Automates & Langages

Frédéric Olive [1]

2010 / 2011

1. LIF/CMI, 39 rue joliot Curie, 13453 Marseille - 04 13 55 13 16 - frederic.olive@lif.univ-mrs.fr

Table des matières

Introduction

Quels sont les problèmes susceptibles d'un traitement informatique ? Cette interrogation est fondamentale pour l'Informatique. D'une certaine manière, elle revient à demander quel est l'*objet* de cette discipline. De plus, elle revêt une dimension tout à fait pratique : l'informaticien appelé à écrire un programme doit s'assurer que le traitement du problème qui l'occupe est effectivement automatisable. Ce point est plus litigieux qu'il n'y parait : on peut facilement exhiber deux problèmes très voisins, dont l'un relève d'un algorithme standard, alors qu'on sait de l'autre qu'aucun programme ne pourra jamais le résoudre. Mais l'approche de cette question pratique nécessite le recours à des outils théoriques. Essentiellement parce que pour identifier les problèmes qui acceptent un traitement algorithmique, il nous faut définir *rigoureusement* les deux concepts en jeu : celui de *problème* et celui d'*algorithme*. Le premier n'opposera pas grande résistance : nous lui réglerons même son compte avant la fin de cette introduction ! Pour le deuxième, ce sera une autre paire de manches. À dire vrai, nous n'auront pas de trop de ce cours pour donner un premier aperçu de la manière dont les théoriciens l'ont formalisé.

Venons-en donc à la première interrogation : **qu'est-ce qu'un problème ?** On distingue plusieurs grandes familles de problèmes : les problème d'*évaluation*, d'*optimisation*, d'*approximation*, d'*énumération*, de *comptage*, de *décision*. Les plus importantes sont sans doute celles des problème d'*évaluation*, pour lesquels on cherche à calculer la valeur d'une fonction sur différentes entrées, et celle des problèmes de *décision*, où l'on veut déterminer si les entrées vérifient une propriété fixée. Par exemple, le calcul du pgcd de couples d'entiers pris en entrée constitue un problème d'évaluation. *A contrario*, déterminer si un $x \in \mathbb{N}$ pris en entrée est premier est un problème de décision.

Quelle que soit la famille considérée, il est important de noter qu'**un problème d'informatique comporte une infinité d'instances**. C'est dans ce cadre que prend sens la notion d'algorithme. Ainsi, il serait absurde d'imaginer un al-

gorithme répondant à la seule question : « 132 683 est-il premier ? ». [1] C'est lorsqu'un problème comporte un nombre abitrairement grand d'instances que l'automatisation de son traitement devient utile et pertinente. Ce qui est crucial dans la méthode de multiplication que nous connaissons tous, par exemple, c'est qu'elle permette de calculer le produit de deux nombres *quelconques*, selon une « recette » qui s'applique de la même manière, quels que soient les nombres en jeu...

Par ailleurs, dans le contexte de formalisation qui est le notre, on ne perd pas de généralité à se restreindre aux problèmes de décision. Si l'on parvient à donner une définition rigoureuse de cette classe de problèmes, la formalisation des autres familles s'en suivra aisément. Par conséquent, **dans ce cours le terme « problème » signifiera « problème de décision »**. Ce que nous exprimons encore en écrivant que pour nous, tout problème est de la forme :

Entrée : $x \in \mathscr{I}$;

Question : x satisfait-il la propriété P ?

où $\mathscr{I}$ désigne un ensemble quelconque, dont les éléments sont appelés *instances* ou *entrées* du problème, et P est une propriété des éléments de $\mathscr{I}$.

Une dernière considération permettra de finaliser notre définition du concept de problème. Il est facile de se convaincre qu'un programme ne manipule des objets abstraits que par le biais d'une *représentation* de ces objets. Et la manière dont il opère est liée à cette représentation. Ainsi, l'algorithme de multiplication évoqué plus haut est intimement lié à la représentation décimale des entiers. (Pour vous en convaincre, essayez d'appliquer cet algorithme à des nombres écrits en caractères Romain !) Les entrées d'un problème d'informatique sont donc les codages d'objets abstraits, plutôt que ces objets eux-mêmes. Et un codage n'est rien d'autre qu'un mot sur un alphabet donné (assorti de règles permettant d'en décoder la signification). En somme, pour un programme, une instance d'un problème est toujours un mot. Et résoudre le problème consiste à décider si cette entrée appartient à l'ensemble des mots qui satisfont une certaine propriété P. Or un ensemble de mots, c'est exactement ce que nous appellerons bientôt un *langage formel*. Ainsi, **pour un informaticien, résoudre un problème c'est reconnaitre un langage**, c'est à dire identifier les mots qui le constituent. Avec la notation introduite plus haut, un problème sera donc toujours associé à un langage L et s'exprimera sous la forme :

1. La réponse est non, puisque $132\,683 = 277 * 479$.

Entrée : un mot x ;

Question : x appartient-il au langage L ?

Ceci clôt notre entreprise de formalisation du concept de problème, sous réserve de se mettre bien d'accord sur ce que nous appelons « langage ». C'est l'un des objectifs de ce cours.

Nous l'avons annoncé, la question « **qu'est-ce qu'un algorithme ?** » est autrement plus difficile à clarifier. Les termes *algorithme*, *programme*, *procédure*, se réfèrent tous à l'idée de succession d'opérations atomiques, selon un ordre qui dépend des objets pris en entrée. Dans le cas de l'algorithme de multiplication usuel, par exemple, les opérations atomiques sont les produits de deux nombres inférieurs à 9 (facile à exécuter, dès lors que l'on connait les tables de multiplication), et la séquence de produits élémentaires à effectuer découle facilement de l'écriture décimale des nombres à multiplier. La généralisation de ces considérations, en vue de définir le mot *algorithme*, mène à la notion de « modèle de calcul ». Un modèle de calcul est un objet mathématique – formé d'ensembles, de fonctions, de relations –, qui abstrait les propriétés évoquées plus haut en définissant une notion très simple d'opération atomique et en précisant scrupuleusement comment ces opérations se succèdent sur une entrée donnée pour former un calcul. Tenter de préciser d'avantage ces aspects nous emmènerait trop loin. Concluons simplement en annonçant que les *automates*, qui sont au cœur de ce cours, forment précisément un premier exemple – le plus simple qui soit – de modèle de calcul.

Chapitre 1

Langages réguliers

1.1 Alphabets - Mots - Langages

Définition 1 (Alphabet et mots). *Un* alphabet *V est un ensemble fini et non vide. Les éléments de V sont appelés* lettres, caractères *ou* symboles. *Un* mot *sur un alphabet V est une suite finie, eventuellement vide, de symboles de V. L'ensemble des mots sur V est noté V^*. Le* mot vide *(suite vide) est noté ε. La* longueur *d'un mot $w \in V^*$ est le nombre de symboles qui le composent. On la note $|w|$. En particulier : $|\varepsilon| = 0$.*

EXEMPLE. L'ensemble $V = \{a, b, c, 0, 1\}$ est un alphabet. Les suites 00, *abab* et *aabcc*001 sont des mots sur V, de longueurs respectives 2, 4 et 8.

Pour $u \in V^*$ et $i \leq |u|$, on note $u[i]$ le $i^{\text{ème}}$ symbole de u. On dit que u *s'écrit* $a_1 \dots a_n$ sur V si $u[1] = a_1, \dots, u[n] = a_n$. Deux mots u, v sont égaux si $|u| = |v|$ et si pour tout $i = 1, \dots, |u| : u[i] = v[i]$. Si u s'écrit $a_1 \dots a_n$ sur V et si $1 \leq i \leq j \leq |u|$, on note $u[i..j]$ le facteur de u compris entre le $i^{\text{ème}}$ et le $j^{\text{ème}}$ caractère de u. Autrement dit : $u[i..j] = a_i \dots a_j$.

Pour tout mot w et toute lettre a, on note $|w|_a$ le nombre d'occurences de a dans w. Autrement dit : $|w|_a = \text{card}\,\{i \in \mathbb{N} \text{ t.q. } w[i] = a\}$.

Définition 2 (Concatenation des mots). *Le* concaténé *de deux mots u et v est le mot obtenu en juxtaposant le mot v à la suite du mot u. On note $u.v$ le mot obtenu ou, plus simplement, uv. Ainsi, si u et v s'écrivent respectivement $u_1 \dots u_m$ et $v_1 \dots v_p$ sur un alphabet V, alors $uv = u_1 \dots u_m v_1 \dots v_p$.*

L'opération de concaténation est clairement associative : pour tous $u, v, w \in V^*$, $(uv)w = u(vw)$. De plus, le mot vide en est l'élément neutre : pour tout $u \in V^*$, $u\varepsilon = \varepsilon u = u$. L'itération de l'opération de concaténation sur un mot w permet d'obtenir les *puissances* de w : $w^0, w^1, w^2, \ldots$ Formellement, on définit par récurrence :

$$w^0 = \varepsilon \text{ et } \forall n \in \mathbb{N},\ w^{n+1} = ww^n.$$

Définition 3 (Facteur, préfixe, suffixe). *Un mot u est un* facteur *d'un mot $w \in V^*$ s'il existe $\alpha, \beta \in V^*$ tels que $w = \alpha u \beta$. Si de plus $\alpha = \varepsilon$ (i.e. $w = u\beta$), u est dit* préfixe*, ou* facteur gauche*, de w. Si $\beta = \varepsilon$ (i.e. $w = \alpha u$), u est dit* suffixe*, ou* facteur droit*, de w.*

EXEMPLES.

- Soit $w = aabccaa$. Alors bcc est un facteur de w ; $aabc$ est un préfixe de w ; caa est un suffixe de w ; aa est à la fois préfixe et suffixe de w.
- Pour tout mot w, ε et w sont à la fois préfixe et suffixe de w.

Les propriétés qui suivent résultent très directement des définitions vues plus haut. Malgré leur caractère d'évidence, nous les regroupons dans un lemme pour nous y référer facilement par la suite :

Lemme 4 *Soient u, v, x, y des mots sur un alphabet V, et $a \in V$ une lettre. Alors :*

1. $|uv| = |u| + |v|$.
2. $|uv|_a = |u|_a + |v|_a$.
3. *pour tout $i \leq |uv|$: $(uv)[i] = u[i]$ si $i \leq |u|$, $v[i - |u|]$ sinon.*
4. $(xu = yv$ *et* $|x| \leq |y|) \Rightarrow x$ *est préfixe de y et v est suffixe de u.*
5. $(xu = yv$ *et* $|x| = |y|) \Rightarrow x = y$ *et* $u = v$.
6. $|u| = 0$ *ssi* $u = \varepsilon$.
7. $|u| \geq n$ *ssi* $\exists x, y \in V^*$ *tels que* $u = xy$ *et* $|x| = n$ *et* $|y| \geq 0$.

Preuves inductives. Le lemme suivant et son corollaire fournissent un outil très utile pour établir des propriétés des mots sur un alphabet fixé.

Lemme 5 *Si un sous-ensemble X de V^* satisfait :*

(i) $\varepsilon \in X$

(ii) $\forall u \in X, \forall a \in V : au \in X$

alors $X = V^*$.

PREUVE. Il s'agit d'établir que tout $u \in V^*$ appartient à X. Montrons cela par récurrence sur la longueur de u : Si $|u| = 0$, alors $u = \varepsilon$ et donc $u \in X$ par (i). Supposons la propriété vraie pour tout mot de longueur un entier $n \geq 0$ fixé. Soit alors u un mot de longueur $n+1$. Puisque $n+1 > 0$, u peut s'écrire $u = ax$, où a est une lettre et x est un mot de longueur n. Par hypothèse de récurrence, on peut affirmer que $x \in X$. Mais alors, en appliquant (ii), on a aussi $u = ax \in X$. Ainsi, l'hypothèse de récurrence est vraie au rang 0 et sa validité au rang n entraîne sa validité au rang $n+1$. Par conséquent, elle est vraie pour tout $n \in \mathbb{N}$: c'est le résultat cherché. □

Etant donnée une propriété P portant sur les mots de V^*, convenons de noter $P(u)$ le fait qu'un mot u satisfait la propriété P.

Corollaire 6 (Preuves inductives). *Soit P une propriété définie sur V^* et telle que :*

- *$P(\varepsilon)$ et*
- *si u satisfait P, alors il en va de même de au pour tout $a \in V$.*

Alors P est vraie de tout $u \in V^*$.

PREUVE. Il suffit de constater que l'ensemble $X = \{u \in V^* \text{ t.q. } P(u)\}$ vérifie les conditions (i) et (ii) du Lemme 5. Par conséquent $X = V^*$, ce qui signifie bien que tout $u \in V^*$ satisfait P. □

Le *miroir d'un mot* $w \in V^*$, noté w^R, est le mot obtenu en inversant l'ordre des lettres de w. Autrement dit, si w s'écrit $w = w_1 \dots w_n$ sur V, alors $w^R = w_n \dots w_1$. Un *palindrome* est un mot w tel que $w = w^R$.

EXEMPLES.

- $abbaa^R = aabba$; $100^R = 001$.
- ressasser, eluparcettecrapule, 10101 et $xyyx$ sont des palindromes.

Définition 7 (Langage formel). *Un* langage formel *sur un alphabet V est une partie de V^*. Autrement dit, c'est un ensemble de mots sur V.*

EXEMPLES.

- Les ensembles $\{00, 01, 110, 1010\}$ et $\{w \in \{0,1\}^* \text{ t.q. } |w|_1 = 1\}$ sont des langages sur l'alphabet $\{0,1\}$. Le premier est fini, le second infini (il contient, par exemple, la suite infinie de mots : $1, 01, 001, 0001, \dots$).

- L'ensemble des écritures d'entiers en base 10 est un langage sur l'alphabet $\{0,1,\ldots,9\}$.
- Pour tout alphabet V, les ensembles V^*, $\{\varepsilon\}$ et $\varnothing$ sont des langages sur V. Le dernier est appelé *langage vide*. On prendra garde à ne pas confondre le langage vide, qui ne contient aucun mot, et le langage $\{\varepsilon\}$, qui contient le mot vide.

Les opérations sur les langages sont les opérations usuelles sur les ensembles : union, intersection, différence, complémentaire. À cela s'ajoute la *concaténation des langages*, définie par extension de la concaténation des mots :

Définition 8 (Concaténation des langages). *si L, L' sont deux langages sur un alphabet V, le* concaténé des langages *L et L' est le langage*

$$LL' = \{uv \text{ t.q. } u \in L, v \in L'\}.$$

L'itération de cette opération sur un langage L permet d'obtenir les *puissances* de L : $L^0, L^1, L^2, \ldots$ Formellement, on définit par récurrence :

$$L^0 = \{\varepsilon\} \text{ et } \forall n \in \mathbb{N},\ L^{n+1} = LL^n.$$

Enfin, on appelle *fermeture de Kleene* (ou *l'étoile*) du langage L, le langage :

$$L^* = \bigcup_{n \geq 0} L^n.$$

Ainsi L^* est-il constitué des mots de de V^* qui s'obtiennent par concaténation d'un nombre quelconque (eventuellement nul) de mots de L. On appelle *étoile stricte* de L l'ensemble des mots obtenus par concaténation d'un nombre non nul de mots de L. On note L^+ ce langage. Autrement dit :

$$L^+ = \bigcup_{n \geq 1} L^n.$$

Notons que quel que soit le langage L, sa fermeture de Kleene L^* contient ε (puisque $\{\varepsilon\} = L^0 \subset L^*$). En particulier : $\varepsilon \in \varnothing^*$.

EXEMPLE. Soient $L = \{ab, ba\}$ et $L' = \{aa, b\}$ deux langages sur $\{a, b\}$. Alors : $LL' = \{abaa, abb, baaa, bab\}$, $L^0 = \{\varepsilon\}$, $L^1 = LL^0 = L\{\varepsilon\} = L$, $L^2 = LL^1 = LL = \{abab, abba, baab, baba\}$, etc.

Définition 9 (Clôture pour la concaténation). *Un langage $L \subseteq V^*$ est* clos pour la concaténation *si pour tous $x, y \in V^*$: $x, y \in L \Rightarrow xy \in L$.*

Lemme 10 *Pour tout langage L, L^* est le plus petit langage contenant L et clos pour la concaténation.*

PREUVE. Montrons d'abord que L^* est bien clos pour la concaténation : si $x, y \in L^*$, alors il existe $n, p \in \mathbb{N}$ tels que $x \in L^n$ et $y \in L^p$. Autrement dit, il existe $x_1, \ldots, x_n \in L$ et $y_1, \ldots, y_p \in L$ tels que $x = x_1 \ldots x_n$ et $y = y_1 \ldots y_p$. Par conséquent $xy = x_1 \ldots x_n y_1 \ldots y_p \in L^{n+p} \subset L^*$. Par ailleurs, si L' est un langage contenant L et clos pour la concaténation, on a, pour tout $n \in \mathbb{N}$: $\forall x_1, \ldots, x_n \in L$, $x_1 \ldots x_n \in L'$. D'où : $L^n \subseteq L'$ pour tout n, et donc $L^* = \bigcup_n L^n \subseteq L'$. □

Lemme 11 *Soient A, B, C trois langages sur un alphabet V. Alors :*

- $(AB)C = A(BC)$.
- $(A \cup B)C = AC \cup BC$.
- $(A \cap B)C \subset AC \cap BC$ *et l'inclusion réciproque est fausse en général.*
- $A \subseteq B \Rightarrow A^* \subseteq B^*$.
- $(A^*)^* = A^*$.
- $(A \cup B)^* = (A^* B^*)^*$.

PREUVE. Exercice. □

Le *miroir d'un langage L*, noté L^R, est l'ensemble des miroirs des mots de L. Autrement dit : $L^R = \{w^R \text{ t.q. } w \in L\}$.

1.2 Langages réguliers

Soit $\mathscr{P}(V^*)$ l'ensemble des parties de V^*, c'est-à-dire l'ensemble des langages d'alphabet V. L'union, la concaténation des langages et la fermeture de Kleene sont des opérations d'arités respectives 2, 2 et 1 sur $\mathscr{P}(V^*)$. La classe des langages réguliers est définie inductivement à partir de ces opération et des atomes $\varnothing$, $\{\varepsilon\}$, $\{a\}$, $a \in V$. Plus précisément :

Définition 12 (Langages réguliers). *L'ensemble* REG *des* langages réguliers *sur un alphabet V est le plus petit des sous-ensembles X de $\mathscr{P}(V^*)$ qui satisfont :*

1. *$\varnothing \in X$, $\{\varepsilon\} \in X$ et $\{a\} \in X$ pour chaque $a \in V$;*
2. *si $A, B \in X$, alors $A \cup B$, AB et A^* sont dans X.*

Autrement dit, REG est le plus petit ensemble de langages sur V qui contienne le langage vide et tous les langages réduit à un mot de longueur ≤ 1, et qui soit clos pour l'union, la concaténation et la fermeture de Kleene. Ou encore : les langages réguliers sont les langages obtenus à partir des « atomes » $\varnothing$, $\{\varepsilon\}$ et $\{a\}$, $a \in V$, par un nombre fini d'applications des trois opérations précitées.

EXEMPLES.

- Pour tout mot $u \in V^*$, le langage $\{u\}$ est régulier. En effet, si u s'écrit $a_1 \dots a_n$ sur V, alors le langage $\{u\}$ s'écrit comme la concaténation $\{u\} = \{a_1\} \cdot \{a_2\} \dots \{a_n\}$. La régularité de $\{u\}$ se déduit alors de celle de chaque $\{a_i\}$ et du fait que REG est clos pour la concaténation des langages.

- Tout langage fini est régulier. En effet, un ensemble fini de mots $L = \{u_1, \dots, u_k\}$ s'écrit : $L = \{u_1\} \cup \{u_2\} \cup \{u_3\} \cup \dots \cup \{u_n\}$. C'est donc une union finie de langages réduits à un seul mots et on a vu que de tels langages sont réguliers. La conclusion découle du fait que REG est clos pour l'union.

- L'ensemble de tous les mots, V^*, est régulier. Si $V = \{a_1, \dots, a_p\}$, il découle du résultat précédent que le langage $\{a_1, \dots, a_p\}$ est régulier. Mais V^* n'est autre que la fermeture de Kleene de ce langage, et la conclusion découle de la fermeture de REG pour cette opération.

- Sur l'alphabet $\{a, b\}$, l'ensemble $\{a^n b^p \,|\, n, p \in \mathbb{N}\}$ des mots qui s'écrivent comme une suite de a suivie d'une suite de b, est régulier. En effet : le langage $\{a^n \,|\, n \in \mathbb{N}\} = \{a\}^*$ est régulier, comme fermeture de Kleene du langage régulier $\{a\}$; pour la même raison, $\{b^p \,|\, p \in \mathbb{N}\} = \{b\}^*$ est régulier. Le langage $\{a^n b^p \,|\, n, p \in \mathbb{N}\}$, concaténé des deux précédents, est alors régulier, par fermeture de REG pour la concaténation.

1.3 Expressions régulières

Définition 13 (Expressions régulières). *L'ensemble ER des* expressions régulières *sur un alphabet V est le plus petit des langages L sur l'alphabet $V \cup \{+, *, \varnothing, \varepsilon\}$ qui satisfont :*

1. *$\varnothing \in L$, $\varepsilon \in L$ et $a \in L$ pour chaque $a \in V$;*
2. *si $\alpha, \beta \in L$, alors $\alpha + \beta$, $\alpha\beta$ et α^* sont dans L.*

A chaque expression régulière ξ sur V on associe un langage sur V selon les modalités suivantes :

1. $L(\varnothing) = \varnothing$; $L(\varepsilon) = \{\varepsilon\}$;

2. $L(a) = \{a\}$ pour tout $a \in V$;
3. $L(\alpha + \beta) = L(\alpha) \cup L(\beta)$;
4. $L(\alpha\beta) = L(\alpha)L(\beta)$;
5. $L(\alpha^*) = L(\alpha)^*$.

Le langage ainsi associé à une expression régulière ξ est dit *dénoté* par ξ.

EXEMPLES.
- $\{a^n b^p,\ n, p \geq 0\}$ est dénoté par a^*b^*.
- $\{aabc^n,\ n \geq 0\}$ est dénoté par $aabc^*$.
- $\{a_1, \ldots, a_k\}^*$ est dénoté par $(a_1 + \cdots + a_k)^*$;
- $(a+b)^*a(a+b)^*$ dénote le langage $\{w \in \{a,b\}^* \text{ t.q. } |w|_a \geq 1\}$.

Théorème 14 *Un langage est régulier si, et seulement si, il est dénoté par une expression régulière. Autrement dit :*

$$\mathsf{REG} = \{L(\alpha) \mid \alpha \in \mathsf{ER}\}.$$

PREUVE. $\subseteq$: il s'agit de montrer que tout langage régulier est dénoté par une expression régulière. On s'appuie sur la définition inductive de REG. L'assertion est clairement vraie des atomes de cette définition : le langage $\varnothing$ est dénoté par l'expression régulière $\varnothing$; le langage $\{\varepsilon\}$ est dénoté par l'expression régulière ε ; pour tout $a \in V$, le langage $\{a\}$ est dénoté par l'expression régulière a. Soient maintenant A, B deux langages dénotés par des expressions régulières α, β. Alors $A \cup B$, AB, A^* sont eux aussi dénotés par des expressions régulières, à savoir, respectivement : $\alpha + \beta$, $\alpha\beta$, α^*. Ceci clôt la preuve de la première inclusion.

$\supseteq$: nous devons montrer que pour toute expression régulière α, le langage $L(\alpha)$ est régulier. On s'appuie sur la définition inductive de l'ensemble des expressions régulières. L'assertion est clairement vraie des atomes de cette définition : $L(\varnothing)$, $L(\varepsilon)$ et $L(a)$ $(a \in V)$ sont réguliers. Supposons maintenant que les langages $L(\alpha)$ et $L(\beta)$ associés à deux expressions régulières α et β soient réguliers. Alors les langages associés aux expressions $\alpha + \beta$, $\alpha\beta$ et α^* valent respectivement : $L(\alpha) \cup L(\beta)$, $L(\alpha)L(\beta)$ et $L(\alpha)^*$. Ces langages sont réguliers, puisque $L(\alpha)$ et $L(\beta)$ le sont et puisque REG est fermé pour l'union, la concaténation et la fermeture de Kleene. Ceci clôt la preuve inductive de l'inclusion $\supseteq$. □

Définition 15 (Expressions régulières équivalentes). *Deux expressions régulières α, β sont dites* équivalentes *si elles dénotent le même langage. On note alors $\alpha \equiv \beta$.*

Les propriétés des opérations usuelles sur les langages se transcrivent en propriétés des opérations portant sur les expressions régulières. Voici les principales :

Proposition 16 *Soient α, β, γ trois expressions régulières sur un alphabet V. On a :*

1. $\varnothing + \varepsilon \equiv \varepsilon + \varnothing \equiv \varepsilon$
2. $\alpha\varepsilon \equiv \varepsilon\alpha \equiv \alpha$
3. $\alpha\varnothing \equiv \varnothing\alpha \equiv \varnothing$
4. $\alpha + \beta \equiv \beta + \alpha$
5. $(\alpha\beta)\gamma \equiv \alpha(\beta\gamma)$
6. $(\alpha + \beta) + \gamma \equiv \alpha + (\beta + \gamma)$
7. $\alpha(\beta + \gamma) \equiv \alpha\beta + \alpha\gamma$
8. $(\alpha + \beta)\gamma \equiv \alpha\gamma + \beta\gamma$
9. $\alpha + \alpha \equiv \alpha$
10. $\varnothing^* \equiv \varepsilon$
11. $\alpha\alpha^* \equiv \alpha^*\alpha$
12. $\alpha\alpha^* + \varepsilon \equiv \alpha^*$

PREUVE. Exercice. □

Concluons cette section par une dernière notation : on notera souvent α^+ l'expression $\alpha\alpha^*$. Clairement, $L(\alpha^+) = L(\alpha)^+$.

1.4 Automates finis déterministes

Dans l'introduction de ce cours, nous avons évoqué la notion de « reconnaissance des langages » dont l'importance tient au fait qu'elle permet de formaliser tous les problèmes de décision susceptibles d'intéresser les informaticiens. Rappelons que reconnaître un langage, c'est décider pour n'importe quel mot s'il appartient ou non au langage. Le propos de cette section, c'est de montrer que la reconnaissance des langages *réguliers* est un problème facile, en ce qu'il peut être résolu par un modèle particulièrement simple de « machines » : les automates finis.

L'automate fini déterministe est un modèle de calcul rudimentaire comportant :

- un *ruban d'entrée* sur lequel est écrit le mot à analyser. Ce ruban est divisé en case, et le mot d'entrée est stocké à raison d'une lettre par case.
- une *tête de lecture*, qui parcourt les cases du ruban en examinant leur contenu ;
- une *unité de contrôle*, à laquelle parviennent les informations collectées par la tête de lecture. L'unité de contrôle est caractérisée par l'ensemble des *états* dans lesquels elle peut se trouver et par son programme, appelé *fonction de transition* de l'automate, qui lui permet d'évoluer d'un état à un autre. Cette évolution dépend de l'état courant de l'unité de contrôle mais aussi du caractère transmis par la tête de lecture.

Ce dispositif est illustré Figure 1.4. Pour conclure cette description informelle, précisons la manière dont l'automate fonctionne et le biais par lequel il renvoit sa réponse – positive ou négative – quant à l'appartenance du mot d'entrée à un langage fixé : parmi les états possibles de l'UC, l'un est qualifié d'*initial*, et certains sont dits *acceptant*. Lorsqu'il démarre son calcul sur une entrée w, l'automate est dans l'état initial et la tête de lecture est positionnée sur la première lettre de w. Puis il lit une par une les lettres de w, en changeant d'état au gré des caractères lus. Lorsque la tête de lecture a scanné la dernière lettre de w, le calcul s'arrête. Le mot lu est alors considéré comme accepté par l'automate si l'état de l'UC à l'issue du calcul est un état acceptant.

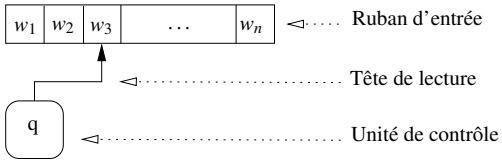


FIGURE 1.1 – Un automate fini

Voici la définition formelle :

Définition 17 (Automate fini déterministe). *Un* automate fini déterministe *(AFD en abrégé) est un quintuplet* (V, Q, δ, q_0, F)*, où :*

1. V *est un alphabet,*
2. Q *est un ensemble fini dont les éléments sont appelés* états *de l'automate,*
3. $\delta : Q \times V \to Q$ *est une fonction dite* fonction de transition*,*
4. $q_0 \in Q$ *est un état désigné appelé* état initial*, et*

5. *F est une partie de Q dont les éléments sont appelés* états acceptants *ou* états finaux.

Si $p,q \in Q$ et $a \in V$, on notera souvent $\delta(p,a) = q$ sous la forme $p \xrightarrow[\mathscr{A}]{a} q$, ou même $p \xrightarrow{a} q$ s'il n'y a pas d'ambiguïté sur l'automate sous-jacent. Si un mot w s'écrit $w_1 \dots w_n$ sur l'alphabet V, on appelle *calcul* de $\mathscr{A}$ sur w à partir de p l'unique suite d'états $p_1, p_2, \dots, p_n$ telle que :

$$p = p_1 \xrightarrow{w_1} p_2 \xrightarrow{w_2} \dots \xrightarrow{w_n} p_n.$$

L'état p_n est alors appelé *résultat* du calcul. On dit aussi que le calcul de $\mathscr{A}$ sur w à partir de p *termine* dans l'état p_n. On note $p \xrightarrow{w} q$ le fait que le calcul de $\mathscr{A}$ sur w à partir de p termine en q. Si X est une partie de Q, on note de plus $p \xrightarrow{w} X$ le fait que le calcul de $\mathscr{A}$ sur w à partir de p résulte en un état appartenant à X.

Dans le cas particulier de l'état initial q_0, on appellera simplement *calcul de $\mathscr{A}$ sur w* (sans mentionner q_0) le calcul de $\mathscr{A}$ sur w à partir de q_0. On dit que w est *accepté* par $\mathscr{A}$ si le calcul de $\mathscr{A}$ sur w termine dans un état acceptant. Le *langage accepté* (ou *reconnu*) par $\mathscr{A}$ est l'ensemble des mots $w \in V^*$ acceptés par $\mathscr{A}$. On le note $L(\mathscr{A})$. Autrement dit :

$$L(\mathscr{A}) = \{w \in V^* \mid q_0 \xrightarrow{w} F\}.$$

Pour décrire un automate, il faut d'abord en préciser l'alphabet, l'ensemble des états, l'état initial et les états finaux. Il y a ensuite plusieurs manières de spécifier sa fonction de transition. La plus immédiate, puisque son domaine est fini, c'est d'en donner une description extensive. Ainsi, on pourra considérer un AFD $\mathscr{A} = (V, Q, \delta, q_0, F)$ défini par : $V = \{a,b\}$, $Q = \{p_1, p_2, p_3\}$, $q_0 = p_1$, $F = \{p_3\}$, et dont la fonction de transition δ est donnée explicitement sous la forme :

$$\begin{array}{ll} \delta(p_1,a) = p_2 & \delta(p_1,b) = p_1 \\ \delta(p_2,a) = p_3 & \delta(p_2,b) = p_1 \\ \delta(p_3,a) = p_3 & \delta(p_3,b) = p_1 \end{array}$$

ou encore, avec les conventions d'écriture établies plus haut, sous la forme :

$$\begin{array}{lll} \delta : & p_1 \xrightarrow{a} p_2 & p_1 \xrightarrow{b} p_1 \\ & p_2 \xrightarrow{a} p_3 & p_2 \xrightarrow{b} p_1 \\ & p_3 \xrightarrow{a} p_3 & p_3 \xrightarrow{b} p_1 \end{array}$$

On choisira le plus souvent de condenser ces écritures sous forme d'un tableau, appelé *table de transition* de l'automate :

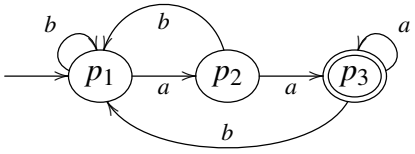
δ	a	b
p_1	p_2	p_1
p_2	p_3	p_1
p_3	p_3	p_1

Dans certains cas, on préfèrera donner la *table de passage* de l'automate, qui précise, pour chaque état, les symboles qui permettent d'accéder aux autres états :

$\vec{\Gamma}$	p_1	p_2	p_3
p_1	b	a	$-$
p_2	b	$-$	a
p_3	b	$-$	a

La flèche située au coin de ce tableau en précise le sens de lecture : ainsi, le symbole situé à l'intersection de la deuxième ligne et de la troisième colonne indique que l'automate peut passer de l'état p_1 à l'état p_2 en lisant la lettre a (*i.e.*, il existe une transition $p_1 \xrightarrow{a} p_2$). La transition inverse, $p_2 \xrightarrow{a} p_1$, n'existe pas.

Enfin, la description que nous utiliserons le plus fréquement passe par la représentation du *graphe* de l'automate : celui-ci s'obtient en dessinant un cercle étiqueté par q pour chaque état q de l'automate, puis en représentant chaque transition $p \xrightarrow{a} q$ par une flèche étiquetée par a joignant les cercles correspondant aux états p et q. On convient de plus de "double-cercler" les états finaux et de pointer d'une flèche sans origine l'état initial. Ce qui donne, pour l'automate pris en exemple plus haut, le graphe suivant :



Cette représentation offre deux avantages : d'abord, elle donne une description exhaustive de l'automate : alphabet, états, fonction de transition, mais aussi état initial et états finaux. Ensuite, elle permet une visualisation très intuitive du fonctionnement de l'automate : pour simuler son calcul sur un mot $w = w_1 \dots w_n$ donné, il suffit de parcourir le graphe, à partir de l'état initial, conformément au "chemin" dicté par la suite de lettres $w_1, \dots, w_n$. Il est facile de voir, par exemple, que le calcul de cet automate sur le mot *babaa* s'écrit :

$$p_1 \xrightarrow{b} p_1 \xrightarrow{a} p_2 \xrightarrow{b} p_1 \xrightarrow{a} p_2 \xrightarrow{a} p_3$$

Du coup, on voit immédiatement que ce mot est accepté par l'automate, puisque p_3 est un état final. Par contre, le calcul de l'automate sur *aaba* donne la suite

$$p_1 \xrightarrow{a} p_2 \xrightarrow{a} p_3 \xrightarrow{b} p_1 \xrightarrow{a} p_2,$$

et ce mot est refusé par l'automate, puisque p_2 n'est pas final. En fait, avec un tout petit peu d'expérience, on se convainc que l'automate ainsi décrit accepte exactement les mots de $\{a,b\}^*$ qui se terminent pas deux a consécutifs.

Pour fixer les idées, terminons par un nouvel exemple : on considère l'automate d'aphabet $V = \{a,b,c\}$, d'ensemble d'états $Q = \{1,2,3,4\}$, d'état initial $q_0 = 1$, d'ensemble d'états acceptants $F = \{4\}$, et dont la fonction de transition δ est donnée en extention par :

$$\begin{array}{lll} \delta(1,a) = 1 & \delta(1,b) = 2 & \delta(1,c) = 1 \\ \delta(2,a) = 3 & \delta(2,b) = 2 & \delta(2,c) = 1 \\ \delta(3,a) = 1 & \delta(3,b) = 2 & \delta(3,c) = 4 \\ \delta(4,a) = 4 & \delta(4,b) = 4 & \delta(4,c) = 4 \end{array}$$

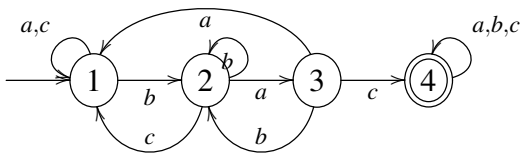
ou encore, par :

$$\begin{array}{lll} \delta : & 1 \xrightarrow{a} 1 & 1 \xrightarrow{b} 2 \quad 1 \xrightarrow{c} 1 \\ & 2 \xrightarrow{a} 3 & 2 \xrightarrow{b} 2 \quad 2 \xrightarrow{c} 1 \\ & 3 \xrightarrow{a} 1 & 3 \xrightarrow{b} 2 \quad 3 \xrightarrow{c} 4 \\ & 4 \xrightarrow{a} 4 & 4 \xrightarrow{b} 4 \quad 4 \xrightarrow{c} 4 \end{array}$$

Alors la table de transition, la table de passage et le graphe de cet automate sont les suivants :

δ	a	b	c
1	1	2	1
2	3	2	1
3	1	2	4
4	4	4	4

$\Gamma^{\rightarrow}$	1	2	3	4
1	ac	b	–	–
2	c	b	a	–
3	a	b	–	c
4	–	–	–	abc



Chapitre 2

Résiduels - Minimisation

2.1 Langages résiduels

Définition 18 (Langage résiduel). *On appelle* résiduel *d'un langage $L \subseteq V^*$ par rapport à un mot $u \in V^*$, le langage : $L/u = \{v \in V^* \text{ tel que } uv \in L\}$. Autrement dit, L/u est l'ensemble des mots de L commençant par u auxquels on a retiré ce préfixe u.*

EXEMPLE. Si $L = aa(bb+c)^*$, alors $L/a = a(bb+c)^*$, $L/b = L/c = \varnothing$.

Lemme 19 *Soient $X, Y \subseteq V^*$, $u, v \in V^*$ et $a \in V$. On a :*

1. $X/uv = (X/u)/v$
2. $(X \cup Y)/u = (X/u) \cup (Y/u)$
3. *si* $\varepsilon \notin X$: $(XY)/a = (X/a)Y$
4. *si* $\varepsilon \in X$: $(XY)/a = (X/a)Y \cup Y/a$
5. $\forall n > 0$: $X^n/a = (X/a)X^{n-1}$
6. $X^*/u = (X/a)X^*$

PREUVE. Exercice. □

Définition 20 (Les langages L_q). *Pour chaque état $q \in Q$ d'un automate $\mathscr{A} = (V, Q, \delta, q_0, F)$, on note L_q l'ensemble des mots w tels que le calcul de $\mathscr{A}$ sur w à partir de q termine dans un état acceptant. Autrement dit : $L_q = \{w \in V^* \mid q \xrightarrow{w} F\}$. En particulier, $L_{q_0} = L(\mathscr{A})$.*

Lemme 21 *Soient $\mathscr{A} = (V, Q, \delta, q_0, F)$ un AFD et $L = L(\mathscr{A})$. Pour tous $u \in V^*$ et $q \in Q$ on a : si $\delta(q_0, u) = q$ alors $L/u = L_q$.*

PREUVE. $x \in L/u$ ssi $ux \in L$ ssi $q_0 \xrightarrow{ux} F$ ssi $q_0 \xrightarrow{u} q \xrightarrow{x} F$ ssi $q \xrightarrow{x} F$ ssi $x \in L_q$. □

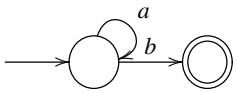
Définition 22 (Automate complet). *Un automate déterministe $\mathscr{A} = (V, Q, \delta, q_0, F)$ est dit* complet *si sa fonction de transition δ est définie sur chaque couple $(p, a) \in Q \times V$.*

Corollaire 23 *Soit $\mathscr{A} = (V, Q, \delta, q_0, F)$ un automate déterministe complet. Alors :*

$$\{L/w,\ w \in V^*\} = \{L_q,\ q \in Q\}.$$

En particulier, le nombre de résiduels de $L(\mathscr{A})$ est fini et inférieur à card (Q).

Remarque 24 *L'hypothèse « $\mathscr{A}$ complet » est impérative, comme en témoigne l'automate suivant sur l'alphabet $\{a, b\}$:*



pour lequel on a card $(Q) = 2$ alors que le langage reconnu, a^*b, admet trois résiduels distincts : a^*b, ε et $\varnothing$.

2.2 Automate des résiduels

Lorsqu'un langage n'a qu'un nombre fini de résiduels, on peut lui associer un automate particulier, appelé « automates des résiduels de L ».

Définition 25 (Automate des résiduels). *Soit $L \subseteq V^*$ un langage tel que $\{L/w, w \in V^*\}$ est fini.* L'automate des résiduels *de L est l'automate déterministe $\mathscr{A}_L = (V, Q, \delta, q_0, F)$ défini par :*

- $Q = \{L/w,\ w \in V^*\}$;
- $\delta(L/w, a) = L/wa$ *pour tous* $w \in V^*$, $a \in V$.
- $q_0 = L$;
- $F = \{L/w,\ w \in L\}$;

Lemme 26 *L'automate des résiduels de L reconnaît L.*

PREUVE. Il est facile de voir que $\delta(q_0,u) = \delta(L,u) = L/u$. Par conséquent, $\delta(q_0,u) \in F$ ssi $L/u \in \{L/w,\ w \in L\}$ ssi $\exists w \in L$ tel que $L/u = L/w$. Puisque nous cherchons à prouver que $u \in L$ ssi $\delta(q_0,u) \in F$, nous sommes ramenés à la question suivante :

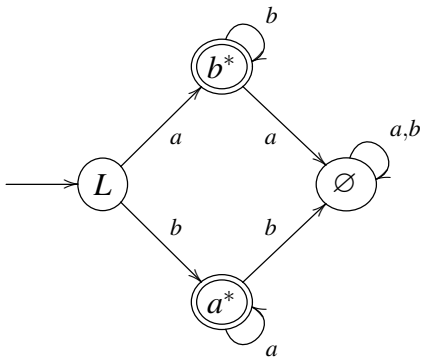
$$\text{A-t-on : } u \in L \Leftrightarrow \exists w \in L \text{ tel que } L/u = L/w \text{ ?}$$

L'implication $\Rightarrow$ est claire (il suffit de prendre $w = u$). Pour l'implication réciproque, rappelons que $L/u = L/w$ signifie : pour tout $x \in V^*$, $ux \in L \Leftrightarrow wx \in L$. En particulier, $w = w\varepsilon \in L$ entraîne $u\varepsilon \in L$, et donc, $u \in L$. Ceci clôt la démonstration. □

EXEMPLE. Les résiduels du langage L dénoté par l'expression régulière $ab^* + ba^*$ sont

1. $L = L/\varepsilon$,
2. $b^* = L/a = L/ab$,
3. $a^* = L/b = L/ba$,
4. $\varnothing = L/aa = L/bb$.

Ce qui donne pour L l'automate des résiduels suivant :



Théorème 27 (Théorème de MYHILL-NERODE). *Un langage est reconnaissable si, et seulement si, il n'a qu'un nombre fini de résiduels.*

PREUVE. Un langage reconnaissable a un nombre fini de résiduels, par le Corollaire 23. Réciproquement, pour tout langage L comportant un nombre fini de résiduels, on peut construire son automate des résiduels, suivant la Définition 25, et cet automate reconnaît L, comme établi au Lemme 26. □

On sait qu'un langage reconnaissable $L \subseteq V^*$ est reconnu non par *un*, mais par une infinité d'automates déterministes. Chacun de ces automates a un nombre

d'états strictement positif. Lorsqu'on construit un automate reconnaissant L, on est particulièrement intéressé à ce qu'il comporte un nombre d'états aussi petit que possible. Pour conclure cette section, nous montrons que l'automate des résiduels satisfait cette requête.

Définition 28 (Automate minimal). *Soit $L \subseteq V^*$ un langage reconnaissable.* L'automate minimal de *L est, parmi les automates déterministes complets reconnaissant L, celui qui minimise le nombre d'états.*

Proposition 29 *L'automate minimal d'un langage reconnaissable est son automate des résiduels.*

PREUVE. Soient L un langage reconnaissable et $\mathscr{A}_L = (V, Q, \delta, q_0, F)$ son automate des résiduels. Alors $Q = \{L/w,\ w \in V\}$ et card (Q) est donc exactement le nombre de résiduels de L. Mais par ailleurs, ce nombre minore le nombre d'états de tout automate déterministe complet qui reconnaît L (Corollaire 23). Donc $\mathscr{A}_L$ est complet (voir Définition 25) et minimal en nombre d'états : c'est l'automate minimal de L. □

2.3 Congruence de Nerode

Cette section et la prochaine sont consacrées au problème suivant : étant donné un langage L, défini par exemple par une expression régulière, nous savons désormais calculer son automate minimal par la méthode des résiduels. Qu'en est-il lorsque le langage est spécifié non pas par une expression régulière, mais par un automate $\mathscr{A}$ qui le reconnaît ? Il est alors généralement difficile d'identifier les résiduels de L sans calculer une expression régulière qui le dénote, ou sans en donner une description plus simple que la seule donnée de l'automate. Revenir à une telle description pour ensuite calculer les résiduels de L, puis l'automate des résiduels, serait très coûteux. Il est préférable de procéder directement, en calculant un automate minimal directement à partir de l'automate initial $\mathscr{A}$. C'est le rôle de l'« algorithme de minimisation », qui sera décrit dans la Section 2.4. Pour l'heure, nous allons établir quelques résultats qui sous-tendent le fonctionnement et la correction dudit algorithme.

Définition 30 (Relation d'équivalence). *Une relation d'équivalence sur un ensemble X est une relation binaire sur X, c'est-à-dire une partie R de $X \times X$, qui est :*

réflexive : xRx *pour tout* $x \in X$;
symétrique : $xRy \Rightarrow yRx$ *pour tous* $x, y \in X$;
transitive : $(xRy$ *et* $yRz) \Rightarrow xRz$ *pour tous* $x, y, z \in X$.

Définition 31 (Congruence). *Soit* $\mathscr{A} = (V, Q, \delta, q_0, F)$ *un automate déterministe. Une relation d'équivalence* $\sim$ *sur* Q *est une* congruence sur $\mathscr{A}$ *si :*

(i) *elle est compatible avec* δ : $(p \sim q) \Rightarrow (\forall a \in V : \delta(p,a) \sim \delta(q,a))$;

(ii) *elle sature* F : $(p \sim q) \Rightarrow (p \in F \Leftrightarrow q \in F)$.

Définition 32 (Automate quotient). *Soit* $\mathscr{A} = (V, Q, \delta, q_0, F)$ *un automate déterministe et* $\sim$ *une congruence sur* $\mathscr{A}$. L'automate quotient de $\mathscr{A}$ par $\sim$ *est noté* $\mathscr{A}/\sim$ *et défini par* $\mathscr{A}/\sim = (V, [Q], [\delta], [q_0], [F])$ *où :*

(i) $[Q] = Q/\sim = \{[q],\ q \in Q\}$;

(ii) $[\delta]([q], a) = [\delta(q,a)]$ *pour tout* $[q] \in [Q]$ *et tout* $a \in V$;

(iii) $[q_0]$ *est la classe de* q_0 *modulo* $\sim$;

(iv) $[F] = F/\sim = \{[q],\ q \in F\}$.

Observons que la définition de $[\delta]$ est cohérente : si p et q sont deux représentants d'une même classe d'équivalence (*i.e.* si $p \sim q$), alors $[\delta(p,a)] = [\delta(q,a)]$, puisque $\sim$ est compatible avec δ (voir Déf. 31(i)). Ceci permet de poser, sans confusion, $[\delta]([q], a) = [\delta(q,a)]$. De même, la définition de $[F]$ est cohérente, puisque pour tout élément p d'une classe $[q]$ (*i.e.* pour tout $p \sim q$), on a $p \in F$ ssi $q \in F$ (voir Déf. 31(ii)).

Lemme 33 *Pour toute congruence* $\sim$ *sur un automate déterministe* $\mathscr{A}$ *on a :*

$$L(\mathscr{A}/\sim) = L(\mathscr{A}).$$

PREUVE. Par définition de $[\delta]$ on a $[p] \xrightarrow{a} [q]$ ssi $p \xrightarrow{a} q$ pour tous $[p], [q] \in [Q]$ et tout $a \in V$. Il s'ensuit que $[q_0] \xrightarrow{u} [q]$ est un calcul de $\mathscr{A}/\sim$ sur le mot u si, et seulement si, $q_0 \xrightarrow{u} q$ est un calcul de $\mathscr{A}$ sur u. Comme de plus $[q] \in [F]$ ssi $q \in F$, on voit facilement que u étiquette un calcul acceptant de $[\mathscr{A}]$ si, et seulement si, il étiquette un calcul acceptant de $\mathscr{A}$. Ainsi, $u \in L([\mathscr{A}])$ ssi $u \in L(\mathscr{A})$, ce qui prouve le résultat attendu. □

Définition 34 (Congruence de NERODE). *Soit* $\mathscr{A} = (V, Q, \delta, q_0, F)$ *un automate déterministe. La* congruence de NERODE de $\mathscr{A}$ *est la relation d'équivalence* $\approx$ *définie sur* Q *par :*

$$p \approx q \text{ si } L_p = L_q.$$

Il est facile de vérifier que $\approx$ est effectivement une congruence sur $\mathscr{A}$. Notons de plus que, par définition des langages L_p, L_q :

$$p \approx q \text{ si } \forall w \in V^* : \delta(p,w) \in F \Leftrightarrow \delta(q,w) \in F$$

Proposition 35 *Soit $\mathscr{A}$ un automate déterministe complet reconnaissant un langage L. L'automate minimal $\mathscr{A}_L$ de L est égal au quotient $\mathscr{A}/\approx$, où $\approx$ est la congruence de Nerode de $\mathscr{A}$.*

PREUVE. Puisque $\approx$ est une congruence sur $\mathscr{A}$, l'automate $\mathscr{A}/\approx$ reconnaît L (Lemme 33). Reste à montrer que l'ensemble $[Q]$ des états de $\mathscr{A}/\approx$ est de cardinal minimal. Or, $[Q] = \{[q],\ q \in Q\}$ et cet ensemble est clairement équipotent à $\{L_q,\ q \in Q\}$ qui, nous l'avons vu, coïncide avec $\{L/u,\ u \in V^*\}$ (Corollaire 23). Par conséquent, card $([Q])$ est égal au nombre de résiduels de L, qui minimise de nombre d'états de tout AFD complet qui reconnaît L (Corollaire 23). D'où le résultat. □

2.4 Minimisation

Les considérations qui précèdent fournissent une stratégie pour construire l'automate minimal de L à partir d'un automate $\mathscr{A} = (V, Q, \delta, q_0, F)$ qui reconnaît L : il nous faut identifier les classes d'équivalence de la congruence de NERODE sur $\mathscr{A}$ puis fabriquer $\mathscr{A}/\approx$. Cette dernière étape est facile : elle se fait essentiellement par fusion les états équivalents de $\mathscr{A}$. L'essentiel de la procédure de minimisation consiste donc à identifier les états équivalents.

Rappelons que $p, q \in Q$ satisfont $p \approx q$ si $L_p = L_q$, c'est à dire si pour tout $u \in V^*$: $u \in L_p \Leftrightarrow u \in L_q$. Considérons maintenant les relations d'équivalence suivantes, définies sur Q par "affaiblissement" de la congruence de NERODE : pour chaque $n \in \mathbb{N}$, on note $V^{\leq n} = \{u \in V^*, |u| \leq n\}$ et on définit la relation $\approx_n$ par : $p \approx_n q$ ssi $L_p \cap V^{\leq n} = L_q \cap V^{\leq n}$ ou, de manière équivalente, par :

$$p \approx_n q \text{ ssi } \forall u \in V^{\leq n} : u \in L_p \Leftrightarrow u \in L_q. \tag{2.1}$$

On exprime alors $\approx_{n+1}$ en fonction de $\approx_n$ en utilisant le fait que $v \in V^{\leq n+1}$ ssi $v \in V^{\leq n}$ ou $v = au$ avec $a \in V$ et $u \in V^{\leq n}$. Par conséquent, $p \approx_{n+1} q$ ssi $\forall a \in V, \forall u \in V^{\leq n} : u \in L_p \Leftrightarrow u \in L_q$ et $au \in L_p \Leftrightarrow au \in L_q$. Or, pour tout p : $au \in L_p \Leftrightarrow u \in$

$L_{\delta(p,a)}$. Il s'ensuit que $p \approx_{n+1} q$ ssi pour tous $a \in V$ et $u \in V^{\leq n}$: $u \in L_p \Leftrightarrow u \in L_q$ et $u \in L_{\delta(p,a)} \Leftrightarrow u \in L_{\delta(q,a)}$. Ceci s'écrit finalement :

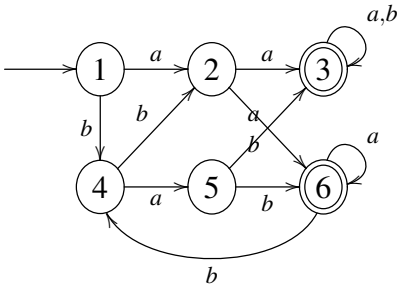
$$p \approx_{n+1} q \text{ ssi } \left(p \approx_n q \text{ et } \forall a \in V : \delta(p,a) \approx_n \delta(q,a)\right). \tag{2.2}$$

Cette dernière équation permet de calculer *pas à pas* les classes d'équivalence modulo $\approx_n$. On commence par identifier les états $\approx_0$-équivalents. L'équation (2.1) donne : $p \approx_0 q$ ssi $\forall u \in V^{\leq 0} : u \in L_p \Leftrightarrow u \in L_q$ ssi $\varepsilon \in L_p \Leftrightarrow \varepsilon \in L_q$. Or $\varepsilon \in L_p \Leftrightarrow p \in F$. D'où : $p \approx_0 q$ ssi $p,q \in F$ ou $p,q \notin F$. Ainsi, $\approx_0$ définit deux classes d'équivalences exactement : F et $Q \setminus F$. À partir de là, on peut déterminer les classes d'équivalence modulo $\approx_1$: deux états satisfont $p \approx_1 q$ si $p \approx_0 q$ et si pour tout $a \in V$, $\delta(p,a) \approx_0 \delta(q,a)$. Autrement dit, les classes de $\approx_1$ sont obtenues en séparant, dans chaque classe de $\approx_0$, les états qui sont "envoyés" par une même lettre sur des $\approx_0$-classes différentes. Ce même raisonnement permet de calculer $\approx_2$ à partir de $\approx_1$, puis, de proche en proche, toutes les relations $\approx_n$. Le point clé, pour obtenir la relation de NERODE initiale, c'est de constater qu'il existe nécessairement un entier n tel que $\approx_n$ et $\approx$ coïncident.

Proposition 36 *Il existe $n \in \mathbb{N}$ tel que pour tout $i \geq 0$: $\approx_n = \approx_{n+i} = \approx$.*

PREUVE.Exercice. □

En guise d'exemple, faisons tourner l'algorithme sur l'automate $\mathscr{A}$ suivant :



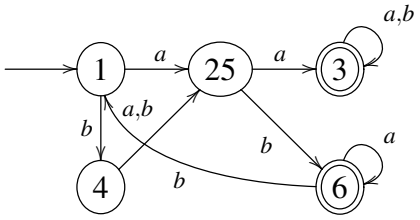
$\approx_0$: $\{1,2,4,5\}$; $\{3,6\}$.

$\approx_1$: $\{1,4\}$; $\{2,5\}$; $\{3\}$; $\{6\}$.
$\{1,4\}$ et $\{2,5\}$ sont séparés par a ;
$\{3\}$ et $\{6\}$ sont séparés par b.

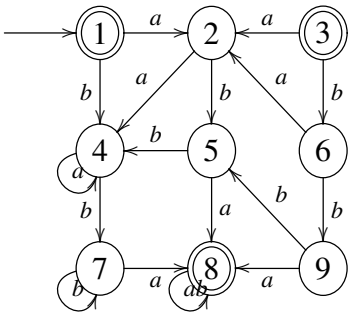
$\approx_2$: $\{1\}$; $\{4\}$; $\{2,5\}$; $\{3\}$; $\{6\}$.
$\{1\}$ et $\{4\}$ sont séparés par b.

$\approx_3$: $\approx_3 = \approx_2$, d'où $\approx_2 = \approx$.

L'automate quotient $\mathscr{A}/\approx$ s'obtient alors, conformément à la Définition 32, en fusionnant les états équivalents et en transformant toute transition $p \xrightarrow{a} q$ de l'automate initial en une transition $[p] \xrightarrow{a} [q]$ de l'automate quotient. Rappelons de plus que l'état initial $\mathscr{A}/\approx$ est la classe d'équivalence de l'état initial de $\mathscr{A}$ et que les états finaux de $\mathscr{A}/\approx$ sont les classes d'équivalence des états finaux de $\mathscr{A}$. On obtient ainsi pour $\mathscr{A}/\approx$:



Considérons un nouvel exemple, avec ce deuxième automate :



$\approx_0$: $\{2,4,5,6,7,9\}$; $\{1,3,8\}$.

$\approx_1$: $\{2,4,6\}$; $\{5,7,9\}$; $\{1,3\}$; $\{8\}$.
$\{2,4,6\}$ et $\{5,7,9\}$ sont séparés par a ;
$\{1,3\}$ et $\{8\}$ sont séparés par a.

$\approx_2$: $\{2,4,6\}$; $\{5\}$; $\{7,9\}$; $\{1,3\}$; $\{8\}$.
$\{5\}$ et $\{7,9\}$ sont séparés par b.

$\approx_3$: $\{2\}$; $\{4,6\}$; $\{5\}$; $\{7\}$; $\{9\}$; $\{1,3\}$; $\{8\}$.
$\{2\}$ et $\{4,6\}$ sont séparés par b ;
$\{7\}$ et $\{9\}$ sont séparés par b.

$\approx_4$: $\{2\}$; $\{4\}$; $\{6\}$; $\{5\}$; $\{7\}$; $\{9\}$; $\{1,3\}$; $\{8\}$.
$\{4\}$ et $\{6\}$ sont séparés par a.

$\approx_5$: $\{2\}$; $\{4\}$; $\{6\}$; $\{5\}$; $\{7\}$; $\{9\}$; $\{1\}$; $\{3\}$; $\{8\}$.
$\{1\}$ et $\{3\}$ sont séparés par b.

La relation $\approx_5$ ne peut plus être raffinée : elle coïncide nécessairement avec la relation $\approx$. Les états de l'automate initial sont deux à deux non équivalents, ce qui prouve que l'automate en question était déjà minimal.

Chapitre 3

Le Théorème de Kleene

3.1 Automates finis non déterministes

Nous allons considérer dans cette section une généralisation des automates finis. Cette généralisation s'éloigne de l'intuition élaborée précédement de l'automate vu comme une formalisation simplifiée d'ordinateur réels, parce qu'elle comporte une part de non déterminisme, incompatible avec le comportement d'un programme. Pourtant, les *automates finis non déterministes* que nous allons introduire (AFN en abrégé) constituent un outil très efficace de description des langages formels, et nous ne tarderons pas à en mesurer l'utilité.

Informellement, les automates non déterministes sont obtenus en enrichissant le fonctionnement des AFD de deux nouvelles capacités :

- on autorise des transitions sur le mot vide (*i.e.*, qui se font sans déplacement de la tête de lecture sur le mot d'entrée),
- on permet qu'à une lettre et un état fixés correspondent plusieurs transitions. Ainsi, lorsqu'il est dans un état p et qu'il lit une lettre a, l'automate peut choisir de manière non déterministe d'évoluer vers différents nouveaux états.

Formellement, les définitions des AFN et des AFD sont similaires : dans les deux cas, il faut préciser un alphabet, un ensemble d'états, une fonction de transition, un état initial et un ensemble d'états finaux. Elles diffèrent cependant dans la nature de la fonction de transition : dans un automate déterministe, la fonction de transition associe à tout couple constitué d'un état et d'une lettre, un unique état $q \in Q$ accessible en une transition. Dans un automate non déterministe, la fonction prend un couple constitué d'un état et d'une lettre *ou du mot vide* et lui associe un

ensemble d'états $q \subseteq Q$ accessibles en une transition. Autrement dit, la fonction de transition d'un tel automate a pour prototype : $\delta : Q \times (V \cup \{\varepsilon\}) \to \mathscr{P}(Q)$. Notant $V_\varepsilon = V \cup \{\varepsilon\}$, on peut finalement écrire :

Un *automate fini non déterministe* est un quintuplet (V, Q, δ, q_0, F), où :

1. V est un alphabet,
2. Q est l'ensemble des *états* de l'automate,
3. $\delta : Q \times V_\varepsilon \to \mathscr{P}(Q)$ est la *fonction de transition* de l'automate,
4. $q_0 \in Q$ est l'*état initial*, et
5. F est l'ensemble des *états acceptant* ou *finaux*.

Les représentations des AFN prolongent de manière naturelle celles que nous avions détaillées pour les AFD. Ainsi, la fonction de transition de l'automate non déterministe $\mathscr{A} = (V, Q, \delta, q_0, F)$ défini par : $Q = \{q_1, q_2, q_3, q_4\}$, $V = \{0, 1\}$, $q_0 = q_1$, $F = \{q_4\}$ et

$$\begin{array}{lll} \delta(q_1,0) = \{q_1\} & \delta(q_1,1) = \{q_1,q_2\} & \delta(q_1,\varepsilon) = \varnothing \\ \delta(q_2,0) = \{q_3\} & \delta(q_2,1) = \varnothing & \delta(q_2,\varepsilon) = \{q_3\} \\ \delta(q_3,0) = \varnothing & \delta(q_3,1) = \{q_4\} & \delta(q_3,\varepsilon) = \varnothing \\ \delta(q_4,0) = \{q_4\} & \delta(q_4,1) = \{q_4\} & \delta(q_4,\varepsilon) = \varnothing \end{array}$$

pourra s'écrire :

$$\begin{array}{lll} \delta : \quad q_1 \xrightarrow{0} \{q_1\} & q_1 \xrightarrow{1} \{q_1,q_2\} & q_1 \xrightarrow{\varepsilon} \varnothing \\ \quad q_2 \xrightarrow{0} \{q_3\} & q_2 \xrightarrow{1} \varnothing & q_2 \xrightarrow{\varepsilon} \{q_3\} \\ \quad q_3 \xrightarrow{0} \varnothing & q_3 \xrightarrow{1} \{q_4\} & q_3 \xrightarrow{\varepsilon} \varnothing \\ \quad q_4 \xrightarrow{0} \{q_4\} & q_4 \xrightarrow{1} \{q_4\} & q_4 \xrightarrow{\varepsilon} \varnothing \end{array}$$

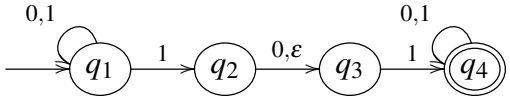
Ou encore, *via* la table de transition et la table de passage de l'automate :

δ	0	1	ε
q_1	$\{q_1\}$	$\{q_1,q_2\}$	$\varnothing$
q_2	$\{q_3\}$	$\varnothing$	$\{q_3\}$
q_3	$\varnothing$	$\{q_4\}$	$\varnothing$
q_4	$\{q_4\}$	$\{q_4\}$	$\varnothing$

ou

$\vec{\Gamma}$	q_1	q_2	q_3	q_4
q_1	$0,1$	1	$-$	$-$
q_2	$-$	$-$	$0,\varepsilon$	$-$
q_3	$-$	$-$	$-$	1
q_4	$-$	$-$	$-$	$0,1$

Enfin, nous utiliserons largement la représentation par le graphe de l'automate :



Une suite d'états $p_1,\ldots,p_n$ est un *calcul* de $\mathscr{A}$ sur w à partir de p si $p_1 = p$ et s'il existe une suite de symboles $x_1,x_2,\ldots,x_n$ dans V_ε telle que $w = x_1\ldots x_n$, et

$$p_1 \xrightarrow{x_1} p_2 \xrightarrow{x_2} \ldots \xrightarrow{x_n} p_n.$$

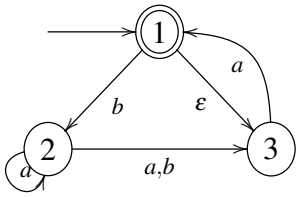
On dit alors que le calcul considéré *termine* dans l'état p_n. Notons que le fait marquant c'est la possibilité pour un AFN, d'admettre *plusieurs* calculs sur w à partir de p.

Un *calcul de $\mathscr{A}$ sur w* est un calcul de $\mathscr{A}$ sur w à partir de q_0. Un tel calcul est dit *acceptant* s'il termine dans un état acceptant. La définition des AFN permettant une multiplicité de calculs sur un mot fixé, il peut se produire en particulier que certains calculs sur w soient acceptants alors que d'autres ne le sont pas. D'où l'importance de la définition qui suit :

On dit que w est *accepté* par $\mathscr{A}$ s'il existe un calcul de $\mathscr{A}$ sur w qui termine dans un état acceptant. Le *langage accepté* (ou *reconnu*) par $\mathscr{A}$ est l'ensemble des mots $w \in V^*$ acceptés par $\mathscr{A}$. On le note $L(\mathscr{A})$. Autrement dit, $L(\mathscr{A})$ est le langage :

$$\{w \in V^* \mid \text{ il existe un calcul acceptant de } \mathscr{A} \text{ sur } w.\}.$$

EXEMPLE. Considérons l'automate $\mathscr{A}$ suivant :



Le mot *baaa* admet les calculs suivant, selon $\mathscr{A}$:

$1 \xrightarrow{b} 2 \xrightarrow{a} 2 \xrightarrow{a} 2 \xrightarrow{a} 2,$

$$1 \xrightarrow{b} 2 \xrightarrow{a} 2 \xrightarrow{a} 2 \xrightarrow{a} 3,$$
$$1 \xrightarrow{b} 2 \xrightarrow{a} 2 \xrightarrow{a} 3 \xrightarrow{a} 1.$$

Les deux premiers terminent dans un état non acceptant. Par contre, le dernier est un calcul acceptant et cela suffit pour affirmer que $baaa \in L(\mathscr{A})$.

3.2 Equivalence des AFN et des AFD

Dans cette section, nous montrons que les automates non déterministes reconnaissent les mêmes langages que leurs analogues déterministes. Cette équivalence est suprenante de prime abord : on aurait pu penser que les capacités supplémentaires dont bénéficient les AFN leur permettraient de reconnaitre une plus large classe de langages. Tel n'est en fait pas le cas. Mais c'est précisément la « robustesse » de ce concept d'automate (c'est-à-dire, la possibilité de modifier sensiblement la formulation du concept sans modifier l'objet qu'il recouvre) qui fonde la pertinence de ce modèle de calcul.

Rappelons que deux automates sont *équivalents* s'ils reconnaissent le même langage.

Théorème 37 *Tout automate non déterministe est équivalent à un automate déterministe.*

ELÉMENTS DE PREUVE. Nous devons montrer que pour chaque automate non déterministe $\mathscr{N} = (V, Q, \delta, q_0, F)$ il existe un automate déterministe $\mathscr{D}$ qui reconnait le même langage. La construction procède en deux étapes :

Dans un premier temps, on construit un AFN équivalent à $\mathscr{N}$ ne comportant plus d'ε-transitions. L'idée, pour ce faire, est de supprimer toutes les ε-transitions et d'ajouter une transition $p \xrightarrow{a} q$, pour $a \in V$, à chaque fois qu'il existe un chemin de p vers q dans le graphe de l'automate dont toutes les arêtes sont étiquetées par ε, sauf une, qui est étiquetée par a. Formellement, on introduit la notion d'*ε-clôture* d'un état $q \in Q$ comme étant l'ensemble $\mathscr{E}(q)$ des états accessibles à partir de q par une suite d'ε-transitions. En d'autres termes, $\mathscr{E}(q)$ est le plus petit ensemble d'états qui contient q et tel que $p \in \mathscr{E}(q)$ et $p \xrightarrow{\varepsilon} p'$ entraînent $p' \in \mathscr{E}(q)$.

On définit un nouvel AFN $\mathscr{N}' = (V, Q, q_0, \delta', F')$ sans ε-transitions en posant :

$$F' = \{p \in Q \,|\, \mathscr{E}(p) \cap F \neq \varnothing\}$$

et pour tout $(p,a) \in Q \times V$:

$$\delta'(p,a) = \bigcup_{q \in \mathscr{E}(p)} \{\mathscr{E}(r) \,|\, r \in \delta(q,a)\}$$

Autrement dit, il y a une transition $p \xrightarrow{a} p'$ dans $\mathscr{N}'$ ssi il existe deux états q et r tels que, pour l'automate $\mathscr{N}$: q est accessible à partir de p par une suite d'ε-transitions ; $q \xrightarrow[\mathscr{N}]{a} r$ et p' est accessible à partir de r par une suite d'ε-transitions. On peut alors montrer facilement que $\mathscr{N}'$ est équivalent à $\mathscr{N}$;

Reste à éliminer le non déterminisme. Ce qui précède permet de supposer sans perte de généralité que $\mathscr{N}$ ne comporte pas d'ε-transitions. Comment traiter les transitions non déterministes, du type $p \xrightarrow{a} \{p_1, \ldots, p_k\}$? L'idée est de construire un nouvel automate dans lequel l'ensemble $\{p_1, \ldots, p_k\}$ représente un unique état. Autrement dit, dans ce nouvel automate, l'ensemble des états est $\mathscr{P}(Q)$. Pour simuler toutes les transitions de $\mathscr{N}$, il faudra associer à l'état $\{p_1, \ldots, p_k\}$ du nouvel automate l'ensemble des états accessibles à partir de n'importe lequel des p_i. Plus formellement, notons $\mathscr{D} = (V, Q_\mathscr{D}, \delta_\mathscr{D}, q_0, F_\mathscr{D})$ l'automate défini par :

$$Q_\mathscr{D} = \mathscr{P}(Q), F_\mathscr{D} = \{q \in \mathscr{P}(Q) \,|\, q \cap F \neq \varnothing\},$$

et pour tout $(q,a) \in Q_\mathscr{D} \times V$:

$$\delta_\mathscr{D}(q,a) = \bigcup_{q \in q} \delta(q,a).$$

On montrerait sans difficulté que cet automate – clairement déterministe, au vu du prototype de sa fonction de transition – est équivalent à $\mathscr{N}$.

Nous verrons en TD que derrière cette idée de preuve se cache un algorithme simple permettant de construire effectivement un AFD équivalent à un AFN donné. □

Il va de soi que la réciproque du Théorème 37 est vraie : les AFD étant des cas particuliers d'AFN, tout langage reconnu par un AFD est aussi reconnu par un AFN. Il s'ensuit que la classe REC peut être vue indifféremment comme la classe des langages reconnus par un AFD ou par un AFN. Ce fait nous permettra de prouver un résultat fondamental du cours : l'identité des classes REC et REG. Dans cette perspectives, commençons par prouver quelques propriétés de clôture de la classe REC.

Proposition 38 *L'ensemble REC est clos pour l'union, la concaténation des langages et la fermeture de Kleene. Autrement dit, si L_1 et L_2 sont reconnaissables, alors $L_1 \cup L_2$, $L_1 L_2$ et L_1^* sont reconnaissables.*

PREUVE. Soient donc L_1, L_2 deux langages sur un alphabet V, reconnus, respectivement, par des automates finis $\mathscr{A}_1 = (V, Q_1, \delta_1, s_1, F_1)$ et $\mathscr{A}_2 = (V, Q_2, \delta_2, s_2, F_2)$. Nous allons construire, à partir de $\mathscr{A}_1$ et $\mathscr{A}_2$, des automates reconnaissant les langages $L_1 \cup L_2$, $L_1 L_2$ et L_1^*, prouvant ainsi l'appartenance de ces trois nouveaux langages à REC. Représentons ces automates en mettant uniquement en évidence leur état initial et leurs états finaux :

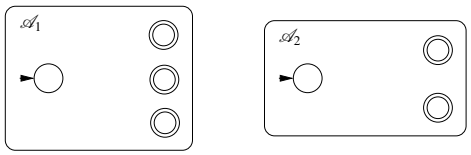


FIGURE 3.1 – Des automates reconnaissant L_1 et L_2.

On obtient alors un AFN reconnaissant l'union $L_1 \cup L_2$ en ajoutant un nouvel état, que l'on décrète initial, relié aux anciens états initiaux de $\mathscr{A}_1$ et $\mathscr{A}_2$ par des ε-transitions. Cette construction est illustrée Figure 3.2.

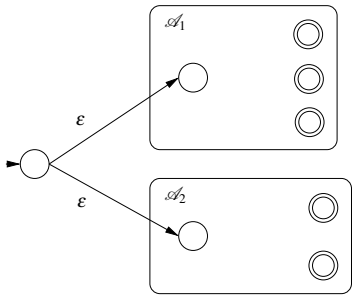


FIGURE 3.2 – Un AFN pour $L_1 \cup L_2$.

Formellement, ce nouvel automate peut être défini comme suit : $\mathscr{A} = (V, Q, \delta, s, F)$

où s est un nouvel état, $Q = Q_1 \cup Q_2 \cup \{s\}$, $F = F_1 \cup F_2$, et δ est donnée par :

$$\forall (q,a) \in Q \times V_\varepsilon \ : \ \delta(q,a) = \begin{cases} \delta_1(q,a) & \text{si } q \in Q_1 \\ \delta_2(q,a) & \text{si } q \in Q_2 \\ \{s_1, s_2\} & \text{si } q = s \text{ et } a = \varepsilon \\ \varnothing & \text{si } q = s \text{ et } a \in V \end{cases}$$

Il est alors facile de montrer qu'un mot w est accepté par $\mathscr{A}$ ssi il existe un calcul de la forme $s \xrightarrow[\mathscr{A}]{\varepsilon} s' \xrightarrow[\mathscr{A}]{w} q$ avec $s' \in \{s_1, s_2\}$ et $q \in F$, ou de manière équivalente, ssi il existe un calcul de l'une des formes $s_1 \xrightarrow[\mathscr{A}_1]{w} q$ avec $q \in F_1$ ou $s_2 \xrightarrow[\mathscr{A}_2]{w} q$ avec $q \in F_2$. Autrement dit, w est accepté par $\mathscr{A}$ si, et seulement si, il est accepté par $\mathscr{A}_1$ ou par $\mathscr{A}_2$. Ce qui montre bien que $L(\mathscr{A}) = L_1 \cup L_2$.

Suivant une méthode similaire, on construit un AFN reconnaissant $L_1 L_2$ en supprimant le statut d'état acceptant des anciens états finaux de $\mathscr{A}_1$ et en ajoutant une ε-transition de chacun de ces état vers l'ancien état initial de $\mathscr{A}_2$.

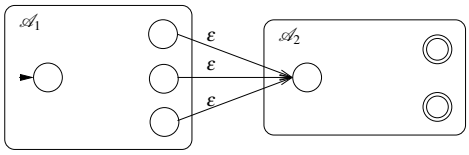


FIGURE 3.3 – Un AFN pour $L_1 L_2$.

Enfin, un AFN reconnaissant L_1^* est obtenu à partir de $\mathscr{A}_1$ en ajoutant un nouvel état, qui devient l'état initial puis en ajoutant des ε-transitions partant de cet état et de chacun des états finaux vers l'ancien état initial.

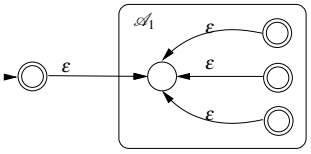


FIGURE 3.4 – Un AFN pour L_1^*.

La preuve formelle que ces deux dernières constructions sont correctes (*i.e.*, réalisent bien ce qu'elles prétendent réaliser) est laissée au lecteur. □

Corollaire 39 *Tout langage régulier est reconnaissable par un automate fini. Autrement dit : REG $\subseteq$ REC.*

PREUVE. Il s'agit de montrer l'inclusion REG $\subseteq$ REC. Compte tenu de la définition inductive de REG (voir Définition 12), il suffit de prouver que :

(i) REC contient les langages $\varnothing$, $\{\varepsilon\}$ et $\{a\}$ (pour chaque lettre a) ;

(ii) REC est clos pour l'union, la concaténation des langages et la fermeture de Kleene.

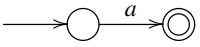
L'assertion (ii) vient précisément d'être établie par le Théorème 38. Reste donc à prouver (i). Le résultat est immédiat : $\varnothing$ est reconnu par l'automate réduit à un unique état initial et non acceptant ; $\{\varepsilon\}$ est reconnu par l'automate réduit à un unique état initial et acceptant ; pour chaque lettre a, $\{a\}$ est reconnu par l'automate à deux états, l'un initial, l'autre acceptant, admettant une unique transition étiquetée par a et joignant l'état initial à l'état final. □

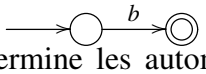
3.3 Des expressions régulières aux automates

Avant d'établir la réciproque du Corollaire 39, nous allons montrer que la preuve de la Proposition 38 induit un algorithme simple pour construire l'automate reconnaissant le langage dénoté par une expression régulière fixée. Commençons par illustrer la méthode sur un exemple. Soit e l'expression régulière, sur l'alphabet $\{a,b\}$:

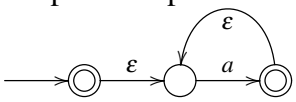
$$e = a^*b + b^*a.$$

Pour construire un automate équivalent à e (c-à-d reconnaissant le langage dénoté par e), on commence par construire les automates équivalents aux expressions régulière atomiques intervenant dans e (ici, a et b). Ce qui donne :

AF pour a : 

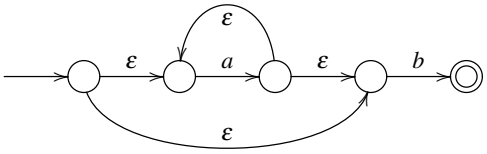
AF pour b : 

Puis on détermine les automates équivalents aux "briques" suivantes de e, en utilisant les modalitées de construction décrites dans la preuve de la Proposition 38. On obtient par exemple ici :

AF pour a^* : 

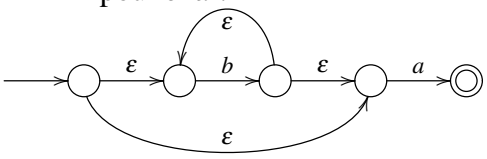
Et donc :

AF pour a^*b :



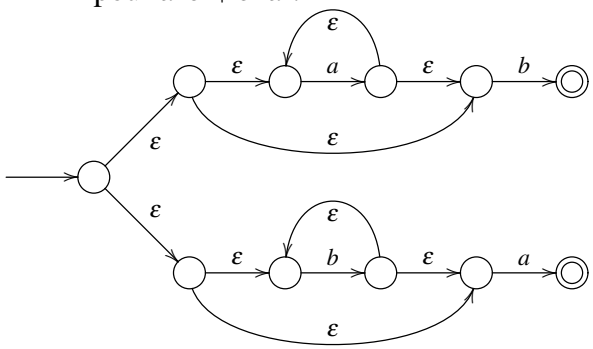
On a de même, par symmétrie :

AF pour b^*a :

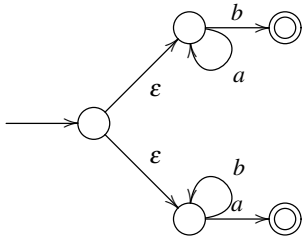


Et on obtient finalement, avec la règle de construction de l'automate associé à l'union de deux langages :

AF pour $a^*b + b^*a$:



Notons que l'expression régulière e est aussi équivalente à l'automate :



Ceci témoigne du fait que l'algorithme utilisé plus haut ne garantit aucune forme de "minimalité" de l'automate calculé. Mais du moins sommes nous assurés de la correction de cet algorithme et de sa capacité, quelle que soit la complexité de l'expression régulière prise en entrée, à fournir un automate équivalent.

3.4 Des automates aux expressions régulières

Le but de cette section est de présenter la construction "inverse" de celle qui précède : partant d'un automate (déterministe), nous allons construire une expression régulière équivalente. La possibilité d'effectuer cette construction pour n'importe quel AFD aura pour conséquence immédiate l'inclusion de la classe REC dans la classe REG.

Pour commencer, il nous faut définir la notion de système d'équations associé à un automate. Soit $\mathscr{A} = (V, Q, \delta, q_0, F)$ un automate fini déterministe. Rappelons que pour un état $p \in Q$, on note L_p le langage :

$$L_p = \{w \in V^* \mid p \xrightarrow{w} q \text{ avec } q \in F\}.$$

Autrement dit, L_p est l'ensemble des mots w tel que le calcul de $\mathscr{A}$ sur w à partir de p est acceptant. On peut aussi remarquer que L_p serait le langage reconnu par $\mathscr{A}$ si p était son état initial. En particulier, L_{q_0} (où q_0 est l'état initial) est le langage reconnu par l'automate. Supposons maintenant que $V = \{a, b\}$ et que les transitions partant d'un état p donné soient : $p \xrightarrow{a} q$ et $p \xrightarrow{b} r$. Alors si p est non final, un mot w admet un calcul acceptant à partir de p si :

- soit $w = au$, où u est un mot de V^* admettant un calcul acceptant à partir de q,
- soit $w = bu$, où u est un mot de V^* admettant un calcul acceptant à partir de r.

Par conséquent, on a :

$$L_p = aL_q \cup bL_r.$$

Si p est acceptant, on montre tout aussi facilement qu'un mot w est dans L_p si et seulement si $w = \varepsilon$ ou $w \in aL_q \cup bL_r$. Par conséquent, on a dans ce cas :

$$L_p = aL_q \cup bL_r \cup \{\varepsilon\}.$$

Notant X_p une expression régulière qui dénote L_p, les équations qui précèdent donnent lieu aux équations suivantes, entre expressions régulières :

$$\begin{array}{ll} X_p = aX_q + bX_r & \text{si } p \notin F \\ X_p = aX_q + bX_r + \varepsilon & \text{si } p \in F \end{array}$$

Ceci se généralise facilement : étant donnés un AFD $\mathscr{A} = (V, Q, \delta, q_0, F)$ et $p \in Q$, si les transitions d'origine p s'écrivent $p \xrightarrow{a_1} p_1$, ..., $p \xrightarrow{a_k} p_k$, où $p_1, \ldots, p_k \in$

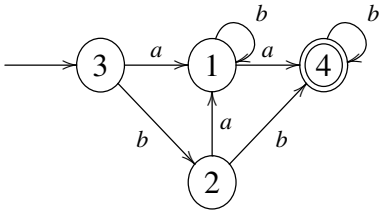
Q et $V = \{a_1, \ldots, a_k\}$, alors les expressions régulières qui dénotent les langages $L_p, L_{p_1}, \ldots, L_{p_k}$ sont liées par l'équation :

$$X_p = a_1 X_{p_1} + \cdots + a_k X_{p_k} \text{ si } p \notin F,$$

ou par l'équation

$$X_p = a_1 X_{p_1} + \cdots + a_k X_{p_k} + \varepsilon \text{ si } p \in F.$$

Le *système d'équations associé à* $\mathscr{A}$ est le système obtenu en écrivant les équations correspondant à chaque état $p \in Q$.
EXEMPLE. L'automate suivant :



est associé au système d'équations :

$$\begin{cases} X_1 = bX_1 + aX_4 \\ X_2 = aX_1 + bX_4 \\ X_3 = aX_1 + bX_2 \\ X_4 = bX_4 + \varepsilon \end{cases}$$

Le point clé, dans la recherche de l'expression régulière associée à un automate $\mathscr{A}$, c'est que le système d'équations associé admet toujours une solution $(X_1, \ldots, X_k)$, et que cette solution se calcule facilement.

Lemme 40 (Lemme d'ARDEN). *Soient A, B deux langages sur V tels que $\varepsilon \notin A$. Alors l'équation*

$$X = AX \cup B \tag{3.1}$$

admet A^*B comme unique solution.

PREUVE. D'abord, A^*B est solution de (3.1) puisque $A(A^*B) \cup B = A^+B \cup B = A^*B$. Ensuite, c'est la *plus petite* des solutions : si X vérifie (3.1), une récurrence simple permet d'établir :

$$\forall n \geq 0 : X = A^{n+1}X \cup A^nB \cup \cdots \cup AB \cup B \tag{3.2}$$

Il s'ensuit que $A^nB \subset X$ pour tout n et donc que $A^*B = \bigcup_{n\geq 0} A^nB \subseteq X$.

Enfin, toute solution X de (3.1) est incluse dans A^*B. Considérons $w \in X$ et notons $n = |w|$. Par (3.2), $w \in A^{n+1}X \cup A^nB \cup \cdots \cup B$. Comme les mots de $A^{n+1}X$ sont de longueur $> n$ (puisque $\varepsilon \notin A$), on a $w \in A^nB \cup \cdots \cup AB \cup B \subseteq A^*B$. □

Les méthodes de substitutions usuelles en algèbre permettent alors d'appliquer ce lemme à la résolution de systèmes d'équations :

Corollaire 41 *On considère des langages régulier A_i^j, B_i, tels qu'aucun A_i^j ne contienne ε. Alors le système d'équations :*

$$\begin{cases} X_1 = A_1^1 X_1 \cup \cdots \cup A_n^1 X_n \cup B_1 \\ \qquad\qquad \vdots \\ X_n = A_1^n X_1 \cup \cdots \cup A_n^n X_n \cup B_n \end{cases}$$

admet une unique solution, qui est un n-uplet $(L_1, \ldots, L_n)$ de langages réguliers.

Ainsi, pour le système évoqué plus haut :

$$\begin{cases} X_1 = bX_1 + aX_4 & (1) \\ X_2 = aX_1 + bX_4 & (2) \\ X_3 = aX_1 + bX_2 & (3) \\ X_4 = bX_4 + \varepsilon & (4) \end{cases}$$

le jeu des substitutions permet d'obtenir :

$$\begin{aligned} \text{par } (4): \quad & X_4 = b^*\varepsilon = b^* && (5) \\ \text{par } (1,5): \quad & X_1 = b^*aX_4 = b^*ab^* && (6) \\ \text{par } (5,6): \quad & X_3 = ab^*ab^* + bb^* && (7) \end{aligned}$$

Or le langage reconnu par l'automate est l'ensemble des mots qui admettent un calcul acceptant à partir de l'état initial. Autrement dit, $L(\mathscr{A}) = L_3$ et l'automate est équivalent à l'expression régulière qui dénote L_3, c'est-à-dire à $ab^*ab^* + bb^*$.

Ce procédé se généralise de la façon suivante : soit $\mathscr{A}$ un automate déterministe (V, Q, δ, q_0, F). On suppose que $V = \{a_1, \ldots, a_k\}$ et $Q = \{1, \ldots, n\}$, et on écrit les équations $(E_1), \ldots, (E_n)$ associé aux états de $\mathscr{A}$:

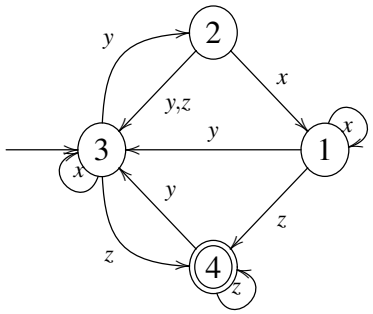
$$\begin{cases} X_1 = \sum_{i=1}^k a_i X_{\delta(1,a_i)} + \lambda_1 & (E_1) \\ \vdots \\ X_n = \sum_{i=1}^k a_i X_{\delta(n,a_i)} + \lambda_n & (E_n) \end{cases}$$

où $\lambda_i = \varepsilon$ si l'état i est acceptant, $\varnothing$ sinon. On résoud alors le système en appliquant l'algorithme suivant :

---

1 **pour** $i = 1, \ldots, n$ **faire**
2 | **si** X_i *apparait dans le membre droit de l'équation* (E_i) **alors**
3 | | Ecrire (E_i) sous la forme $X_i = \alpha X_i + \beta$;
4 | | Remplacer (E_i) par l'équation $X_i = \alpha^* \beta$;
| | **fin**
5 | Remplacer chaque occurence de X_i par le membre droit de l'équation (E_i) dans les équations $(E_{i+1}), \ldots, (E_n)$;
fin
/ A ce stade de l'algorithme, chaque équation* (E_i) *exprime* X_i *en fonction de* $X_{i+1}, \ldots, X_n$. *En particulier,* (E_n) *permet d'exprimer* X_n *comme une expression régulière constante. */* ;
6 **pour** $i = n-1, \ldots, 1$ **faire**
7 | Calculer X_i en fonction des valeurs de $X_{i+1}, \ldots, X_n$;
fin
8 **retourner** (X_j), où j est l'état initial.

EXEMPLE. Considérons l'automate déterministe $\mathscr{A}$ suivant :



On lui associe le système d'équations :

$$\begin{aligned} X_1 &= xX_1 + yX_3 + zX_4 \\ X_2 &= xX_1 + yX_3 + zX_3 \\ X_3 &= xX_3 + yX_2 + zX_4 \\ X_4 &= yX_3 + zX_4 + \varepsilon \end{aligned}$$

L'exécution de la boucle **pour** de la ligne 1 pour $i = 1$ donne :

$$\begin{aligned} X_1 &= x^*(yX_3 + zX_4) \\ X_2 &= xx^*(yX_3 + zX_4) + yX_3 + zX_3 \\ X_3 &= xX_3 + yX_2 + zX_4 \\ X_4 &= yX_3 + zX_4 + \varepsilon \end{aligned}$$

L'exécution de la boucle **pour** de la ligne 1 pour $i = 2$ donne :

$$\begin{aligned} X_1 &= x^*(yX_3 + zX_4) \\ X_2 &= xx^*(yX_3 + zX_4) + yX_3 + zX_3 \\ &= (x^+y + y + z)X_3 + x^+zX_4 \\ &= (x^*y + z)X_3 + x^+zX_4 \\ X_3 &= xX_3 + y((x^*y + z)X_3 + x^+zX_4) + zX_4 \\ X_4 &= yX_3 + zX_4 + \varepsilon \end{aligned}$$

L'exécution de la boucle **pour** de la ligne 1 pour $i = 3$ donne :

$$\begin{aligned} X_1 &= x^*(yX_3 + zX_4) \\ X_2 &= (x^*y + z)X_3 + x^+zX_4 \\ X_3 &= xX_3 + y((x^*y + z)X_3 + x^+zX_4) + zX_4 \\ &= (x + y(x^*y + z))X_3 + (yx^+z + z)X_4 \\ &= (x + y(x^*y + z))^*(yx^+z + z)X_4) \\ X_4 &= y(x + y(x^*y + z))^*(yx^+z + z)X_4 + zX_4 + \varepsilon \end{aligned}$$

L'exécution de la boucle **pour** de la ligne 1 pour $i = 4$ donne :

$$\begin{aligned} X_1 &= x^*(yX_3 + zX_4) \\ X_2 &= (x^*y + z)X_3 + x^+zX_4 \\ X_3 &= (x + y(x^*y + z))^*(yx^+z + z)X_4) \\ X_4 &= y(x + y(x^*y + z))^*(yx^+z + z)X_4 + zX_4 + \varepsilon \\ &= (y(x + y(x^*y + z))^*(yx^+z + z) + z)X_4 + \varepsilon \\ &= (y(x + y(x^*y + z))^*(yx^+z + z) + z)^* \end{aligned}$$

On obtient ainsi une valeur constante pour l'expression régulière X_4 :

$$X_4 = (y(x + y(x^*y + z))^*(yx^+z + z) + z)^*.$$

Par ailleurs, chaque X_i, pour $i < 4$, s'exprime en fonction de $X_{i+1}, \ldots, X_4$. On va donc pouvoir, en "remontant" de X_4 à X_1 calculer les valeurs de chaque expression régulière inconnue. C'est le but de la boucle **pour** de la ligne 6. L'exécution de

cette boucle pour $i = 3$ donne : $X_3 = (x + y(x^*y + z))^*(yx^+z + z)(y(x + y(x^*y + z))^*(yx^+z + z) + z)^*$. On pourrait de même calculer les valeurs de X_2 et X_1 en fonction de celles X_3 et X_4. Ce sera inutile ici : X_3 est l'expression équivalent à l'automate, puisque 3 est l'état initial. Ainsi, $L(\mathscr{A})$ est-il dénoté par

$$(x + y(x^*y + z))^*(yx^+z + z)(y(x + y(x^*y + z))^*(yx^+z + z) + z)^*.$$

Le théorème pricipal de ce chapitre se déduit immédiatement des corollaires 39 et 41 :

Théorème 42 (Théorème de KLEENE). *Les langages réguliers et reconnaissables coïncident : REG = REC.*

3.5 Lemme de l'étoile

Théorème 43 (Lemme de l'étoile). *Soit L un langage régulier. Il existe un entier k tel que tout mot $w \in L$ de longueur $\geq k$ peut s'écrire sous la forme $w = xyz$ avec :*

1. $|xy| \leq k$,
2. $|y| > 0$,
3. $xy^iz \in L$ *pour tout* $i \geq 0$.

PREUVE. Soient $\mathscr{A}$ un AFD reconnaissant L et k le nombre d'états de A. Soit $w = w_1 \dots w_n$ un mot de L de longueur n. Notons

$$q_0 \xrightarrow{w_1} q_1 \xrightarrow{w_2} \dots \xrightarrow{w_n} q_n$$

le calcul de $\mathscr{A}$ sur w. Si $n \geq k$, ce calcul passe nécessairement deux fois par le même état. Autrement dit, il existe q_i, q_j dans la suite ci-dessus tels que $0 \leq i < j \leq n$ et $q_i = q_j$. Mais alors, pour chaque $t \geq 0$, la suite

$$q_0 \xrightarrow{w_1} \dots \xrightarrow{w_i} \{q_i \xrightarrow{w_{i+1}} \dots \xrightarrow{w_j} q_j\}^t \xrightarrow{w_{j+1}} \dots \xrightarrow{w_n} q_n$$

est un calcul de $\mathscr{A}$ ($\{q_i \xrightarrow{w_{i+1}} \dots \xrightarrow{w_j} q_j\}^t$ dénote le fait que cette séquence de calcul est répétée t fois). Notons $x = w_1 \dots w_i$, $y = w_{i+1} \dots w_j$ et $z = w_{j+1} \dots w_n$. Alors $xy^tz \in L$ pour chaque t, et $|xy| \leq k$, $|y| > 0$. □

Corollaire 44 *$\{a^n b^n, n \geq 0\}$ n'est pas régulier.*

PREUVE. Exercice. □

Corollaire 45 *Soit $\mathscr{A}$ un automate déterministe à k états. Alors :*

1. *$L(\mathscr{A}) \neq \varnothing$ ssi $L(\mathscr{A})$ contient un mot de longueur $< k$.*
2. *$L(\mathscr{A})$ est infini ssi $L(\mathscr{A})$ contient un mot de longueur comprise entre k et $2k$.*

PREUVE. Exercice. □

Bibliographie

[Aut92] J.-M. Autebert. *Calculabilité et Décidabilité : une Introduction*. Collection ERI. Masson, 1992.

[Car08] O. Carton. *Langages formels, Calculabilité et Complexité*. Capes Agreg. Vuibert, 2008.

[CLR94] T. Cormen, C. Leiserson, and R. Rivest. *Introduction à l'algorithmique*. Dunod, 1994.

[DW83] M.D. Davis and E.J. Weyuker. *Computability, Complexity, and Langages*. Academic press, 1983.

[HU79] J. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Languages and Computation*. Addison-Wesley, 1979.

[Koz97] Dexter Kozen. *Automata and computability*. Springer-Verlag, New York, 1997.

[Sip97] M. Sipser. *Introduction to the theory of computation*. PWS Publishing Company, 1997.

[Wol91] P. Wolper. *Introduction à la Calculabilité*. Collection IIA. InterEditions, 1991.

[Xuo92] N.H. Xuong. *Mathématiques Discrètes et Informatique*. Collection LMI. Masson, 1992.

[Xuo04] N.H. Xuong. *Informatique Théorique*. Sciences Sup. Dunod, 2004.