Automates finis

Jean-Éric Pin

Mots-clés : automate, langage, expression rationnelle, reconnaissable, automate séquentiel, industrie de la langue, vérification, spécification.

Résumé : introduits vers 1950, les automates finis constituent le modèle le plus élémentaire de machine. Ce chapitre présente d'une part les automates usuels, qui se contentent de lire un mot en entrée pour l'accepter ou le rejeter, et les automates séquentiels, munis d'une entrée et d'une sortie. Après une brève présentation du théorème de Kleene, clé de voûte de la théorie des automates, nous décrivons les applications des automates dans divers domaines, notamment la modélisation et les industries de la langue.

1 Un bref historique

La théorie des automates est née de la convergence de plusieurs courants scientifiques. Le premier est issu des tentatives de logiciens tels que Church, Gödel ou Turing pour formaliser la notion de *calcul* et de *machine*. Cet effort a occupé toute la première moitié du vingtième siècle et pourtant, les automates finis, qui constituent le modèle le plus simple de machine, ne seront définis formellement que bien après les machines de Turing. Les *systèmes dynamiques discrets* forment la seconde source d'inspiration. Bien que leur étude remonte aux travaux de Morse datant de la première moitié du vingtième siècle, leurs liens avec les automates finis font encore à ce jour l'objet de recherches très actives. Le troisième courant, proche cousin du précédent, est la *théorie de l'information* bâtie par Shannon en 1948. Les problèmes de codage, étudiés notamment par Schützenberger dès les années cinquante, ont en effet profondément influencé la théorie des automates. La quatrième source provient de linguistes tels que Chomsky qui, en cherchant à formaliser les langues naturelles, ont introduit les concepts de *mots*, *langages*, *grammaires*, que nous utilisons aujourd'hui. Le domaine des circuits électroniques a été la cinquième source d'inspiration. Il a conduit notamment à la notion d'automate avec sortie et au concept de spécification du comportement d'un circuit.

Les cinq domaines que nous venons d'évoquer brièvement ont eu une influence considérable sur la génèse et le développement de la théorie des automates. Mais historiquement, c'est un article sur les réseaux neuronaux publié en 1943 par McCulloch et Pitts qui est à l'origine de la notion d'automate. Il semble en effet que le terme « théorie des automates » ait été introduit en 1948 par Von Neumann en référence à cet article. Par ailleurs, à la demande de la RAND Corporation, Kleene a longuement analysé cet article dans un mémoire rédigé durant l'été 1951, mais publié seulement en 1956. Cet article marque la naissance de la théorie des automates.

Kleene y démontre un théorème qui affirme que les langages reconnus par un automate sont exactement les langages *rationnels*, appelés aussi langages *réguliers*, que l'on peut décrire à partir des lettres de l'alphabet eu utilisant trois opérations : l'union (qui joue le rôle de l'addition), le produit et l'étoile. On obtient ainsi un procédé descriptif tout à fait différent des automates, et ce résultat est assez surprenant. Il a pour conséquence que l'ensemble des langages rationnels est fermé par intersection et complément.

Les automates avec sortie ont été introduits eux aussi à la fin des années cinquante. Un automate avec sortie lit un mot en entrée et produit un mot en sortie. Les modèles les plus intéressants sont les automates séquentiels, qui permettent d'obtenir la sortie au fur et à mesure de la lecture du mot d'entrée. Ces automates sont très proches des circuits électroniques et leur synthèse en circuit peut d'ailleurs être entièrement automatisée.

2 Mots et langages

En informatique, mais aussi en mathématiques, en linguistique ou en biologie, les informations sont souvent représentées par des chaînes de caractères. Par exemple pour les données informatiques, on utilise des suites de 0 et de 1, pour l'information génétique, des suites formées des quatre caractères A (Adénine), C (Cytosine), G (Guanine) et T (Thymine) et pour les langues naturelles, les mots figurant dans un dictionnaire.

La formalisation commune à ces exemples est la suivante. Un *alphabet* est un ensemble fini dont les éléments sont appelés des *lettres*. Ainsi, on parle de l'alphabet binaire $\{0, 1\}$, de l'alphabet du génome $\{A, C, G, T\}$, de l'alphabet latin usuel $\{a, \dots, z, A, \dots, Z\}$. Dans les exemples qui vont suivre, on utilisera le plus souvent des alphabets assez petits tels que $\{a, b\}$ ou $\{a, b, c\}$ et on notera A l'alphabet tout entier.

Un *mot* sur l'alphabet A est une suite finie de lettres de A. On note ces lettres par simple juxtaposition : ainsi le mot *abracadabra* est un mot sur l'alphabet $\{a, b, c, d, r\}$. La longueur d'un mot u, notée $|u|$, est égale au nombre de lettres figurant dans u, chaque lettre étant comptée autant de fois qu'elle apparaît. Ainsi, $|abaabb| = 6$ et $|abracadabra| = 11$. Il existe aussi un mot de longueur 0, que l'on note ε.

Le produit (ou concaténation) de deux mots est le mot obtenu en mettant ces mots bout à bout. Par exemple, le produit des mots *abra* et *cadabra* est le mot *abracadabra*. On note aussi u^n le produit de n mots égaux à u. Ainsi, $(ab)^3 = ababab$.

On appelle *langage* tout ensemble de mots sur un alphabet donné. Par exemple, les ensembles $\{aba, babaa, bb\}$ et $\{a^n b^n \mid n \geqslant 0\}$ sont des langages sur l'alphabet $\{a, b\}$.

3 Automates déterministes

La figure 1.1 représente un *automate déterministe*. Cet automate possède trois états, 1, 2 et 3, entourés par des cercles sur la figure. L'état 1 est l'état *initial*, ce qu'on indique par une flèche entrante. Les états 2 et 3 sont des états *finaux*, ce qui est indiqué par une flèche sortante. Cet automate possède aussi des *transitions*, représentées par des flèches étiquetées par des lettres. De plus, de chaque état sort au plus une flèche d'étiquette donnée.

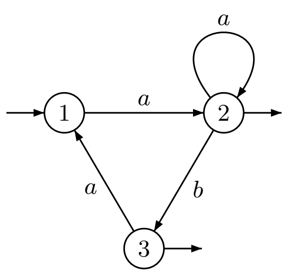

FIG. 1.1 – Un automate

Pour savoir si un mot est *accepté* ou non par l'automate, on lit de gauche à droite les lettres de ce mot en partant de l'état initial et en suivant les transitions. Si on parvient dans un état final, ce mot est accepté, sinon, il est rejeté.

Prenons par exemple le mot *aab*. En lisant ce mot en partant de l'état initial 1, on utilise d'abord la transition d'étiquette a qui va de 1 vers 2, puis celle de 2 vers 2 d'étiquette a et enfin celle d'étiquette b de 2 vers 3. Comme 3 est un état final, le mot *aab* est accepté par l'automate.

Si on lit maintenant le mot *aba* en partant de l'état initial 1, on utilise successivement les transitions de 1 vers 2, de 2 vers 3 et enfin de 3 vers 1. À l'issue de la lecture du mot *aba* on arrive donc dans l'état 1, qui n'est pas un état final : le mot est rejeté par l'automate.

Enfin, examinons le mot *abb*. Comme pour le mot *aba*, on commence par utiliser les transitions de 1 vers 2, puis de 2 vers 3, mais arrivé dans l'état 3, on constate qu'il n'y a pas de transition d'étiquette b issue de 3. La lecture du mot ne peut donc se poursuivre et le mot est également rejeté.

4 Langages reconnaissables

L'ensemble des mots acceptés par un automate est par définition le *langage reconnu* par l'automate. On dit qu'un langage est *reconnaissable* s'il existe un automate déterministe qui le reconnaît. En voici deux exemples.

L'automate de la figure 1.2 reconnaît le langage des mots sur l'alphabet $\{a, b\}$ qui commencent par *aba*.

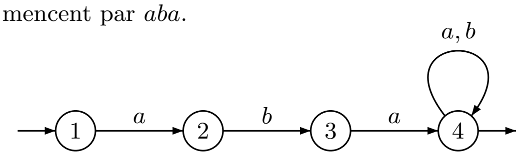

FIG. 1.2 – Un automate déterministe

Notre second exemple est un peu plus sophistiqué. Rappelons que la représentation binaire d'un entier s'obtient en le décomposant comme somme de puissances de 2. Par exemple, le nombre 330 qui vaut $2^8 + 2^6 + 2^3 + 2^1$, s'écrit 101001010 en base 2. On peut montrer que l'automate de la figure 1.3 reconnaît l'ensemble des mots sur l'alphabet $\{0, 1\}$ qui sont la représentation binaire d'un multiple de 3. Par exemple, 330 est divisible par 3 et le mot 101001010 est bien accepté par l'automate.

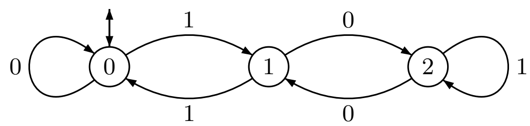

FIG. 1.3 – Multiples de 3

Tous les langages ne sont pas reconnaissables. Par exemple, on montre que le langage $\{0^n 1^n \mid n \geqslant 0\}$ n'est pas reconnaissable. La signification intuitive de ce résultat est qu'un automate fini ne peut simuler une pile de hauteur non bornée (voir chapitre £Modèles de machines£).

5 Automates non déterministes

Les automates *non déterministes* constituent une variante importante des automates déterministes. Ils sont eux aussi représentés par un graphe étiqueté, comme celui de la figure 1.4.

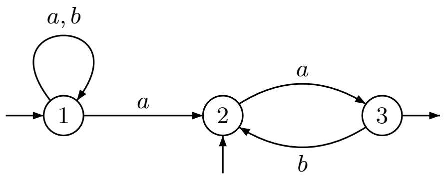

FIG. 1.4 – Un automate non déterministe

Mais contrairement au cas des automates déterministes, on peut avoir plusieurs états initiaux et plusieurs flèches de même étiquette issues du même état. Ainsi sur notre exemple, il y a deux états initiaux, 1 et 2, et deux transitions d'étiquette a issues de l'état 1 : une de 1 vers 1 et l'autre de 1 vers 2.

Un *chemin* est une suite de transitions consécutives. Le mot formé par les étiquettes de ces transitions consécutives est l'*étiquette* du chemin. Un chemin est *réussi* s'il part d'un état initial et aboutit dans un état final. Un mot est *accepté* s'il est l'étiquette d'au moins un chemin réussi (ce qui n'exclut pas qu'il puisse être simultanément l'étiquette d'un chemin non réussi !). Le *langage reconnu* par l'automate $\mathcal{A}$ est l'ensemble des mots acceptés par $\mathcal{A}$.

Les automates non déterministes permettent une représentation plus concise de certains langages. Par exemple, l'automate non déterministe de la figure 1.5 accepte l'ensemble des mots sur l'alphabet $\{0, 1\}$ contenant la chaîne 010.

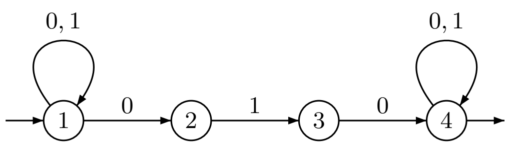

FIG. 1.5 – Un automate non déterministe pour les mots contenant la chaîne 010

Deux automates sont dits *équivalents* s'ils reconnaissent le même langage. On démontre que tout automate non déterministe est équivalent à un automate déterministe, comme illustré sur la figure 1.6.

Toutefois, déterminiser un automate peut avoir un coût prohibitif puisqu'on connaît des exemples d'automates non déterministes à n états dont le plus petit équivalent déterministe possède 2^n états. Fort heureusement, cette situation extrême se rencontre rarement en pratique.

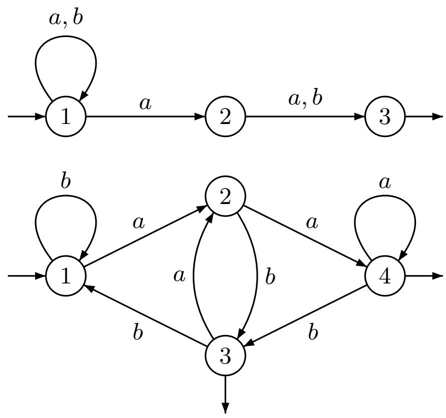

FIG. 1.6 – Un automate non déterministe et un automate déterministe équivalent

6 Automate minimal

Si on dispose d'un automate déterministe reconnaissant un langage L, on peut éliminer tous les états inaccessibles à partir de l'état initial et tous ceux à partir desquels on ne peut pas atteindre un état final. On obtient ainsi la version *émondée* de l'automate, qui est équivalente à l'automate de départ.

On peut ensuite *minimiser* l'automate. Cette opération consiste à identifier deux états p et q lorsque ce sont exactement les mêmes mots qui permettent d'aller de p à un état final et de q à un état final. L'automate obtenu après cette opération, appelé *automate minimal* de L, est encore équivalent à l'automate de départ. On démontre que parmi tous les automates déterministes reconnaissant L, l'automate minimal est celui qui possède le plus petit nombre d'états.

7 Expressions rationnelles

Si on connaît un automate déterministe reconnaissant un langage L, on peut facilement tester si un mot appartient ou non à L. En revanche, il n'est pas toujours facile de décrire les mots de L. Les *expressions rationnelles*, appelées aussi *expression régulière*, permettent de résoudre ce problème.

L'expression $(a+b)^*a+\big((bab+ca)^*(b+\varepsilon)\big)^*c$ est un exemple d'expression rationnelle sur l'alphabet $\{a, b, c\}$. Comme on le voit, c'est une forme d'expression algébrique utilisant les lettres de l'alphabet, le mot vide, un opérateur +, un opérateur produit et une sorte d'exposant, noté *.

Le symbole + désigne le *ou* logique. Autrement dit, $L + L'$ représente l'union des deux langages L et L', formée des mots qui sont dans L

ou dans L'. Le produit de L et L', noté LL', est formé des mots qui sont produits d'un mot de L et d'un mot de L'.

$$L + L' = \{u \in A^* \mid u \in L \text{ ou } u \in L'\}$$
$$LL' = \{uu' \mid u \in L \text{ et } u' \in L'\}.$$

Pour $L = ab + abc$ et $L' = ab + cab$, on trouve $L + L' = ab + abc + cab$ et $LL' = abab + abcab + abccab$. Comme pour les mots, on peut définir les *puissances* d'un langage en posant $L^0 = \{\varepsilon\}$, $L^1 = L$, $L^2 = LL$, etc.

L'*étoile* d'un langage L, notée L^*, est l'union de toutes les puissances de L :

$$L^* = \sum_{n \geqslant 0} L^n.$$

De façon équivalente, L^* est formé de tous les mots qui peuvent s'écrire comme produit d'un nombre arbitraire de mots de L.

Les langages *rationnels* sont les langages décrits par des expressions rationnelles. Si $A = \{a, b\}$, les langages suivants sont rationnels : l'alphabet A, qui peut s'écrire $a + b$, l'ensemble $A^* = (a + b)^*$ formé de tous les mots sur l'alphabet A, le langage A^*aA^* des mots contenant la lettre a, le langage $(A^2)^*A$ des mots de longueur impaire, le langage $(ab)^*$ formé des mots $\varepsilon, ab, abab, ababab, \ldots$.

Les expressions rationnelles sont d'un usage courant en informatique. Ainsi, les éditeurs vi et emacs d'Unix utilisent des expressions rationnelles pour la recherche d'expressions. Une syntaxe voisine est utilisée par les analyseurs lexicaux (lex) ou les langages de script, tels que Perl ou PHP.

8 Le théorème de Kleene

Le théorème de Kleene, clef de voûte de la théorie des automates, affirme qu'un langage est rationnel si et seulement s'il est reconnaissable. De plus, la démonstration de ce théorème fournit un algorithme pour passer d'un automate à une expression rationnelle et vice-versa. Le lecteur pourra par exemple vérifier que le langage accepté par l'automate de la figure 1.1 est le langage rationnel $(aa^*ba)^*aa^*(b + \varepsilon)$.

Le théorème de Kleene a plusieurs conséquences importantes, notamment le fait que l'intersection de deux langages rationnels et la différence de deux langages rationnels sont encore des langages rationnels.

Pour illustrer la portée de ce dernier résultat, cherchons une expression rationnelle représentant le langage L des mots *ne contenant pas* la chaîne 010. Pour l'obtenir, on part de l'automate de la figure 1.5, qui reconnaît le langage des mots qui contiennent la chaîne 010. La figure 1.7 donne un automate déterministe équivalent.

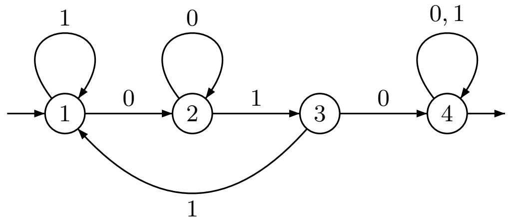

FIG. 1.7 – Déterminisation de l'automate

En échangeant les états finaux et non finaux, puis en supprimant l'état 4 devenu inutile, on obtient un automate déterministe qui reconnaît L (cf. figure 1.8).

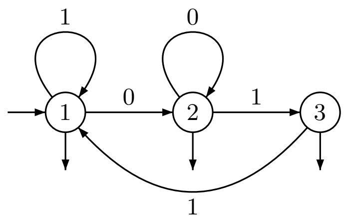

FIG. 1.8 – Un automate pour L

Il reste alors à convertir cet automate en expression rationnelle pour résoudre notre problème. Plusieurs réponses sont possibles, par exemple $(1 + 00^*11)^*(\varepsilon + 00^* + 00^*1)$.

9 Automates séquentiels

Les *automates séquentiels* sont des automates déterministes qui produisent un mot de sortie. Ils permettent de réaliser plusieurs transformations familières telles que l'addition des entiers, la multiplication par une constante, divers codages et décodages, le « couper-coller » dans un texte, etc.

Dans un automate séquentiel, l'état initial et les états finaux sont étiquetés par des mots et les transitions sont étiquetées par des couples formés d'une lettre et d'un mot séparés par une barre verticale. La lettre figurant à gauche de la barre est lue en entrée, et le mot figurant à droite de la barre est alors produit en sortie (voir figure 1.9). Le mot de sortie associé à un mot d'entrée s'obtient en lisant l'entrée depuis l'état initial et en produisant les sorties spécifiées par les transitions, y compris celles données au début par l'état initial et à la fin par l'état final. Si on ne parvient pas dans un état final, aucune sortie n'est produite.

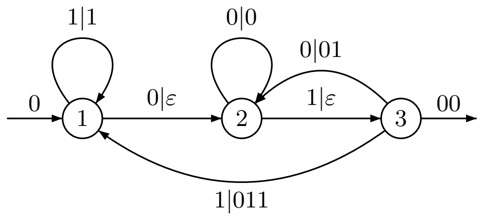

FIG. 1.9 – Un automate séquentiel

Considérons l'automate séquentiel de la figure 1.9 et prenons 1001101 comme mot d'entrée. En partant de l'état initial 1, et en lisant le mot de gauche à droite, on parcourt le chemin :

$$\xrightarrow{0} 1 \xrightarrow{1|1} 1 \xrightarrow{0|\varepsilon} 2 \xrightarrow{0|0} 2 \xrightarrow{1|\varepsilon} 3$$
$$3 \xrightarrow{1|011} 1 \xrightarrow{0|\varepsilon} 2 \xrightarrow{1|\varepsilon} 3 \xrightarrow{00}$$

La sortie est obtenue comme produit des mots de sortie $0, 1, \varepsilon, 0, \varepsilon, 011, \varepsilon, \varepsilon, 00$, soit 01001100.

Comme pour les automates déterministes, on peut *minimiser* un automate séquentiel, bien que l'algorithme soit plus délicat à mettre en place. Une fonction est dite *séquentielle* si elle peut être réalisée par un automate séquentiel. Un résultat essentiel assure que la composition de deux fonctions séquentielles est séquentielle.

Un automate séquentiel est dit *lettre à lettre* si les mots de sortie de chaque transition sont des lettres. Un *automate de Mealy* est un automate lettre à lettre dont l'état initial et les états finaux sont étiquetés par le mot vide, ce qui signifie en pratique que l'on peut ignorer ces étiquettes. Un *automate de Moore* est une variante d'automate séquentiel dans laquelle les sorties sont liées aux états traversés et non aux transitions. On peut démontrer que tout automate de Moore peut être simulé par un automate de Mealy et réciproquement.

La figure 1.10 représente un automate séquentiel lettre à lettre qui permet de réaliser la multiplication par 3 des entiers représentés sous forme binaire *inversée*. Comme son nom l'indique, la représentation binaire inversée est obtenue en lisant de droite à gauche la représentation binaire usuelle. Dans le jargon des informaticiens, c'est la représentation binaire dans laquelle le bit de droite a le plus fort poids.

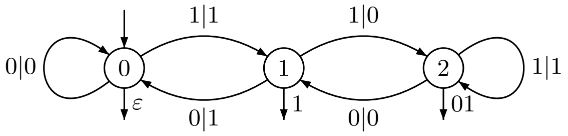

FIG. 1.10 – Un automate séquentiel lettre à lettre pour la multiplication par 3

Par exemple, le mot 1011 est la représentation binaire inversée du nombre $2^0+2^2+2^3 = 1+4+8 = 13$. Or $13\times 3 = 39$ et $39 = 2^0+2^1+2^2+2^5$ dont la représentation binaire inversée est 111001, ce qui est bien la sortie obtenue en lisant 1011 en entrée dans l'automate séquentiel de la figure 1.10.

L'addition des entiers est aussi une opération séquentielle. Pour la réaliser, on écrit les entiers n et m en binaire inversé et on rajoute éventuellement des zéros pour que les deux représentations aient la même longueur. Par exemple, si $n = 22$ et $m = 13$, on prend pour n la représentation 01101 (car $2+4+16 = 22$) et pour m la représentation 10110 (car $1+4+8 = 13$). On considère le couple (01101, 10110) comme un mot sur l'alphabet $\left\{\binom{0}{0}, \binom{1}{0}, \binom{0}{1}, \binom{1}{1}\right\}$ obtenu en superposant les deux représentations et en lisant les colonnes : $\binom{0}{1}\binom{1}{0}\binom{1}{1}\binom{0}{1}\binom{1}{0}$. L'automate qui réalise l'addition est représenté sur la figure 1.11. Il a pour alphabet de sortie $\{0, 1\}$. Les états 0 et 1 correspondent respectivement à l'absence et à la présence d'une retenue.

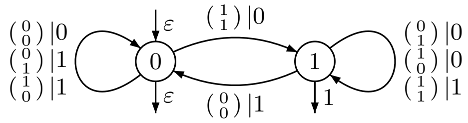

FIG. 1.11 – Un automate séquentiel lettre à lettre pour l'addition

Sur notre exemple, on trouve en sortie 110001, qui code bien $35 = 1 + 2 + 32$.

Un automate de Mealy suffit pour réaliser la division d'un polynôme à coefficients dans $\mathbb{Z}/2\mathbb{Z}$ par le polynôme $X^2 + X + 1$. Pour cela, on associe à chaque mot $a_0a_1 \cdots a_n$ sur l'alphabet $\{0, 1\}$ le polynôme $a_0X^n + a_1X^{n-1} + a_2X^{n-2} + \ldots + a_{n-1}X + a_n$ à coefficients dans $\mathbb{Z}/2\mathbb{Z}$.

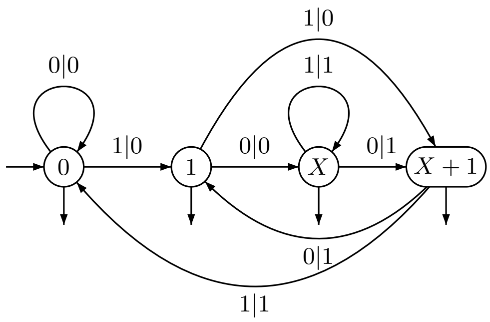

FIG. 1.12 – Division d'un polynôme à coefficients dans $\mathbb{Z}/2\mathbb{Z}$ par $X^2 + X + 1$

Le quotient de la division est donné par le mot de sortie et le reste par l'état d'arrivée. Ainsi le mot d'entrée 10001 donne en sortie le mot 00110 et arrive dans l'état $X + 1$, ce qui correspond à la formule $X^4+1 = (X^2+X+1)(X^2+X)+(X+1)$.

Les exemples qui précèdent sont génériques. Les automates séquentiels permettent de réaliser la multiplication et la division entière par une constante. Si on travaille sur des polynômes à coefficients dans un corps fini, l'addition, la multiplication ou la division par un polynôme constant peuvent être réalisées par des automates séquentiels.

En revanche, ni la multiplication, ni la division de deux entiers ne sont des opérations séquentielles. De fait, la conception d'un circuit électronique réalisant la multiplication de deux entiers est beaucoup plus difficile que celle d'un additionneur.

Voici un dernier exemple, de nature assez différente des précédents. Un *filtre à rebonds* lit une suite de bits commençant et finissant par 0 et change chaque 0 en 1 s'il est encadré par des 1, et chaque 1 en 0 s'il est encadré par des 0, les autres bits étant inchangés. Ainsi, si le mot d'entrée est 01011010, la sortie sera 00111100. La figure 1.13 représente un automate séquentiel qui réalise un filtre à rebonds.

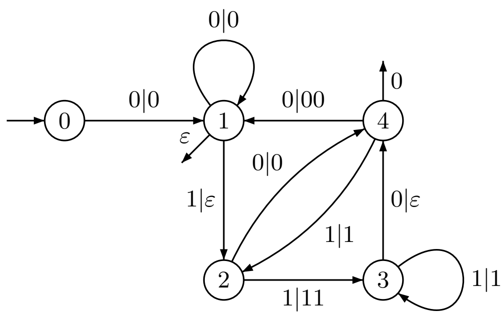

FIG. 1.13 – Un filtre à rebonds

Ce type de filtre est utilisé en traitement d'images. Chaque ligne de l'image est constituée d'une suite de bits. Les éventuelles imperfections de l'image, dues à une mauvaise qualité du film ou à un défaut de transmission peuvent être éventuellement corrigées de cette façon.

10 Utilisation pratique des automates

Les automates interviennent souvent dans la modélisation de problèmes concrets. C'est d'ailleurs un bon réflexe, face à un problème discret (i.e. ne faisant pas intervenir de nombres réels), de commencer par regarder si une approche par automate n'est pas envisageable. Le modèle s'applique même à certains problèmes continus. Il est vrai que les chaînes de Markov utilisées en probabilité sont de proches parents des automates.

Nous présentons ci-dessous quelques applications pratiques des automates. Toutefois, nous ne dirons que quelques mots de l'une des applications les plus importantes, la spécification et la vérification de systèmes (électroniques ou informatiques), car elle est traitée en détail au chapitre £Automates et vérification£.

10.1 Analyse lexicale

Les automates finis sont utilisés en compilation (voir chapitre £Compilateurs£) pour constituer des *analyseurs lexicaux*, qui permettent notamment de repérer les mots-clés d'un langage de programmation. L'automate de la figure 1.14, dans lequel les états finaux sont en noir, reconnaît un ensemble de mots-clés du langage Java : `do`, `double`, `final`, `finally`, `this`, `throw`, `throws`.

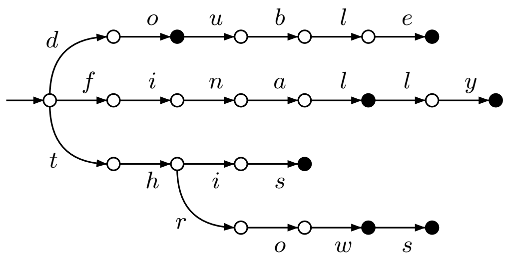

FIG. 1.14 – Mots-clés en Java

10.2 Modélisation de systèmes finis

On peut (au moins théoriquement !) utiliser des automates pour modéliser des situations dans lesquelles il n'y a qu'un nombre fini de configurations possibles. Par exemple, la figure 1.15 représente un modèle très rudimentaire de monte-charge. Les états symbolisent les étages. Le monte-charge est muni de deux boutons, D et M qui permettent de descendre ou monter d'un seul étage à la fois. Le bouton D est inactif au rez-de-chaussée, de même que M au deuxième étage.

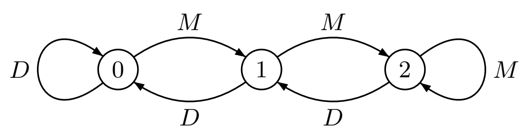

FIG. 1.15 – Un modèle de monte-charge

L'exemple qui suit repose sur une devinette classique. Un passeur doit faire passer d'une rive à l'autre un loup, une chèvre et une salade. Toutefois, son bateau ne peut transporter qu'un seul passager en dehors de lui-même. Bien entendu, il ne peut laisser le loup et la chèvre seuls sans surveillance, sinon le loup mangera la chèvre. Même chose pour le couple chèvre-salade, car la chèvre rêve de manger la salade. Pouvez-vous aider le passeur ?

On peut modéliser ce problème par l'automate de la figure 1.16.

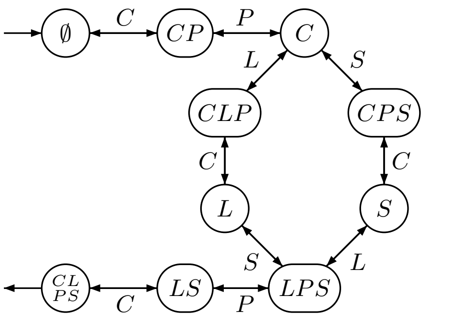

FIG. 1.16 – Le loup, la chèvre et la salade

Chaque état représente les protagonistes sur la rive opposée. Ainsi l'état CP signifie et que la chèvre et le passeur sont sur la rive opposée (et que le loup et la salade n'ont pas encore traversé). Comme le précise l'énoncé, certains états sont interdits. L'état initial est $\emptyset$ et l'état final et $CLPS$. Les actions possibles (qui constituent donc l'alphabet de l'automate) sont les suivantes :

(1) traverser seul (P)
(2) traverser avec le loup (L)
(3) traverser avec la chèvre (C)
(4) traverser avec la salade (S)

On obtient ainsi l'automate de la figure 1.16 qui donne immédiatement les deux solutions les plus courtes : $CPLCSPC$ et $CPSCLPC$.

10.3 Modélisation de systèmes infinis

Les automates permettent aussi de modéliser certains systèmes de taille non bornée, ou même infinie. Ces techniques sont particulièrement utilisées pour résoudre les problèmes d'analyse et de vérification de protocoles (nécessité de satisfaire certaines contraintes logiques, comme l'absence de blocage, ou l'exclusion mutuelle). Le plus souvent, on utilise des automates travaillant sur des mots infinis (voir chapitre £Automates et vérification£), mais l'exemple ci-dessous utilise uniquement des automates traditionnels.

Considérons un réseau de processus alignés qui communiquent avec chacun de leurs voisins, conformément à la figure 1.17.

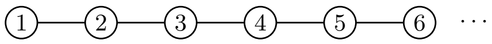

FIG. 1.17 – Un réseau de processus

Ces processus réalisent un calcul en se passant un témoin, qui est initialement détenu par le processus 1. À chaque étape du calcul, un processeur peut passer le témoin à son voisin de droite. Comment formaliser ce système ? On peut représenter la configuration du système par un mot sur l'alphabet $\{T, N\}$, dont la i-ème lettre vaut T ou N selon que le processeur i possède on non le témoin. S'il y a k processeurs, la configuration initiale est donc représentée par le mot du langage TN^{k-1}.

On représente ensuite la transmission du témoin par l'automate séquentiel de la figure 1.18. On peut donc considérer cette opération comme une fonction séquentielle $\tau : \{T, N\}^* \to \{T, N\}^*$.

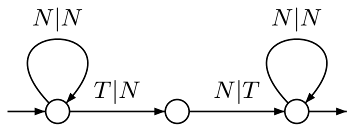

FIG. 1.18 – La transmission du témoin

La configuration du système après n transmissions de témoin s'obtient en itérant n fois la fonction τ à partir de la configuration initiale.

Cet exemple semblera peut-être élémentaire au lecteur, car l'itération de la fonction séquentielle se calcule ici sans difficulté. Mais c'est loin d'être toujours le cas ! Considérons par exemple l'automate séquentiel de la figure 1.19.

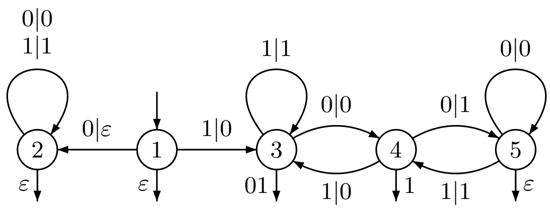

FIG. 1.19 – Un automate séquentiel

Si on utilise la représentation binaire inversée des entiers, cet automate calcule une fonction f bien connue des programmeurs, définie par :

$$f(n) = \begin{cases} n/2 & \text{si } n \text{ est pair} \\ 3n+1 & \text{sinon} \end{cases}$$

Une conjecture célèbre affirme que, partant d'un entier quelconque n, on arrive toujours à 1 en itérant f. Par exemple, en partant de $n = 31$, on trouve la suite 31, 94, 47, 142, 71, etc., qui aboutit à 1 après 106 itérations. La conjecture a été vérifiée pour $n \leqslant 5{,}764 \cdot 10^{17}$, mais n'est toujours pas résolue à ce jour. Elle montre en tout cas que l'étude de l'itération des fonctions séquentielles présente des difficultés insoupçonnées.

10.4 Industries de la langue

Les automates sont très utilisés dans ce qu'on appelle les industries de la langue : informatique linguistique, traitement des langues naturelles, correction orthographique automatique, génération automatique de texte, réalisation de dictionnaires électroniques, etc.

L'automate non déterministe de la figure 1.20, dû au linguiste Maurice Gross, indique l'ordre de succession des particules préverbales en français (sans tenir compte des élisions). Par exemple les phrases *je ne le lui ai pas donné* ou *il n'y en a jamais !* sont correctes, mais la phrase *il se le lui est fait volé* ne l'est pas... Afin d'alléger la figure 1.20, seules certaines particules préverbales sont représentées. Pour les obtenir toutes, il suffit d'opérer les substitutions suivantes :

$$
\begin{array}{rlrl}
je & \rightarrow je, tu, nous, vous, on, ce & y & \rightarrow y, \varepsilon \\
me & \rightarrow me, te, nous, vous, \varepsilon & ne & \rightarrow ne, \varepsilon \\
il & \rightarrow il, ils, elle, elles & se & \rightarrow se, \varepsilon \\
le & \rightarrow le, la, les, \varepsilon & en & \rightarrow en, \varepsilon \\
lui & \rightarrow lui, leur, \varepsilon & &
\end{array}
$$

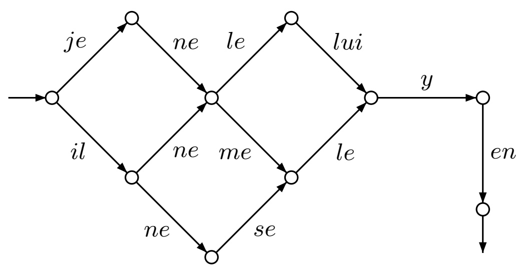

FIG. 1.20 – Suites de particules préverbales

Les automates sont également utilisés dans la réalisation de dictionnaires électroniques. La représentation la plus naïve d'un dictionnaire consiste à conserver la liste de tous les mots du dictionnaire. Une structure d'arbre permet d'obtenir une représentation plus concise. Ainsi, l'arbre de la figure 1.21 représente l'ensemble des mots {bal, bals, ban, bans, do, don, dons, doux, pal, pals, pan, pans}. Mais l'utilisation d'un automate (voir figure 1.22) donne une représentation beaucoup plus compacte.

Le gain en place peut être très important. Pour un dictionnaire de Scrabble anglais, la représentation par arbre nécessite 117 150 nœuds et 780 kilo-octets de mémoire. La représentation par automate nécessite seulement 19 853 états et 175 kilo-octets de mémoire, soit un gain de près de 80%.

Pour le Dictionnaire électronique des formes fléchies du français (DELAF) réalisé par les linguistes de l'université de Marne-la-Vallée, qui contient 802009 entrées sur un alphabet de 90 lettres (minuscules et majuscules accentuées, chiffres, et autres signes), la représentation par arbre nécessite 2203261 nœuds, alors que l'automate minimal ne possède que 273716 états.

De plus, on dispose d'algorithmes très efficaces pour les opérations de recherche et de mise à jour pour ces dictionnaires électroniques. Si on dispose d'un dictionnaire sous forme d'automate, on peut ensuite facilement réaliser un programme informatique champion du monde de Scrabble...

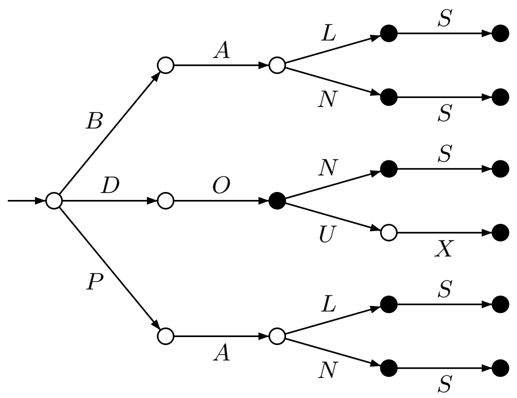

FIG. 1.21 – Représentation par arbre

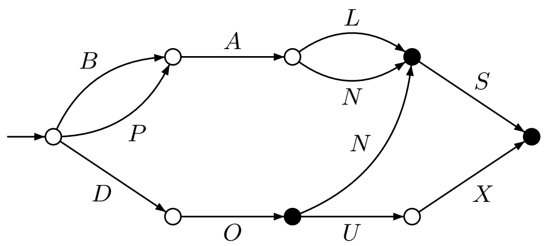

FIG. 1.22 – Représentation par automate

10.5 Recherche d'information

Tout ce qui concerne l'analyse de textes et l'extraction d'information fait grand usage des automates. Les automates interviennent souvent dans la conception d'algorithmes de recherche d'information. Nous nous contenterons ici de décrire le principe de la recherche dans un texte, dans lequel les automates interviennent de façon naturelle.

Supposons que l'on cherche à déterminer si la chaîne `ada` apparaît dans le texte `abracadabra`. Cela revient à tester si le mot `abracadabra` appartient au langage A^*adaA^*. Comme ce langage est rationnel, il suffit de trouver son automate minimal, puis de vérifier que le texte est bien accepté par cet automate.

Plus généralement, rechercher un mot u dans un texte conduit à calculer l'automate minimal du langage A^*uA^*. Si n est la longueur du mot

u, il est facile de construire un automate non déterministe à $n+1$ états pour ce langage (voir la figure 1.5, pour $u = 010$). Et, chose remarquable, l'automate minimal de ce langage possède lui aussi $n+1$ états. La figure 1.7 illustre cette construction pour la chaîne 010.

10.6 Autres applications

Citons également la manipulation d'images parmi les applications récentes. La compression, et les transformations usuelles telles que rotations, translations, homothéties peuvent en effet être réalisées par des automates séquentiels appropriés. N'oublions pas la théorie de la commande et du contrôle, où certains algorithmes récemment développés sont entièrement basés sur les automates. En théorie des jeux, dont les applications sont assez diverses, certaines stratégies peuvent être modélisées par des automates finis, ce qui donne en prime des algorithmes efficaces.

11 Extensions de la notion d'automate

L'utilisation des automates n'est pas limitée aux mots. Elle a été étendue aux *mots infinis*, qui sont des suites infinies de lettres $a_0a_1a_2\cdots$, aux arbres et à bien d'autres structures. Le cas des mots infinis mérite que l'on s'y arrête, en raison de ses applications à la vérification des systèmes.

11.1 Automates et mots infinis

Pour pouvoir utiliser des automates finis pour reconnaître des mots infinis, Büchi a proposé dès 1960 la définition suivante : un chemin infini est *réussi* s'il part d'un état initial et *passe infiniment souvent par un état final*. L'ensemble des mots infinis acceptés par un automate est alors l'ensemble des étiquettes des chemins infinis réussis. Considérons l'automate de la figure 1.23.

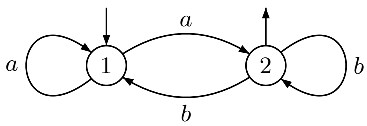

FIG. 1.23 – Un automate de Büchi

On voit que les mots infinis acceptés par cet automate sont ceux commençant par a et contenant une infinité de b.

En adaptant la notion d'expression rationnelle, on peut alors étendre le théorème de Kleene au cas des mots infinis. On montre cependant qu'un automate de Büchi non déterministe n'est pas toujours équivalent à un automate de Büchi déterministe. Pour rétablir cette équivalence, il faut utiliser un mode d'acceptation différent, introduit par Muller. Il consiste à donner non pas un ensemble d'états finaux, mais une table d'ensembles d'états $\{F_1, F_2, \ldots, F_k\}$. Un chemin infini est alors dit réussi s'il est issu d'un état initial et si l'ensemble F des états visités infiniment souvent figure dans la table.

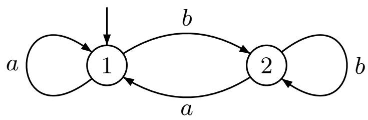

FIG. 1.24 – Un automate de Muller

Considérons l'automate de la figure 1.24. Si on prend pour table $\{\{2\}\}$, un chemin issu de l'état 1 est réussi s'il passe infiniment souvent par 2 et par 1 un nombre fini de fois. L'automate reconnaît donc l'ensemble des mots infinis ne contenant qu'un nombre fini de a.

Si on prend pour table $\{\{1,2\}\}$, l'automate reconnaît l'ensemble des mots infinis contenant une infinité de a et une infinité de b. Enfin, si on prend pour table $\{\{1\},\{2\}\}$ l'automate reconnaît l'ensemble des mots contenant soit une infinité de a et un nombre fini de b, soit un nombre fini de a et une infinité de b.

11.2 Transducteurs

Un automate non déterministe muni de sorties est appelé un *transducteur*. Les transducteurs permettent de représenter des *relations* entre mots. Ce sont des outils formels très puissants, qui ont aussi des applications pratiques, notamment dans les industries de la langue.

On peut aussi considérer des transducteurs dont les sorties ne sont pas des mots, mais des entiers ou des réels. Ce type de transducteur est utilisé notamment en traitement d'image. Une variante, connue sous le nom d'*automate max-plus* est particulièrement adaptée pour étudier les *systèmes à événements discrets*, qui comprennent par exemple les systèmes de production, les réseaux de transport, les systèmes informatiques, etc. Les automates permettent d'aborder certains problèmes d'évaluation de performance (calcul de taux de production, de débit), ou des problèmes d'optimisation (nombre optimal de processeurs pour réaliser une tâche donnée). Ils permettent en outre de formaliser simplement certains algorithmes d'accessibilité sur les graphes.

12 Autres développements

Il n'est pas possible de résumer en quelques lignes les développements actuels de la théorie des automates et nous nous contenterons d'évoquer quelques grands axes.

Les liens entre la *logique* et les automates remontent au tout début de la théorie (travaux de Kleene, Büchi, Rabin et Mc Naughton notamment). Le formalisme logique fournit en effet, après les automates et les expressions rationnelles, une troisième façon de décrire des ensembles de mots. Il s'applique également aux mots infinis et aux arbres et est très utilisé en vérification (voir chapitre £Automates et vérification£) pour spécifier le comportement de certains systèmes.

Les développements *algébriques* de la théorie des automates sont tout aussi importants. En dotant les mots de coefficients, on peut définir des *séries formelles en variables non commutatives*, pour lesquelles Schützenberger a étendu le théorème de Kleene. Ces séries ont de multiples applications, notamment en théorie du contrôle. Une autre branche en plein essor est la vision abstraite de la reconnaissance, développée par Schützenberger et Eilenberg. Elle consiste à remplacer les automates par des objets mathématiques, les *semigroupes*. Cette approche a permis de résoudre de très nombreux problèmes théoriques. Outre la logique et l'algèbre, les automates entretiennent également des liens étroits avec d'autres parties des mathématiques : topologie, probabilité, systèmes dynamiques, théorie des groupes, etc.

13 Pour en savoir plus...

Pour écrire ce chapitre, je me suis appuyé sur deux articles remarquables de Perrin (1989, 1995), que je recommande particulièrement au lecteur. Il y trouvera en particulier des compléments bibliographiques qui n'ont pas pu être repris ici.

Le lecteur francophone trouvera un exposé du théorème de Kleene et de ses conséquences dans le livre de Séébold (1999) et une présentation beaucoup plus exhaustive de la théorie des automates dans le récent traité de Sakarovitch (2003). Pour un exposé des algorithmes utilisant des automates, nous renvoyons au livre de Beauquier et al. (1992) ou, pour tout ce qui concerne les algorithmes sur les mots, à Crochemore et al. (2001).

On trouve souvent un chapitre ou deux consacrés aux automates dans les ouvrages sur la compilation ou sur la théorie des langages tels que le livre de Hopcroft et al. (2001). Pour les aspects les plus mathématiques, le lecteur pourra consulter le traité d'Eilenberg (1974 et 1976) ou encore Pin (1986). Le livre de Berstel et Reutenauer (1988) est consacré aux séries formelles et celui de Perrin et Pin (2004) aux automates sur les mots infinis.

Par ailleurs, plusieurs articles de synthèse parus ces dernières années décrivent divers aspects de la théorie des automates. Citons ceux de Perrin et de Thomas du *Handbook of Theoretical Computer Science* (van Leeuwen, 1990), et ceux de Béal et Perrin, de Pin, de Staiger, de Thomas et de Yu dans le *Handbook of Formal Languages* (Rozenberg et Salomaa, 1997).

Bibliographie

Beauquier D., Berstel J. et Chrétienne P. (1992), *Éléments d'algorithmique*. Masson.

Berstel J. et Reutenauer C. (1988), *Rational Series and Their Languages*. Springer-Verlag. (Traduction de *Les séries rationnelles et leurs langages*, paru chez Masson en 1984).

Crochemore M., Hancart C. et Lecroq T. (2001), *Algorithmique du texte*. Vuibert.

Eilenberg S. (1974 et 1976), *Automata, Languages and Machines*, volume A et B. Academic Press.

Hopcroft J. E., Motwani R. et Ullman J. D. (2001), *Introduction to Automata Theory, Languages and Computation*. Addison Wesley. second edition.

Perrin D. (1989), Automates et algorithmes sur les mots. *Annales des Télécommunications*, 44 :20–33.

Perrin D. (1995), Les débuts de la théorie des automates. *Technique et Science Informatique*, 14 : 409–433.

Perrin D. et Pin J.-É. (2004), *Infinite Words. Automata, semigroups, logic and games*, volume 141 of *Pure and Applied Mathematics*. Elsevier.

Pin J.-É. (1986), *Varieties of formal languages*. North Oxford, London et Plenum, New-York. (Traduction de *Variétés de langages formels*, paru chez Masson en 1984).

Rozenberg G. et Salomaa A. (1997), *Handbook of formal languages*. Springer Verlag. 3 volumes.

Sakarovitch J. (2003), *Éléments de théorie des automates*. Vuibert, Paris.

Séébold P. (1999), *Théorie des automates – Méthodes et exercices corrigés*. Vuibert, Paris.

van Leeuwen J. (1990), *Handbook of theoretical computer science. Vol. B.* Elsevier Science Publishers B.V., Amsterdam.

Index