on définit l'automate fini A_ν par : $A_\nu = (\Sigma, Q_\nu, [q_0]_\nu, F_\nu, \delta_\nu)$, avec : $\delta_\nu([q]_\nu, a) = [\delta(q, a)]_\nu$; et $F_\nu = [q]_\nu$, avec q dans F. δ_ν est correctement défini en ce sens que si p et q sont indistinguables, alors nécessairement $\delta(q, a) \equiv_\nu \delta(p, a)$.

Montrons, dans un premier temps, que ce nouvel automate est bien identique à l'automate canonique A_L. On définit pour cela une application ϕ, qui associe un état de A_ν à un état de A_L de la manière suivante :

$$\phi([q]_\nu) = [u]_L \text{ s'il existe } u \text{ tel que } \delta^\star(q_0, u) = q$$

Notons d'abord que ϕ est une application : si u et v de $\Sigma^\star$ aboutissent tous deux dans des états indistinguables de A, alors il est clair que u et v sont également indistinguables, et sont donc dans la même classe d'équivalence pour $\equiv_L$: le résultat de ϕ ne dépend pas d'un choix particulier de u.

Montrons maintenant que ϕ est une bijection. Ceci se déduit de la suite d'équivalences suivante :

$$\phi([q]_\nu) = \phi([p]_\nu) \quad \Leftrightarrow \quad \exists u, v \in \Sigma^\star, \delta^\star(q_0, u) = q, \delta^\star(q_0, u) = p, \text{ et } u \equiv_L v \tag{3.1}$$

$$\Leftrightarrow \quad \delta^\star(q_0, u) \equiv_\nu \delta^\star(q_0, u) \tag{3.2}$$

$$\Leftrightarrow \quad [q]_\nu = [p]_\nu \tag{3.3}$$

Montrons enfin que les calculs dans A_ν sont en bijection par ϕ avec les calculs de A_L. On a en effet :
- $\phi([q_0]_\nu) = [\varepsilon]_L$, car $\delta^\star(q_0, \varepsilon) = q_0 \in [q_0]_\nu$
- $\phi(\delta_\nu([q]_\nu, a) = \delta_L(\phi([q]_\nu), a)$ car soit u tel que $\delta^\star(q_0, u) \in [q]_\nu$, alors : (i) $\delta(\delta^\star(q_0, u), a) \in \delta_\nu([q]_\nu, a)$ (cf. la définition de δ_ν) et $[ua]_L = \phi(\delta_\nu([q]_\nu, a)) = \delta_L([u], a)$ (cf. la définition de δ_L), ce qu'il fallait prouver.
- si $[q]_\nu$ est final dans A_ν, alors il existe u tel que $\delta^\star(q_0, u)$ soit un état final de A, impliquant que u est un mot de L, et donc que $[u]_L$ est un état final de l'automate canonique.

Il s'ensuit que chaque calcul dans A_ν est isomorphe (par ϕ) à un calcul dans A_L, puis que, ces deux automates ayant les mêmes états initiaux et finaux, ils reconnaissent le même langage.

L'idée de l'algorithme de minimisation de $A = (\Sigma, Q, q_0, F, \delta)$ consiste alors à chercher à identifier les classes d'équivalence pour $\equiv_\nu$, de manière à dériver l'automate A_ν (alias A_L). La finitude de Q nous garantit l'existence d'un algorithme pour calculer ces classes d'équivalences. La procédure itérative décrite ci-dessous esquisse une implantation naïve de cet algorithme, qui construit la partition correspondant aux classes d'équivalence par raffinement d'une partition initiale Π_0 qui distingue simplement états finaux et non-finaux. Cet algorithme se glose comme suit :
- Initialiser avec deux classes d'équivalence : F et $Q \backslash F$
- Itérer jusqu'à stabilisation :
  - pour toute paire d'état q et p dans la même classe de la partition Π_k, s'il existe $a \in \Sigma$ tel que $\delta(q, a)$ et $\delta(p, a)$ ne sont pas dans la même classe pour Π_k, alors ils sont dans deux classes différentes de Π_{k+1}.

On vérifie que lorsque cette procédure s'arrête (après un nombre fini d'étapes), deux états sont dans la même classe si et seulement si ils sont indistinguables. Cette procédure est connue sous le nom *d'algorithme de Moore*. Implantée de manière brutale, elle aboutit à une complexité quadratique (à cause de l'étape de comparaison de toutes les paires d'états). En utilisant des structures auxiliaires, il est toutefois possible de se ramener à une complexité en $n \log(n)$, avec n le nombre d'états.

Considérons pour illustrer cette procédure l'automate reproduit à la Figure 3.18 :

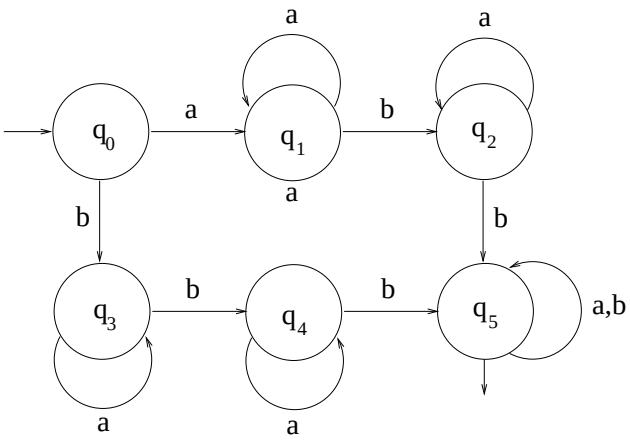

FIG. 3.18 – Un DFA à minimiser

Les itérations successives de l'algorithme de construction des classes d'équivalence pour $\equiv_\nu$ se déroulent alors comme suit :

– $\Pi_0 = \{\{q_0, q_1, q_2, q_3, q_4\}, \{q_5\}\}$ (car q_5 est le seul état final)
– $\Pi_1 = \{\{q_0, q_1, q_3\}, \{q_2, q_4\}, \{q_5\}\}$ (car q_2 et q_4, sur le symbole b, atteignent q_5).
– $\Pi_2 = \{\{q_0\}, \{q_1, q_3\}, \{q_2, q_4\}, \{q_5\}\}$ (car q_1 et q_3, sur le symbole b, atteignent respectivement q_2 et q_4).
– $\Pi_3 = \Pi_2$ fin de la procédure.
L'automate minimal résultant de ce calcul est reproduit à la Figure 3.19.

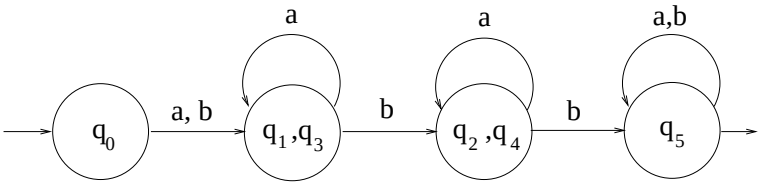

FIG. 3.19 – L'automate minimal de $(a + b)a^\star ba^\star b(a + b)^\star$