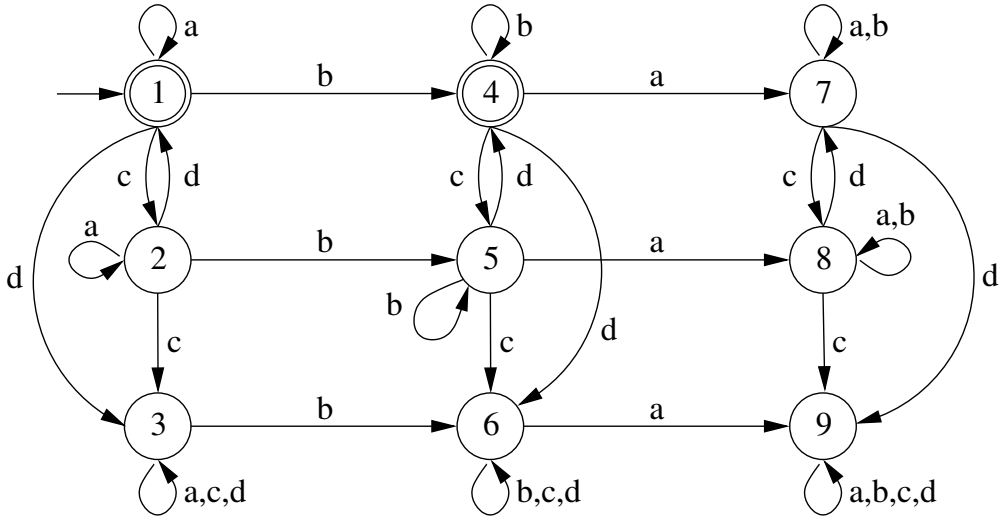
Université de Liège

Faculté des sciences
Département de mathématiques

Théorie des automates et langages formels



Année académique 2009–2010
Michel Rigo

Table des matières

CHAPITRE I

Mots et langages

Ce premier chapitre introduit quelques concepts fondamentaux de la théorie des langages formels et de la combinatoire sur les mots. La combinatoire des mots étudie les propriétés des suites de symboles. La théorie des langages formels englobe la théorie des automates et s'intéresse aux propriétés mathématiques des langages qui sont des ensembles de mots. Elle trouve notamment des applications en vérification et pour la compilation.

1. Premières définitions

Définition I.1.1. Un *alphabet* est un ensemble fini. Un alphabet sera en général désigné par une lettre grecque majuscule. Ainsi,

$$\Sigma = \{a, b, c\},\ \Gamma = \{\heartsuit, \diamondsuit, \clubsuit, \spadesuit\},\ \Delta = \{0, 1\}, \Phi = \{\rightarrow, \leftarrow, \uparrow, \downarrow\}$$

sont des alphabets. Les éléments d'un alphabet sont appelés *lettres* ou *symboles*

Exemple I.1.2. Le biologiste intéressé par l'étude de l'ADN utilisera un alphabet à quatre lettres $\{A, C, G, T\}$ pour les quatre constituants des gènes: Adénine, Cytosine, Guanine et Thymine.

Définition I.1.3. Soit Σ un alphabet. Un *mot* sur Σ est une suite finie (et ordonnée) de symboles. Par exemple, *abbac* et *ba* sont deux mots sur l'alphabet $\{a, b, c\}$. La *longueur* d'un mot w est le nombre de symboles constituant ce mot; on la note $|w|$. Ainsi,

$$|abbac| = 5 \quad \text{et} \quad |ba| = 2.$$

L'unique mot de longueur 0 est le mot correspondant à la suite vide. Ce mot s'appelle le *mot vide* et on le note ε. L'ensemble des mots sur Σ est noté Σ^*. Par exemple,

$$\{a, b, c\}^* = \{\varepsilon, a, b, c, aa, ab, ac, ba, bb, bc, ca, cb, cc, aaa, aab, \dots\}.$$

Définition I.1.4. Si σ est une lettre de l'alphabet Σ, pour tout mot $w = w_1 \cdots w_k \in \Sigma^*$, on dénote par

$$|w|_\sigma = \#\{i \in \{1, \dots, k\} \mid w_i = \sigma\}$$

le nombre de lettres σ_i apparaissant dans le mot w. Par exemple, $|abbac|_a = 2$ et $|abbac|_c = 1$.

Si l'alphabet Σ de cardinal $n \geq 1$ est ordonné, on pourra le considérer comme un n-uple $\Sigma = (\sigma_1, \ldots, \sigma_n)$. On définit alors la *fonction de Parikh* $\psi : \Sigma^* \to \mathbb{N}^n$ par

$$\psi(w) = (|w|_{\sigma_1}, \ldots, |w|_{\sigma_n}).$$

Le n-uple $\psi(w)$ est appelé *vecteur de Parikh* de w. Il est clair que si $n > 1$, alors ψ n'est pas injectif.

Définition I.1.5. Soit $w = w_1 \cdots w_\ell$ un mot sur Σ. Les mots

$$\varepsilon,\ w_1,\ w_1w_2,\ \ldots,\ w_1 \cdots w_{\ell-1},\ w_1 \cdots w_\ell = w$$

sont les *préfixes* de w. Un préfixe de w différent de ε et de w est dit *propre*. De façon semblable,

$$\varepsilon,\ w_\ell,\ w_{\ell-1}w_\ell,\ \ldots,\ w_2 \cdots w_\ell,\ w_1 \cdots w_\ell = w$$

sont les *suffixes* de w. Un suffixe de w est qualifié de *propre* s'il diffère de ε et de w. Soient $1 \leq i \leq j \leq \ell$. Le mot $w_i \cdots w_j$ est un *facteur* du mot w. On le note parfois $w[i, j]$. Une fois encore, on parle de *facteur propre* lorsque ce dernier diffère de w et de ε. L'ensemble des préfixes (resp. suffixes, facteurs) de w est noté $\mathrm{Pref}(w)$ (resp. $\mathrm{Suff}(w)$, $\mathrm{Fac}(w)$).

Remarque I.1.6. On peut observer que puisque Σ est un ensemble fini, Σ^* est dénombrable[1].

Rappelons la définition d'un monoïde.

Définition I.1.7. Soient A un ensemble et $\circ : A \times A \to A$ une opération binaire interne et partout définie. L'ensemble A muni de l'opération $\circ$ possède une structure de *monoïde* si les propriétés suivantes sont satisfaites.

- L'opération $\circ$ est associative :

$$\forall x, y, z \in A : (x \circ y) \circ z = x \circ (y \circ z).$$

- Il existe un neutre (unique) $e \in A$ tel que

$$\forall x \in A : x \circ e = e \circ x = x.$$

Remarque I.1.8. Un monoïde $(A, \circ)$ qui est tel que tout élément de A possède un inverse est un groupe.

Exemple I.1.9. Tout groupe est un monoïde; $(\mathbb{N}, +)$ est un monoïde qui n'est pas un groupe.

Profitons-en pour rappeler la définition d'un morphisme de monoïdes.

[1] En effet, les éléments de Σ^* peuvent chacun être caractérisés par un nombre fini d'indices prenant leur valeur dans des ensembles dénombrables (ici, il s'agit même d'ensembles finis, à savoir Σ).

Définition I.1.10. Soient $(A, \circ)$ et (B, ∇) deux monoïdes de neutre respectif e_A et e_B. Une application $f : A \to B$ est un *morphisme* (ou encore homomorphisme) *de monoïdes* si

$$\forall x, y \in A : f(x \circ y) = f(x) \nabla f(y) \tag{1}$$

et

$$f(e_A) = e_B. \tag{2}$$

Remarque I.1.11. Dans le cas d'un homomorphisme de groupes, la condition (2) est une conséquence directe de (1) et de l'existence d'inverse au sein des groupes[2]. Par contre, dans le cas de monoïdes, la condition (2) fait bel et bien partie de la définition d'un morphisme de monoïdes.

Définition I.1.12. Soit Σ un alphabet. On définit l'opération de *concaténation* sur Σ^* de la façon suivante. Pour tous mots $u = u_1 \cdots u_k$ et $v = v_1 \cdots v_\ell$, $u_i, v_i \in \Sigma$, la concaténation de u et v, notée $u.v$ ou simplement uv, est le mot

On utilisera dorénavant la notation multiplicative.

$$w = w_1 \cdots w_{k+\ell} \quad \text{où} \quad \begin{cases} w_i = u_i & , 1 \leq i \leq k \\ w_{k+i} = v_i & , 1 \leq i \leq \ell \end{cases}.$$

Ainsi, Σ^* muni de l'opération de concaténation est un monoïde de neutre ε. En particulier, on définit la puissance n-ième d'un mot w comme la concaténation de n copies de w,

$$w^n = \underbrace{w \cdots w}_{n \text{ fois}}.$$

On pose $w^0 = \varepsilon$.

Remarque I.1.13. Il est utile de remarquer que si $\#\Sigma > 1$, alors Σ^* est un monoïde *non commutatif*, i.e., il existe $u, v \in \Sigma^*$ tels que $uv \neq vu$.

Exemple I.1.14. L'application longueur

$$|\cdot| : \Sigma^* \to \mathbb{N}$$

est un morphisme de monoïdes entre $(\Sigma^*, .)$ et $(\mathbb{N}, +)$. En effet,

$$\forall u, v \in \Sigma^* : |uv| = |u| + |v|$$

et $|\varepsilon| = 0$.

Exemple I.1.15. Considérons l'alphabet $\Sigma = \{a, b, c\}$ et le morphisme $\varphi : \Sigma^* \to \Sigma^*$ défini par $\varphi(a) = abc$, $\varphi(b) = ac$ et $\varphi(c) = b$. En effet, pour définir un tel morphisme, on remarquera qu'il suffit de se donner l'image de lettres. On a, par exemple,

$$\varphi(abbc) = \varphi(a)\varphi(b)\varphi(b)\varphi(c) = abcacacb.$$

[2] pour tout $x \in A$, $f(x) = f(e_A \circ x) = f(e_A) \nabla f(x)$. D'où la conclusion en multipliant par $f(x)^{-1}$.

Voici à présent quelques propriétés classiques de *combinatoire des mots* (classification `68R15` de l'American Mathematical Society). On s'intéresse principalement aux configurations des lettres, des facteurs ou encore des motifs pouvant apparaître dans un cadre non commutatif (caractère inévitable, fréquence d'apparition, etc.). Voir, par exemple, l'excellent survol [**10**].

Proposition I.1.16. *Sur un alphabet binaire, tout mot de longueur au moins* 4 *contient un carré, i.e., un facteur de la forme* uu, $u \neq \varepsilon$.

Cette propriété triviale montre donc que l'apparition d'un carré est *inévitable* sur un alphabet de deux lettres. Par contre, sur trois lettres, il n'en est rien. Ainsi, la classification des motifs évitables ou non est loin d'être aisée.

Un *mot infini* sur un alphabet Σ est simplement une application $w : \mathbb{N} \to \Sigma$ (i.e., une suite de lettres indexée par $\mathbb{N}$). On peut munir l'ensemble Σ^ω des mots infinis sur Σ d'une distance $d : \Sigma^\omega \times \Sigma^\omega \to \mathbb{R}$ définie comme suit. Si x et y sont deux mots infinis, alors $x \wedge y$ désigne leur plus long préfixe commun. Si $x = y$, alors on pose $d(x, y) = 0$, sinon

$$d(x, y) = 2^{-|x \wedge y|}.$$

On vérifiera aisément qu'il s'agit bien d'une distance. Cette distance possède une propriété supplémentaire, elle est *ultramétrique*[3] (on utilise parfois le terme *non-archimédienne*) :

$$\forall x, y, z \in \Sigma^\omega : \ d(x, z) \leq \max\{d(x, y), d(y, z)\}.$$

Ayant à notre disposition un espace métrique (Σ^ω, d), on peut parler de suites convergentes de mots infinis, etc. Soit c une lettre n'appartenant pas à Σ. On peut plonger Σ^* dans $(\Sigma \cup \{c\})^\omega$ en identifiant le mot fini $w \in \Sigma^*$ avec le mot infini $wccc \cdots \in (\Sigma \cup \{c\})^\omega$. Cette identification faite, il est licite de parler d'une suite de mots finis convergeant vers un mot infini limite.

Proposition I.1.17. *Le mot infini* $\varphi^\omega(a)$ *où* $\varphi(a) = abc$, $\varphi(b) = ac$ *et* $\varphi(c) = b$, *est sans carré.*

On remarque facilement que $\varphi^n(a)$ est préfixe de $\varphi^{n+1}(a)$ pour tout $n \geq 0$. Il suffit de procéder par récurrence. Si $\varphi^{n+1}(a) = \varphi^n(a)u$, alors $\varphi^{n+2}(a) = \varphi^{n+1}(a)\varphi(u)$. De plus, la suite $(|\varphi^n(a)|)_{n \geq 0}$ est strictement croissante. Pour ces deux raisons et avec la topologie associée à la métrique présentée précédemment, on peut dire que la suite $(\varphi^n(a))_{n \geq 0}$ converge vers un mot infini limite.

[3] On rencontre notamment ce type de propriété en analyse p-adique. La topologie associée est intéressante : tout point d'une boule en est le centre, deux boules ont une intersection non vide si et seulement si l'une est incluse dans l'autre, tout triangle est isocèle, etc.

$$\begin{aligned}
&\varphi^0(a) = a\\
&\varphi^1(a) = abc\\
&\varphi^2(a) = abcacb\\
&\varphi^3(a) = abcacbabcbac\\
&\vdots
\end{aligned}$$

La démonstration du fait que le mot infini limite $\varphi^\omega(a)$ est sans carré[4] sera donnée en fin de section. En particulier, sur un alphabet de trois lettres, il existe des mots (finis) arbitrairement longs sans carré. Pour obtenir ce résultat, nous montrerons d'abord qu'il existe, sur deux lettres, des mots arbitrairement longs sans chevauchement.

Proposition I.1.18. *Deux mots u et v commutent s'ils sont puissances d'un même troisième, i.e., s'il existe un mot w et des entiers i, j tels que $u = w^i$ et $v = w^j$.*

Démonstration. On procède par récurrence sur la longueur de uv. Si $|uv| = 0$, le résultat est immédiat. Supposons à présent le résultat satisfait pour $|uv| < n$. Soient u, v tels que $|uv| = n$. On peut même considérer que $u \neq \varepsilon$ et $v \neq \varepsilon$ car sinon, le résultat serait trivial. Si $|u| = |v|$, alors il est immédiat que $u = v$. Sinon, on peut supposer que $|u| < |v|$ (voir figure I.1). Dès lors, il existe u' tel que $v = u'u$ et $|u'| < |v|$. Ainsi,

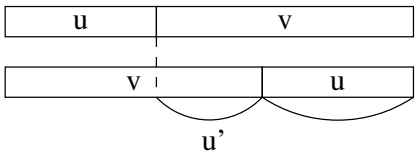


FIGURE I.1. $uv = vu$.

$uv = uu'u = vu = u'uu$ et donc on trouve $u'u = uu'$. Puisque $|uu'| < |uv|$, on peut appliquer l'hypothèse de récurrence. Il existe un mot w et des entiers p, q tels que $u = w^p$ et $u' = w^q$. Pour conclure, on remarque que $v = u'u = w^{p+q}$.

■

Remarque I.1.19. Noter que la réciproque du résultat ci-dessus est triviale.

On a également le résultat plus général suivant (dont la réciproque est elle aussi immédiate).

Proposition I.1.20. *Si x, y, z sont des mots tels que*

$$xy = yz$$

[4]cf. par exemple, M. Lothaire, *Combinatorics on words*, Cambridge Mathematical Library, Cambridge University Press, Cambridge, 1997.

avec x non vide, alors il existe des mots u, v et un entier $k \geq 0$ tels que

$$x = uv,\ y = (uv)^k u = u(vu)^k \text{ et } z = vu.$$

Démonstration. Si $|x| \geq |y|$, alors nous avons la situation suivante. Ainsi,

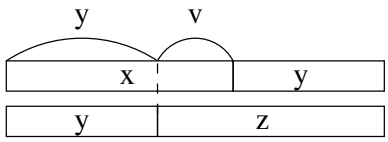


FIGURE I.2. $xy = yz$, $|x| \geq |y|$.

il existe un mot v tel que $x = yv$ (si $|x| = |y|$, alors $v = \varepsilon$). Dans ce cas, on peut prendre $u = y$ et $k = 0$.

Si $0 < |x| < |y|$, on procède par récurrence sur $|y|$. Si $|y| = 2$ et $|x| = |z| = 1$, on a

$$x\, y_1\, y_2 = y_1\, y_2\, z, \quad x, z, y_1, y_2 \in \Sigma$$

et on en déduit que $x = y_1 = y_2 = z$. Donc, $u = y_1$, $v = \varepsilon$ et $k = 1$ conviennent. Supposons à présent la propriété satisfaite pour $|y| \leq n$ et vérifions-la pour $|y| = n + 1$. Puisque $|x| < |y|$, il existe un mot w tel que

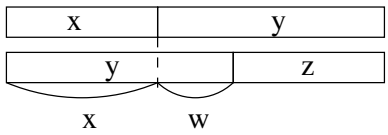


FIGURE I.3. $xy = yz$, $|x| < |y|$.

$y = xw$. Ainsi, $xy = yz$ se réécrit

$$xxw = xwz.$$

De là, on tire $xw = wz$ avec $|w| < |y|$ car $|x| > 0$. Soit $|x| \geq |w|$ et on applique la première partie de la preuve, soit $|x| < |w|$ et on peut dès lors appliquer l'hypothèse de récurrence : il existe des mots u, v et un entier k tels que

$$x = uv,\ w = (uv)^k u = u(vu)^k \text{ et } z = vu.$$

Pour conclure, on remarque que

$$y = xw = uv(uv)^k u = (uv)^{k+1} u.$$

■

Définition I.1.21. Soit $w = w_1 \cdots w_\ell$ un mot, avec $w_i \in \Sigma$ pour tout i. L'entier $k \geq 1$ est une *période* de w si

$$w_i = w_{i+k}, \quad \forall i = 1, \ldots, \ell - k.$$

On dit aussi que w est *k-périodique*. Un mot 1-périodiqe est *constant*. Par exemple, le mot

$$abbabbabba$$

est 3-périodique. Au vu de cette définition, un mot de longueur ℓ est p-périodique pour tout $p \geq \ell$.

Lemme I.1.22. *Soient p, q deux entiers. Si w est $(p.q)$-périodique avec $|w| \geq p.q$ et si le préfixe de w de longueur $p.q$ est p-périodique, alors w est lui-même p-périodique.*

Démonstration. C'est évident.

■

Théorème I.1.23 (Fine-Wilf[5]). *Si un mot w possède deux périodes p et q et si $|w| \geq p + q - \text{pgcd}(p, q)$, alors $\text{pgcd}(p, q)$ est aussi une période de w.*

Commençons par un lemme traitant d'un cas particulier du théorème.

Lemme I.1.24. *Si un mot w de longueur $p + q - 1$ possède deux périodes p et q premières entre elles, alors w est constant, i.e., 1-périodique.*

Démonstration. Soit $w = w_1 \cdots w_{p+q-1}$ de longueur $p + q - 1$ avec $\text{pgcd}(p, q) = 1$. Soit l'application $f : \{0, \ldots, p + q - 1\} \to \{0, \ldots, p + q - 1\}$ définie[6] par

$$f(x) = \begin{cases} x + p & \text{si } 0 \leq x < q \\ x - q & \text{si } q \leq x \leq p + q - 1. \end{cases}$$

On remarque que f est en fait une permutation de $\{0, \ldots, p + q - 1\}$ qui envoie $[0, q - 1]$ (resp. $[q, p + q - 1]$) sur $[p, p + q - 1]$ (resp. $[0, p - 1]$). Montrons à présent que $\{f^i(0) \mid i \geq 0\}$ décrit $\{0, \ldots, p + q - 1\}$. Soit $j > 0$ tel que $f^j(0) = 0$ (un tel j existe toujours car f est une permutation et se décompose donc en produit de cycles). Par définition même de f, cela signifie qu'il existe $a, b \in \mathbb{N}$ tels que $j = a + b$ et $ap - bq = 0$. Puisque $ap = bq$ et que p et q sont premiers entre eux, on en conclut que $p|b$, $q|a$ et donc $j \geq p + q$. Par conséquent, la permutation de $\{0, \ldots, p + q - 1\}$ induite par f se compose d'un unique cycle de longueur $p + q$.

Remarquons à présent que l'application f est en relation étroite avec nos hypothèses de périodicité : $w_i = w_{i+p}$ pour tout $1 \leq i < q$ et $w_j = w_{j-q}$ pour tout $q < j \leq p + q - 1$. De ce qui précède, on tire que w est un mot constant (i.e., 1-périodique). En effet, on obtient en fait

$$w_{f(0)} = w_{f^2(0)} = \cdots = w_{f^{p+q-1}(0)}$$

et $\{f(0), \ldots, f^{p+q-1}(0)\} = \{1, \ldots, p + q - 1\}$.

■

[5] N. J. Fine, H. S. Wilf, Uniqueness theorems for periodic functions, *Proc. Amer. Math. Soc.* **16** (1965), 109–114.

[6] Définir f sur $\{0, \ldots, p + q - 1\}$ et non sur $\{1, \ldots, p + q - 1\}$ comme cela aurait pu sembler naturel, nous sera bien utile. Au vu de la preuve, pourquoi ?

Nous pouvons à présent procéder à la preuve du théorème de Fine et Wilf.

Démonstration. On peut supposer que $|w| = p + q - \mathrm{pgcd}(p, q)$. En fait, si le mot w est plus long, on considère son préfixe v de longueur $p + q - \mathrm{pgcd}(p, q)$. Si l'on montre que v possède $\mathrm{pgcd}(p, q)$ comme période, alors grâce au lemme I.1.22, le résultat s'étend à w tout entier car w possède déjà p ou q comme période.

On peut de plus supposer que $d = \mathrm{pgcd}(p, q) = 1$, car sinon, en prenant une lettre sur d dans w, on est présence de d mots de longueur $k = p/d + q/d - 1$

$$w_i\, w_{i+d} \cdots w_{i+(k-1)d}, \quad i = 1, \ldots, d$$

et de périodes p/d et q/d premières entre elles. Au vu du lemme I.1.24, chacun des d mots est constant et on en tire la d-périodicité de w.

■

Exemple I.1.25. La borne donnée dans le théorème de Fine et Wilf est optimale :

$$\underbrace{abaab}\underbrace{abaab}\underbrace{abaab}a$$

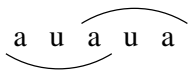
est 5-périodique, 13-périodique mais est de longueur $16 = 5+13-\mathrm{pgcd}(5, 13)-1$.

Pour terminer cette section, on définit, par récurrence sur la longueur de w, l'opération *miroir*[7] de la manière suivante : si $|w| = 0$, alors $w = \varepsilon$ et $w^R = \varepsilon$; sinon $|w| > 0$ et $w = \sigma u$, $\sigma \in \Sigma$, $u \in \Sigma^*$ et $w^R = u^R\sigma$. Si w est tel que

$$w^R = w,$$

alors w est un *palindrome*.

Définition I.1.26. Un mot fini de la forme $auaua$ où $u \in \Sigma^*$ et $a \in \Sigma$ est un *chevauchement* (en anglais, *overlap*).



On remarque que tout chevauchement contient un carré. De même, un cube (i.e., mot de la forme uuu) est un chevauchement particulier.

Nous avons vu que sur un alphabet binaire, tout mot de longueur ≥ 4 contient un carré. Le fait de contenir un chevauchement est une propriété plus forte. Cette propriété est-elle évitable sur deux lettres ?

Proposition I.1.27. *Le mot de Thue-Morse, défini comme $t = f^\omega(a)$ où $f(a) = ab$ et $f(b) = ba$, est un mot infini sans chevauchement.*

[7] On utilise la lettre R car la terminologie anglo-saxonne fait souvent référence au mot "reversal" ou "reverse". Dans la littérature, on trouve parfois la notation $\widetilde{w}$.

$$t = abbabaabbaababbabaababbaabbabaab\cdots$$

La preuve de ce résultat est calquée sur celle présentée dans [**21**]. Pour des applications du mot de Thue-Morse, on lira [**4**].

Lemme I.1.28. *Soit $X = \{ab, ba\}$. Si x appartient à X^*, alors axa et bxb n'appartiennent pas à X^*.*

Démonstration. On procède par récurrence sur $|x|$. Si $x = \varepsilon$, il est clair que $aa, bb \notin X^*$. Supposons le résultat vérifié pour les mots de longueur $< n$. Soit $x \in X^*$, un mot de longueur n. Procédons par l'absurde et supposons que $u = axa \in X^*$ (on procède de manière semblable avec bxb). Dans ce cas, $u = abyba$ avec $|y| = |x| - 2$. Puisque $y \in X^*$, on en conclut, par hypothèse de récurrence, que $byb = x$ n'appartient pas à X^*. Ceci est une contradiction.

■

Lemme I.1.29. *Soient $w \in \{a, b\}^+$ et $f : a \mapsto ab, b \mapsto ba$ le morphisme de Thue-Morse. Si w est sans chevauchement, alors $f(w)$ aussi.*

Démonstration. Montrons que si $f(w)$ possède un chevauchement, alors w aussi. Supposons que $f(w)$ se factorise en

$$f(w) = x\,c\,v\,c\,v\,c\,y, \quad c \in \{a, b\},\ x, v, y \in \{a, b\}^*.$$

Puisqe f est 2-uniforme (i.e., $|f(a)| = |f(b)| = 2$), $|f(w)|$ est pair. On remarque que $|cvcvc| = 3 + 2|v|$ est impair. Par conséquent, $|xy|$ est impair.

Montrons à présent que $|v|$ est impair.

- Si $|x|$ est pair, alors x, $cvcv$ et cy appartiennent à $\{ab, ba\}^*$.
  Dès lors, si $|v|$ était pair, alors cvc et v appartiendraient à $\{ab, ba\}^*$.
  Ceci est en contradiction avec le lemme précédent.
- Si $|x|$ est impair, alors xc, $vcvc$ et y appartiennent à $\{ab, ba\}^*$. Si $|v|$ était pair, on aboutirait à la même contradiction.

Nous pouvons à présent conclure, en discutant une fois encore sur la parité de $|x|$.

- Si $|x|$ est pair, alors, puisque $|v|$ est impair, on a

$$f(w) = \underbrace{x}_{\text{pair}}\,\underbrace{c\,\overbrace{v}^{\text{impair}}}_{\text{pair}}\,\underbrace{cv}_{\text{pair}}\,cy \in \{ab, ba\}^*$$

  et x, cv, cy appartiennent à $\{ab, ba\}^*$. Il existe r, s, t tels que $f(r) = x$, $f(s) = cv$, $f(t) = cy$ et

$$w = rsst.$$

  Or $f(s)$ et $f(t)$ débutent par la même lettre, donc s et t aussi (vu la définition de f). Par conséquent, sst débute par un chevauchement.

▸ Si $|x|$ est impair, alors, puisque $|v|$ est impair, on a

$$f(w) = \underbrace{xc}_{\text{pair}}\overbrace{v}^{\text{impair}}\underbrace{c}_{\text{pair}}\underbrace{vc}_{\text{pair}}y \in \{ab, ba\}^*$$

et il existe r, s, t tels que $f(r) = xc$, $f(s) = vc$, $f(t) = y$. La conclusion est identique.

■

Nous pouvons à présent démontrer la proposition I.1.27.

Démonstration. Supposons que t possède un chevauchement. En particulier, ce chevauchement apparaît dans le préfixe $f^k(a)$ pour un certain k. Or a étant sans chevauchement, le lemme précédent stipule que $f(a)$ est sans chevauchement et donc, en itérant, $f^k(a)$ ne peut possèder de chevauchement.

■

Nous pouvons à présent reconsidérer la proposition I.1.17 et sa preuve.

Remarque I.1.30. Soit r un mot infini sur $\{a, b\}$ sans chevauchement et commençant par a. Alors r se factorise de manière unique sous la forme[8] $r = y_1 y_2 \cdots$ où pour tout $i \geq 1$, $y_i \in \{a, ab, abb\}$. En effet, r ne contenant aucun cube, il ne peut contenir le facteur aaa ou bbb.

Soit le morphisme $g : \{a, b, c\}^* \to \{a, b\}^*$ défini par

$$g : \begin{cases} a \mapsto abb \\ b \mapsto ab \\ c \mapsto a \end{cases} .$$

Si r un mot infini sur $\{a, b\}$ sans chevauchement et débutant par a, alors il existe un unique mot infini s sur $\{a, b, c\}$ tel que $g(s) = r$.

Proposition I.1.31. *Soit t un mot infini sur $\{a, b\}$ sans chevauchement et débutant par a (comme le mot de Thue-Morse). Soit s l'unique mot infini $\{a, b, c\}$ tel que $g(s) = t$. Alors s est un mot infini sur trois lettres sans carré.*

Démonstration. Supposons que s contienne un carré : $s = x\,u\,u\,\sigma y$ avec u non vide, σ une lettre et y un mot infini. Alors $g(s)$ contient le facteur $g(u)g(u)g(\sigma)$ qui débute par un chevauchement car $g(u)$ et $g(\sigma)$ débutent par la même lettre.

■

2. Langages

Nous en avons terminé avec notre brève introduction à la combinatoire des mots. Passons à la théorie des langages formels.

[8] On remarquera que $\{a, ab, abb\}$ est un code.

Définition I.2.1. Un *langage* sur Σ est simplement un ensemble (fini ou infini) de mots sur Σ. En d'autres termes, un langage est une partie de Σ^*. On distingue en particulier le langage vide[9] $\emptyset$.

Exemple I.2.2. Considérons l'alphabet $\Sigma = \{a, b, c\}$. L'ensemble

$$\{a, aa, bbc, ccca, ababab\}$$

est un langage fini. L'ensemble L_{2a} des mots sur Σ comprenant un nombre pair de a est aussi un langage (infini),

$$L_{2a} = \{\varepsilon, b, c, aa, bb, bc, cb, cc, aab, aac, aba, aca, \ldots, abaacaaa, \ldots\}.$$

L'ensemble $\mathfrak{Pal}(\Sigma^*)$ formé des palindromes de Σ^* est aussi un langage infini,

$$\begin{aligned}\mathfrak{Pal}(\Sigma^*) \;=\; & \{\varepsilon, a, b, c, aa, bb, cc, aaa, aba, aca, bab, bbb, bcb, cac, cbc, ccc,\\ & aaaa, abba, acca, baab, bbbb, bccb, caac, cbbc, cccc, \ldots\}.\end{aligned}$$

Soit l'alphabet $\Delta = \{0, 1\}$. L'ensemble constitué des écritures binaires[10] des entiers positifs pairs est un langage sur Δ

$$\{10, 100, 110, 1000, 1010, 1100, 1110, \ldots\}$$

de même que le langage formé des écritures binaires des nombres premiers

$$\{10, 11, 101, 111, 1011, 1101, 10001, \ldots\}.$$

Passons à présent en revue quelques opérations sur les langages. Tout d'abord, puisqu'un langage est un ensemble, on dispose des opérations ensemblistes usuelles comme l'union, l'intersection ou encore la complémentation.

Définition I.2.3. Soient $L, M \subseteq \Sigma^*$ deux langages. La *concaténation* des langages L et M est le langage

$$LM = \{uv \mid u \in L,\ v \in M\}.$$

En particulier, on peut définir la *puissance* n-ième d'un langage L, $n > 0$, par

$$L^n = \{w_1 \cdots w_n \mid \forall i \in \{1, \ldots, n\},\ w_i \in L\}$$

et on pose $L^0 = \{\varepsilon\}$. Par exemple, si $L = \{a, ab, ba, ac\}$, alors

$$\begin{aligned}L^2 \;=\; & \{aa, aab, aba, aac, abab, abba, abac, baa, baab,\\ & baba, baac, aca, acab, acba, acac\}.\end{aligned}$$

[9] Ne pas confondre le langage vide ne contenant aucun élément et le langage $\{\varepsilon\}$ contenant uniquement le mot vide.

[10] Un mot $w = w_\ell \cdots w_0 \in \{0, 1\}^*$ représente l'entier n si $n = \sum_{i=0}^{\ell} w_i 2^i$. En général, on ne considère que des mots dont le premier symbole w_ℓ diffère de 0. Par convention, l'entier zéro est alors représenté par le mot vide.

Remarque I.2.4. Soit $n \geq 0$. L'ensemble des mots de longueur n sur Σ est Σ^n. Notons aussi que si un mot uv appartient à LM avec $u \in L$ et $v \in M$, cette factorisation n'est pas nécessairement unique. Par exemple, avec $L = \{a, ab, ba\}$, L^2 contient le mot aba qui se factorise en $a(ba)$ et $(ab)a$. Demander l'unicité de la factorisation débouche sur la notion de *code*. Ainsi, $X \subset \Sigma^*$ est un code, si tout mot de X^* se factorise de manière unique comme concaténation de mots de X.

Proposition I.2.5. *La concaténation de langages est une opération associative, elle possède $\{\varepsilon\}$ pour neutre, $\emptyset$ pour absorbant et est distributive à droite et à gauche pour l'union, i.e., si L_1, L_2, L_3 sont des langages*

$$L_1(L_2L_3) = (L_1L_2)L_3,$$
$$L_1\{\varepsilon\} = \{\varepsilon\}L_1 = L_1,$$
$$L_1\emptyset = \emptyset L_1 = \emptyset,$$
$$L_1(L_2 \cup L_3) = (L_1L_2) \cup (L_1L_3),$$
$$(L_1 \cup L_2)L_3 = (L_1L_3) \cup (L_2L_3).$$

Démonstration. C'est immédiat.

■

Définition I.2.6. Soit $L \subseteq \Sigma^*$. L'*étoile de Kleene*[11] de L est donnée par

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

Ainsi, les mots de L^* sont exactement les mots obtenus en concaténant un nombre arbitraire de mots de L.

Remarque I.2.7. On remarque que la notation Σ^* introduite précédemment est cohérente puisqu'il s'agit en fait de l'étoile de Kleene du langage fini Σ. On dit parfois que Σ^* est le *monoïde libre engendré par* Σ.

On rencontre parfois l'opération L^+ définie par

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

Par exemple, si Σ est un alphabet, alors $\Sigma^+ = \Sigma^* \setminus \{\varepsilon\}$. D'une manière générale, si L est un langage ne contenant pas le mot vide, alors $L^+ = L^* \setminus \{\varepsilon\}$.

Proposition I.2.8. *Soit $L \subseteq \Sigma^*$ un langage. Le langage L^* est le plus petit*[12] *langage M tel que $\varepsilon \in M$, $L \subseteq M$ et $M^2 \subseteq M$.*

[11] Stephen Cole Kleene (1909–1994), logicien, est, avec K. Gödel, A. Turing, A. Church, E. Post, l'un des pères fondateurs de l'informatique théorique. On lui doit notamment le concept d'expression régulière. S.C. Kleene, Representation of Events in Nerve Nets and Finite Automata, *Automata Studies*, Princeton, Princeton University Press, (1956) Ed. C. Shannon, J. McCarthy.

[12] Le plus petit pour l'inclusion.

Démonstration. Il est clair que L^* vérifie les trois propriétés. Si M satisfait les propriétés indiquées, nous devons montrer que $L^* \subseteq M$. Puisque $L \subseteq M$ et $M^2 \subseteq M$, on en conclut que $L^2 \subseteq M$. De proche en proche, on s'aperçoit que

$$L^i \subseteq M, \ \forall i > 0.$$

Ceci conclut la preuve.

■

Le résultat suivant concerne les langages sur un alphabet unaire et traduit en fait une propriété arithmétique élémentaire.

Théorème I.2.9. *Soit L un langage arbitraire sur un alphabet unaire. Il existe un langage fini F tel que $L^* = F^*$.*

Démonstration. Si L est fini, le résultat est immédiat. Il suffit de prendre $F = L$. Sinon, considérons le mot non vide a^p le plus court, $p \geq 1$, appartenant à L. Il est évident que $\{a^p\}^* \subseteq L^*$. Si cette inclusion est une égalité, alors le résultat est démontré ($F = \{a_p\}$). Sinon, soit a^{q_1} le mot le plus court appartenant à $L^* \setminus \{a^p\}^*$. Dès lors,

$$q_1 = t_1\, p + r_1, \quad \text{avec } 0 < r_1 < p \text{ et } t_1 \geq 1.$$

En effet, $q_1 > p$ et q_1 ne peut être multiple de p. Nous avons à présent que $\{a^p, a^{q_1}\}^* \subseteq L^*$. On effectue le même raisonnement. Si $\{a^p, a^{q_1}\}^* \neq L^*$, il existe un mot le plus court a^{q_2} appartenant à $L^* \setminus \{a^p, a^{q_1}\}^*$ tel que

$$q_2 = t_2\, p + r_2, \quad \text{avec } 0 < r_2 < p, \ r_2 \neq r_1 \text{ et } t_2 \geq t_1.$$

En effet, $q_2 > q_1$ et si $r_2 = r_1$, alors on aurait

$$q_2 = (t_2 - t_1)\, p + \underbrace{t_1\, p + r_1}_{=q_1}.$$

Cela signifierait alors que a^{q_2} appartient à $\{a^p, a^{q_1}\}^*$. On peut alors effectuer la même démarche avec $\{a^p, a^{q_1}, a^{q_2}\}$ et définir q_3 si $L^* \neq \{a^p, a^{q_1}, a^{q_2}\}^*$. Cependant, on remarque qu'il y a au plus $p-1$ restes non nuls distincts lors d'une division euclidienne par p. Par conséquent, on ne saurait effectuer ce raisonnement indéfiniment et finalement

$$L^* = \{a^p, a^{q_1}, \ldots, a^{q_s}\} \quad \text{avec } s \leq p - 1.$$

■

Définition I.2.10. On peut étendre les opérations d'obtention de préfixes, suffixes et facteurs aux langages. Soit L un langage. On définit

$$\mathrm{Pref}(L) = \bigcup_{w \in L} \mathrm{Pref}(w)$$

comme l'ensemble des préfixes des mots du langage L. De la même manière, on pose

$$\mathrm{Suff}(L) = \bigcup_{w \in L} \mathrm{Suff}(w) \quad \text{et} \quad \mathrm{Fac}(L) = \bigcup_{w \in L} \mathrm{Fac}(w).$$

Enfin, un langage L est *préfixiel* si $\mathrm{Pref}(L) = L$. Il suffit donc de vérifier que tout préfixe d'un mot de L est encore un mot de L. De la même manière, L est *suffixiel* (resp. *factoriel*) si $\mathrm{Suff}(L) = L$ (resp. $\mathrm{Fac}(L) = L$).

Définition I.2.11. Soit f un morphisme de monoïdes entre Σ^* et Γ^*. On remarque que f est complètement caractérisé par les images de f sur les symboles de Σ. Si L est un langage sur Σ, alors l'*image* de L par le morphisme f est

$$f(L) = \{f(u) \in \Gamma^* \mid u \in L\}.$$

De la même manière, si M est un langage sur Γ, alors l'*image inverse* de M par le morphisme f est

$$f^{-1}(M) = \{u \in \Sigma^* \mid f(u) \in M\}.$$

Exemple I.2.12. Soient $\Sigma = \{a, b, c\}$, $\Gamma = \{\mu, \nu\}$ et f le morphisme défini par

$$f(a) = \mu,\ f(b) = \nu,\ f(c) = \nu.$$

Si $L = \{ab, bc, cb, aaab, aaac\}$, alors

$$f(L) = \{\mu\nu, \nu\nu, \mu\mu\mu\nu\}.$$

Si $M = \{\mu\nu, \nu\mu, \nu\mu\nu\}$, alors

$$f^{-1}(M) = \{ab, ac, ba, ca, bab, bac, cab, cac\}.$$

Dans notre exemple, pour tout $\sigma \in \Sigma$, $|f(\sigma)| = 1$. Néanmoins, on peut en toute généralité considérer un morphisme dont les images des lettres de l'alphabet d'origine seraient de longueurs différentes.

Remarque I.2.13. Il arrive, dans de nombreuses situations, qu'on distingue le cas où il existe $\sigma \in \Sigma$ tel que $f(\sigma) = \varepsilon$ (on parle de "*morphisme effaçant*"), du cas où, pour tout $\sigma \in \Sigma$, $f(\sigma) \neq \varepsilon$ (on utilise dès lors l'expression "*morphisme non effaçant*").

Dans la section précédente, on a introduit le miroir d'un mot. Cette opération s'étend naturellement aux langages.

Définition I.2.14. Le *miroir* d'un langage L est

$$L^R = \{u^R \mid u \in L\}.$$

On peut avoir $L = L^R$ sans pour autant que les mots de L soient tous des palindromes.

Définition I.2.15. La *clôture commutative* d'un langage $L \subseteq \Sigma^*$ est définie par

$$\mathfrak{Com}(L) = \{w \in \Sigma^* \mid \exists u \in L : \forall \sigma \in \Sigma, |w|_\sigma = |u|_\sigma\}.$$

Cela signifie que $\mathfrak{Com}(L)$ contient les mots obtenus en permutant les lettres des mots de L. Par exemple, si $L = \{ab, bac, ccc\}$, alors

$$\mathfrak{Com}(L) = \{ab, ba, abc, acb, bac, bca, cab, cba, ccc\}.$$

En utilisant la fonction de Parikh introduite à la définition I.1.4, il est clair que

$$\mathfrak{Com}(L) = \psi^{-1}\psi(L).$$

Si L est un langage tel que $\mathfrak{Com}(L) = L$, alors L est dit *commutatif*.

Voici une dernière opération sur les mots et les langages.

Définition I.2.16. Le *shuffle*[13] de deux mots u et v est le langage

$$u \sqcup\!\sqcup v = \{u_1 v_1 \cdots u_n v_n \mid u = u_1 \cdots u_n,\ v = v_1 \cdots v_n,\ u_i, v_i \in \Sigma^*, n \geq 1\}.$$

Par exemple[14], si $u = ab$ et $v = cde$, alors

$$\begin{aligned} u \sqcup\!\sqcup v \ = \ & \{abcde, acbde, acdbe, acdeb, cabde, \\ & cadbe, cadeb, cdabe, cdaeb, cdeab\}. \end{aligned}$$

Le *shuffle* de deux langages se définit comme suit,

$$L \sqcup\!\sqcup M = \bigcup_{\substack{u \in L, \\ v \in M}} u \sqcup\!\sqcup v.$$

3. Expressions régulières et langages associés

La notion d'expression régulière est d'usage fréquent en informatique. En effet, on a souvent recours aux expressions régulières lorsqu'on désire rechercher certains motifs récurrents. Un exemple banal est celui d'un répertoire contenant divers fichiers :

```
> ls monrepertoire/
 memoire.aux   memoire.tex      picture001.jpg      rapsody.jpg
 memoire.dvi   picture001.jpg   presentation.exe    raw.jpg
 memoire.old   picture002.jpg   price-list.txt
 memoire.log   picture003.jpg   taches.txt
```

Si l'utilisateur désire afficher uniquement les images au format "JPEG" et comportant l'extension `.jpg`, il aura par exemple recours à une commande comme

```
ls *.jpg
```

De la même manière, s'il veut effacer tous les fichiers relatifs à `memoire`, il exécutera

```
rm m*
```

On pourrait imaginer, dans un répertoire plus fourni, vouloir sélectionner des fichiers dont les noms satisfont à des critères plus fins. Nous allons voir comment définir ce genre de critères dans le formalisme développé lors des précédentes sections.

[13] On pourrait tenter de traduire ce terme par "mélange". Nous avons choisi de conserver la dénomination anglo-saxonne.

[14] Nous avons pris ici deux mots n'ayant aucune lettre en commun pour rendre l'exemple plus simple. En toute généralité, on peut bien sûr prendre des mots possédant les mêmes lettres.

Définition I.3.1. Soit Σ un alphabet. Supposons que $0, e, +, ., (,), ^*$ sont des symboles n'appartenant pas à Σ. L'ensemble $\mathcal{R}_\Sigma$ des *expressions régulières* sur Σ est défini récursivement par

- 0 et e appartiennent à $\mathcal{R}_\Sigma$,
- pour tout $\sigma \in \Sigma$, σ appartient à $\mathcal{R}_\Sigma$,
- si ϕ et ψ appartiennent à $\mathcal{R}_\Sigma$, alors
  - $(\phi + \psi)$ appartient à $\mathcal{R}_\Sigma$,
  - $(\phi.\psi)$ appartient à $\mathcal{R}_\Sigma$,
  - ϕ^* appartient à $\mathcal{R}_\Sigma$.

Exemple I.3.2. Si $\Sigma = \{a, b\}$, voici quelques exemples d'expressions régulières :

$$\alpha_1 = (e + (a.b)),$$
$$\alpha_2 = (((a.b).a) + b^*)^*,$$
$$\alpha_3 = ((a + b)^*.(a.b)).$$

A une expression régulière, on associe un langage grâce à l'application[15]

$$\mathcal{L} : \mathcal{R}_\Sigma \to 2^{\Sigma^*}$$

par

- $\mathcal{L}(0) = \emptyset$, $\mathcal{L}(e) = \{\varepsilon\}$,
- si $\sigma \in \Sigma$, alors $\mathcal{L}(\sigma) = \{\sigma\}$,
- si ϕ et ψ sont des expressions régulières,
  - $\mathcal{L}[(\phi + \psi)] = \mathcal{L}(\phi) \cup \mathcal{L}(\psi)$,
  - $\mathcal{L}[(\phi.\psi)] = \mathcal{L}(\phi)\mathcal{L}(\psi)$,
  - $\mathcal{L}(\phi^*) = (\mathcal{L}(\phi))^*$.

Exemple I.3.3. Poursuivons l'exemple I.3.2. On a

$$\mathcal{L}(\alpha_1) = \{\varepsilon, ab\},$$
$$\mathcal{L}(\alpha_2) = (\{aba\} \cup \{b\}^*)^*,$$
$$\mathcal{L}(\alpha_3) = \{a, b\}^*\{ab\}.$$

Définition I.3.4. Un langage L sur Σ est *régulier* s'il existe une expression régulière $\phi \in \mathcal{R}_\Sigma$ telle que

$$L = \mathcal{L}(\phi).$$

Si ϕ et ψ sont deux expressions régulières telles que $\mathcal{L}(\phi) = \mathcal{L}(\psi)$, alors on dit que ϕ et ψ sont *équivalentes*.

Remarque I.3.5. Dans la suite, on s'autorisera à confondre une expression régulière et le langage qu'elle représente. Si aucune confusion n'est possible, on s'autorisera également à enlever les parenthèses ou autres symboles superflus. Par exemple,

$$(((b^*.a).(b^*.a))^*.b^*) = (b^*a\,b^*a)^*\,b^*$$

[15] La notation 2^{Σ^*} désigne l'ensemble des parties de Σ^*, c'est-à-dire l'ensemble des langages sur Σ. On trouve parfois la notation $\mathcal{P}(\Sigma^*)$.

représente le langage formé des mots sur $\{a, b\}$ comprenant un nombre pair de a. On se convainc aisément que ce langage est aussi représenté par l'expression

$$b^* \, (a \, b^* a \, b^*)^*.$$

Proposition I.3.6. *L'ensemble $\mathcal{L}(\mathcal{R}_\Sigma)$ des langages réguliers sur Σ est la plus petite famille de langages contenant le langage vide, les langages $\{\sigma\}$ réduits à une lettre ($\sigma \in \Sigma$) et qui est stable pour les opérations d'union, de concaténation et d'étoile de Kleene.*

Démonstration. Par définition de $\mathcal{R}_\Sigma$ et de $\mathcal{L}$, il est clair que l'ensemble des langages réguliers sur Σ vérifie les propriétés énoncées.

Soit $\mathcal{A}$ un ensemble de langages satisfaisant les propriétés énoncées. Nous devons vérifier que $\mathcal{L}(\mathcal{R}_\Sigma) \subset \mathcal{A}$. Soit L un langage régulier. Il existe $\psi \in \mathcal{R}_\Sigma$ tel que $\mathcal{L}(\psi) = L$. On procède par récurrence sur la longueur[16] de l'expression régulière ψ :

Si ψ vaut 0, e ou σ ($\sigma \in \Sigma$), alors $\mathcal{L}(\psi)$ vaut $\emptyset$, $\{\varepsilon\} = \emptyset^*$ ou $\{\sigma\}$. Par conséquent, L appartient à $\mathcal{A}$.

Si $\psi = (\phi + \mu)$ avec ϕ et μ des expressions régulières sur Σ de longueur inférieure à celle de ψ, alors on a

$$\mathcal{L}(\psi) = \mathcal{L}(\phi) \cup \mathcal{L}(\mu).$$

Par hypothèse de récurrence, $\mathcal{L}(\phi)$ et $\mathcal{L}(\mu)$ appartiennent à $\mathcal{A}$. Puisque $\mathcal{A}$ est stable pour l'union, on en conclut que L appartient à $\mathcal{A}$.

Si $\psi = (\phi.\mu)$ ou $\psi = \phi^*$, on utilise le même raisonnement.

■

Remarque I.3.7. Dans la proposition précédente, on aurait pu remplacer "langages $\{\sigma\}$ réduits à une lettre" par "langages finis". C'est équivalent, au vu des propriétés de stabilité énoncées.

Puisque nous avons décidé de substituer des langages aux expressions régulières, les relations suivantes sont immédiates.

Proposition I.3.8. *Soit ψ une expression régulière. On a*

- $\psi + \psi = \psi$,
- $e \, \psi = \psi \, e = \psi$,
- $0 \, \psi = \psi \, 0 = 0$,
- $(\psi^*)^* = \psi^*$,
- $\psi^* = \psi^0 + \psi^1 + \cdots + \psi^k + \psi^{k+1}\psi^*$,
- $(\psi + \phi)^* = (\psi^* \, \phi)^* \psi^*$.

Dans le cas particulier d'un alphabet unaire (i.e., contenant un seul symbole), on dispose d'une caractérisation des langages réguliers.

[16] On peut définir la longueur d'une expression régulière de la manière suivante. Soient ψ, ϕ deux expressions régulières. Si $\psi = 0, e$ ou σ ($\sigma \in \Sigma$), alors $|\psi| = 1$. De plus, $|(\psi + \phi)| = |\psi| + |\phi| + 1$, $|(\psi.\phi)| = |\psi| + |\phi| + 1$ et $|\phi^*| = |\phi| + 1$.

Proposition I.3.9. *Soit $\Sigma = \{\sigma\}$. Les langages réguliers sur Σ sont exactement les langages de la forme*

$$\{\sigma^i \mid i \in A\}$$

où $A \subseteq \mathbb{N}$ est une union finie de progressions arithmétiques.

Rappelons qu'une progression arithmétique est un ensemble de la forme

$$p + \mathbb{N}.q = \{p + n.q \mid n \in \mathbb{N}\}$$

avec $p, q \in \mathbb{N}$.

Démonstration. Nous avons déjà remarqué (cf. exemple I.1.14) que l'application longueur est un morphisme de monoïdes entre $(\Sigma^*, .)$ et $(\mathbb{N}, +)$. Ici, l'application

$$|\cdot| : \{\sigma\}^* \to \mathbb{N} : \sigma^n \mapsto n$$

est même un isomorphisme[17] de monoïdes. L'ensemble $\mathcal{P}$ des unions finies de progressions arithmétiques jouit des propriétés suivantes :

- $\emptyset \in \mathcal{P}$ (cas de l'union vide),
- $\{1\} \in \mathcal{P}$ car $\{1\} = 1 + \mathbb{N}.0$,
- l'union de deux éléments de $\mathcal{P}$ est encore un élément de $\mathcal{P}$ (en effet, l'union de deux unions finies de progressions arithmétiques est encore une union finie de progressions arithmétiques),
- la somme de deux éléments de $\mathcal{P}$ est encore un élément de $\mathcal{P}$. Pour le vérifier, puisque $\mathcal{P}$ est stable pour l'union, il suffit de considérer le cas de deux progressions arithmétiques $p + \mathbb{N}.q$ et $r + \mathbb{N}.s$. Si $q = 0$, alors

  $$(p + \mathbb{N}.q) + (r + \mathbb{N}.s) = (r + p) + \mathbb{N}.s \in \mathcal{P}.$$

  Si $q > 0$, alors

  $$(p + \mathbb{N}.q) + (r + \mathbb{N}.s) = \bigcup_{0 \le i < q} ((p + r + i\,s) + \mathbb{N}.q) \in \mathcal{P}.$$

  Il est clair que le membre de droite est inclus dans le membre de gauche. Montrons l'autre inclusion. Soit $t \in (p + \mathbb{N}.q) + (r + \mathbb{N}.s)$. Il existe $m, n \in \mathbb{N}$ tels que $t = p + r + m\,q + n\,s$. Si on effectue la division euclidienne de n par q, il existe ℓ et i tels que

  $$n = \ell\,q + i,\ 0 \le i < q.$$

  Par conséquent,

  $$t = p + r + m\,q + (\ell\,q + i)\,s = p + r + i\,s + (m + \ell\,s)\,q$$

  avec $0 \le i < q$.

[17] Un isomorphisme est un morphisme bijectif. Il est clair que nous avons une bijection uniquement dans le cas d'un alphabet unaire. En effet, si $\Sigma = \{a, b\}$, alors $|ab| = |ba| = 2$ mais $ab \neq ba$ et l'application longueur n'est donc pas injective.

On peut définir l'*étoile* d'une partie A de $\mathbb{N}$ par

$$A^* = \{a_1 + \cdots + a_n \mid n \in \mathbb{N} \text{ et } \forall i \in \{1, \ldots, n\}, a_i \in A\}.$$

En particulier, 0 appartient toujours à A^* et ce, quel que soit $A \subseteq \mathbb{N}$. Ainsi, l'ensemble $\mathcal{P}$ jouit encore d'une cinquième propriété.

- Si $A \in \mathcal{P}$, alors $A^* \in \mathcal{P}$. Il suffit de le vérifier pour une progression arithmétique car si $A, B \subseteq \mathbb{N}$, alors, puisque l'addition dans $\mathbb{N}$ est commutative,
$$(A \cup B)^* = A^* + B^*$$
et on a vu que $\mathcal{P}$ était stable par addition. Par définition, il vient
$$(p + \mathbb{N}.q)^* = \{p + n_1\, q + \cdots + p + n_j\, q \mid n_1, \ldots, n_j \in \mathbb{N}, j > 0\} \cup \{0\}.$$
Si $p = 0$, $(\mathbb{N}.q)^* = \{q\}^* = \mathbb{N}.q \in \mathcal{P}$. Si $p > 0$,
$$(p + \mathbb{N}.q)^* = \{0\} \cup \bigcup_{0 \leq i < p} ((p + i\, q) + \mathbb{N}.p).$$
Il est clair que le membre de droite est inclus dans le membre de gauche. Vérifions l'autre inclusion. Soit $j > 0$. On a
$$p + n_1\, q + \cdots + p + n_j\, q = p + (j-1)\, p + (n_1 + \cdots + n_j)\, q.$$
En effectuant la division euclidienne de $n_1 + \cdots + n_j$ par p, on trouve
$$n_1 + \cdots + n_j = m\, p + i, \quad \text{avec} \quad 0 \leq i < p$$
et donc
$$p + n_1\, q + \cdots + p + n_j\, q = p + i\, q + (j - 1 + m\, q)\, p$$
avec $0 \leq i < p$.

Supposons à présent que $\mathcal{Q} \subseteq 2^{\mathbb{N}}$ est une famille de parties de $\mathbb{N}$ qui contient $\emptyset$ et $\{1\}$ et qui est stable pour l'union, la somme et l'étoile. Montrons que $\mathcal{P} \subseteq \mathcal{Q}$. Puisque $\mathcal{Q}$ est stable pour l'union, il suffit de montrer que les progressions arithmétiques de la forme $p + \mathbb{N}.q$, $p, q \in \mathbb{N}$, appartiennent à $\mathcal{Q}$.

Puisque $\emptyset \in \mathcal{Q}$, on a $\emptyset^* = \{0\} \in \mathcal{Q}$. En outre, $\{1\} \in \mathcal{Q}$ et en utilisant le fait que $\mathcal{Q}$ est stable pour l'addition, on voit que $\{r\}$ appartient à $\mathcal{Q}$ pour tout $r > 0$. On en déduit que, pour tous $p, q \in \mathbb{N}$,
$$p + \mathbb{N}.q = \{p\} + \{q\}^*$$
appartient à $\mathcal{Q}$.

On conclut en utilisant la proposition I.3.6 et le fait que l'application longueur est un isomorphisme entre $(\mathbb{N}, +)$ et $(\{\sigma\}^*, .)$.

■

La contraposée du corollaire suivant permet parfois de vérifier que certains langages ne sont pas réguliers.

Corollaire I.3.10. *Si $L \subseteq \Sigma^*$ est un langage régulier sur un alphabet fini arbitraire, alors l'ensemble*

$$|L| = \{|w| : w \in L\} \subseteq \mathbb{N}$$

est une union finie de progressions arithmétiques.

Démonstration. Soit σ une lettre de Σ. On définit le morphisme $\varphi : \Sigma^* \to \{\sigma\}^*$ par $\varphi(\alpha) = \sigma$ pour tout $\alpha \in \Sigma$. Il est évident que $\varphi(L) = \{\varphi(w) \mid w \in L\}$ est un langage régulier sur un alphabet unaire et $|L| = |\varphi(L)|$. On conclut grâce à la proposition précédente.

■

Définition I.3.11. Une partie $X \subseteq \mathbb{N}$ est dite *ultimement périodique* s'il existe $N \geq 0$ et $p > 0$ tels que

$$\forall n \geq N,\ n \in X \Leftrightarrow n + p \in X.$$

Le plus petit entier p satisfaisant une telle propriété est appelé la *période* de X et le plus petit N correspondant est parfois appelé la *prépériode*.

Proposition I.3.12. *Une partie $X \subseteq \mathbb{N}$ est ultimement périodique si et seulement si X est une union finie de progressions arithmétiques.*

Démonstration. Supposons qu'il existe $N \geq 0$ et $p > 0$ tels que

$$\forall n \geq N,\ n \in X \Leftrightarrow n + p \in X.$$

Dès lors, X s'exprime comme une union finie de progressions arithmétiques,

$$X = \Big(\bigcup_{\substack{x \in X \\ x < N}} \{x\} \Big) \cup \Big(\bigcup_{\substack{x \in X \\ N \leq x < N+p}} (x + \mathbb{N}.p) \Big).$$

Réciproquement, si

$$X = \bigcup_{i=1}^{n} (q_i + \mathbb{N}.p_i),$$

alors en posant $p = \mathrm{ppcm}_{i=1,\ldots,n}\, p_i$ et $N = \max_{i=1,\ldots,n} q_i$, il est clair que

$$\forall n \geq N,\ n \in X \Leftrightarrow n + p \in X.$$

■

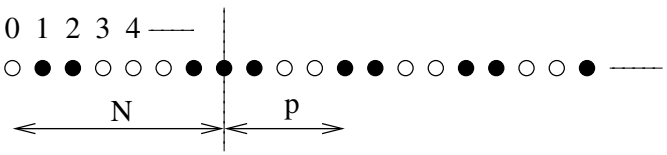


FIGURE I.4. Prépériode et période.

Exemple I.3.13. Utilisons la proposition I.3.9 pour montrer que le langage

$$L = \{a^{n^2} \mid n \in \mathbb{N}\}$$

n'est pas régulier. En effet, si ce langage était régulier, alors

$$|L| = \{n^2 \mid n \in \mathbb{N}\}$$

serait une union finie de progressions arithmétiques de la forme

$$A = \bigcup_{i=1}^{I} r_i + \mathbb{N}.s_i$$

avec au moins un des s_i non nul. Il est clair que la différence entre deux éléments consécutifs de A est majorée par une constante

$$C \leq \sup\{r_1, \ldots, r_I, s_1, \ldots, s_I\}$$

alors que la différence entre deux éléments consécutifs $(n+1)^2$ et n^2 de A est

$$2n + 1 \to \infty, \text{ si } n \to \infty.$$

On peut facilement étendre ce raisonnement à un langage de la forme

$$\{a^{P(n)} \mid n \in \mathbb{N}\}$$

où P est un polynôme à coefficients naturels de degré au moins deux.

Exemple I.3.14. Le langage $L = \{a^n \mid n \text{ premier}\}$ n'est pas régulier. En effet, s'il l'était, l'ensemble des nombres premiers devrait être ultimement périodique[18], disons de période p et de prépériode N. Donc pour tout $n \geq p+1$ suffisamment grand (i.e., $n \geq N$), l'intervalle $[n!+2, n!+p+1]$ devrait contenir un nombre premier. Or

$$n!+2,\ n!+3, \ldots,\ n!+p+1$$

sont tous des nombres composés.

Remarque I.3.15. En fait, plutôt que de parler de langages réguliers (associés à la notion d'expression régulière), on emploie souvent le vocable de *langages rationnels* sur Σ. L'ensemble de ces derniers, noté $\mathrm{Rat}(\Sigma^*)$ est défini comme étant le plus petit ensemble de langages contenant $\emptyset$, les langages finis et qui est stable pour les *opérations rationnelles* d'union, de concaténation (produit considéré sur le monoïde Σ^*) et d'étoile de Kleene. Autrement dit, un *langage rationnel* s'obtient à partir de langages finis en appliquant un nombre fini de fois des opérations rationnelles.

Cette notion de *rationnalité* s'étend aux parties d'un monoïde $(M, \cdot, 1_M)$ arbitraire (comme dans la preuve de la proposition I.3.9 où l'on a effectivement considéré le monoïde $(\mathbb{N}, +, 0)$). Ainsi, l'ensemble $\mathrm{Rat}(M)$ des *parties rationnelles* d'un monoïde $(M, \cdot, 1_M)$ est le plus petit ensemble de 2^M

[18] Ben Green et Terence Tao ont récemment (2004) démontré que l'ensemble des nombres premiers contient des progressions arithmétiques arbitrairement longues, i.e., pour tout k, il existe d tels que $p, p+d, \ldots, p+k\,d$ soient premiers.

contenant $\emptyset$, les parties finies de M et qui est stable pour les *opérations rationnelles* d'union, de produit (du monoïde) et d'étoile de Kleene où pour tout $A \subseteq M$, $A^* = \{a_1 \cdots a_p \mid p \geq 1,\ \forall 1 \leq i \leq p,\ a_i \in A\} \cup \{1_M\}$, i.e., A^* est le sous-monoïde de M engendré par A. On parle aussi de la *clôture rationnelle* des parties finies de M. Autrement dit, une partie rationnelle de M s'obtient à partir de sous-ensembles finis de M en appliquant un nombre fini de fois des opérations rationnelles. La proposition I.3.9 montre que $Rat(\mathbb{N})$ est exactement l'ensemble des parties ultimement périodiques de $\mathbb{N}$.

4. Exercices

4.1. Mots et langages.

Exercice I.4.1. On définit le miroir d'un mot $w \in \Sigma^*$, par récurrence sur la longueur de w, de la manière suivante : si $w = \varepsilon$, alors $w^R = \varepsilon$; sinon, il existe $\sigma \in \Sigma$ et $v \in \Sigma^*$ tels que $w = \sigma v$ et $w^R = v^R\sigma$. Montrer que cette définition est équivalente à celle qui suit : si $w = \varepsilon$, alors $w^R = \varepsilon$; sinon, il existe $\sigma \in \Sigma$ et $v \in \Sigma^*$ tels que $w = v\sigma$ et $w^R = \sigma v^R$. Démontrer que pour tous $u, v, w \in \Sigma^*$,

$$(w^R)^R = w \quad \text{et} \quad (uv)^R = v^R u^R.$$

Exercice I.4.2. Démontrer que pour tous $u, v, w \in \Sigma^*$, on a

$$uw = vw \Rightarrow u = v$$

et

$$wu = wv \Rightarrow u = v.$$

Exercice I.4.3. Soient x, y, u, v des mots sur un alphabet Σ tels que $xy = uv$. Démontrer que

- si $|x| > |u|$, alors il existe $w \neq \varepsilon$ tel que $x = uw$ et $v = wy$,
- si $|x| = |u|$, alors $x = u$ et $y = v$,
- si $|x| < |u|$, alors il existe $w \neq \varepsilon$ tel que $u = xw$ et $y = wv$.

Exercice I.4.4. Il est clair que pour tout mot w, $\#(\mathrm{Pref}(w)) = |w| + 1$. Quelles sont les bornes (inférieures et supérieures) exactes pour $\#(\mathrm{Fac}(w))$.

Exercice I.4.5. Soit $\Sigma = \{a, b\}$ un alphabet. Quels sont les mots $w \in \Sigma^*$ pour lesquels $w^2 = w^3$?

Exercice I.4.6. Soit $\Sigma = \{a, b\}$ un alphabet. Quels sont les mots $w \in \Sigma^*$ pour lesquels il existe $v \in \Sigma^*$ tel que $w^3 = v^2$?

Exercice I.4.7. Caractériser les mots w tels que $w = w^R$ (i.e., les palindromes).

Exercice I.4.8. Soit $L \subseteq \Sigma^*$ un langage. Si $L = L^R$, cela implique-t-il que tous les mots du langage sont des palindromes ? Justifier votre réponse et envisager également le cas particulier d'un alphabet Σ unaire.

Exercice I.4.9. Soit le langage $L \subseteq \{a, b\}^*$ défini récursivement par les trois conditions :

- $\varepsilon \in L$,
- si u appartient à L, alors $aaub$ appartient à L,
- un mot appartient à L seulement s'il peut être obtenu à partir de la première règle et d'un nombre fini d'applications de la deuxième règle.

Donner une définition implicite du langage L.

Exercice I.4.10. Soient Σ un alphabet et $L \subseteq \Sigma^*$ le langage défini récursivement par les trois conditions :

- $\varepsilon \in L$ et pour tout $\sigma \in \Sigma$, $\sigma \in L$,
- si w appartient à L, alors pour tout $\sigma \in \Sigma$, $\sigma w \sigma$ appartient à L,
- un mot appartient à L seulement s'il peut être obtenu à partir de la première règle et d'un nombre fini d'applications de la deuxième règle.

Quel est ce langage L ?

Exercice I.4.11. Soit $L \subseteq \Sigma^*$ un langage. Démontrer que

$$(L^*)^* = L^*, \quad L^*L^* = L^*, \quad L^*L \cup \{\varepsilon\} = L^* = LL^* \cup \{\varepsilon\}.$$

A-t-on toujours $LL^* = L^*$?

Exercice I.4.12. Soient L, M, N des langages sur un alphabet Σ. Montrer que $L(M \cap N) \subseteq LM \cap LN$. Montrer qu'en général, l'autre inclusion est fausse.

Exercice I.4.13. Soient $L = \{aa, bb\}$ et $M = \{\varepsilon, b, ab\}$.

- Enumérer les mots de LM et $L \sqcup\!\sqcup M$.
- Enumérer les mots de M^* de longueur au plus trois.
- Combien de mots de longueur 6 possède le langage L^* ?
- Combien de mots de longueur $n \in \mathbb{N}$ possède le langage L^* ?

Exercice I.4.14. Soient L et M deux langages finis. A-t-on toujours

$$\#L.\#M = \#(LM) \quad ?$$

Justifier votre réponse.

Exercice I.4.15. Soient des alphabets disjoints Σ et Γ. Pour $m, n \in \mathbb{N}$, on note

$$t(m, n) := \#(u \sqcup\!\sqcup v), \quad u \in \Sigma^m,\ v \in \Gamma^n.$$

Montrer que

$$t(m, n) = t(m, n-1) + t(m-1, n), \quad m, n > 0$$

et que $t(m, 0) = t(0, m) = 1$. Utiliser cette formule pour en déduire la valeur de

$$\#(abba \sqcup\!\sqcup cd).$$

Pour calculer $t(m,n)$ au moyen de la formule donnée ci-dessus, combien d'étapes sont nécessaires ?

Exercice I.4.16. Soit $\Sigma = \{a,b\}$ un alphabet binaire. Un mot $w \in \Sigma^*$ est *sans carré* s'il ne contient aucun facteur de la forme xx avec x un mot non vide. Enumérer tous les mots sans carré de Σ^*. (Que se passe-t-il dans le cas d'un alphabet contenant plus de deux lettres ?)

Exercice I.4.17. Soit Σ un alphabet. Un mot $w \in \Sigma$ est *primitif* si l'équation

$$w = u^i, \quad u \in \Sigma^*$$

n'est satisfaite pour aucun exposant $i \geq 2$. On appelle *racine primitive* de w, le plus petit mot $u \in \Sigma^*$ tel que $w = u^i$, pour un $i \geq 1$. Démontrer qu'un mot est primitif si et seulement si il est égal à sa propre racine primitive.

Exercice I.4.18. Deux mots u et v sur Σ sont *conjugués* s'il existe $x, y \in \Sigma^*$ tels que $u = xy$ et $v = yx$. Enumérer tous les conjugués du mot $abbaa$. Montrer que la relation "être conjugué" est une relation d'équivalence sur Σ^*. Démontrer que si w est primitif, alors ses conjugués le sont aussi.

Exercice I.4.19. Soit l'alphabet $\{\rightarrow, \leftarrow, \uparrow, \downarrow\}$ où chaque flèche représente un déplacement d'une unité dans le plan muni d'un repère orthonormé. Caractériser l'ensemble des mots correspondant à un déplacement du point de coordonnées $(0,0)$ au point de coordonnées $(1,1)$. Même question, mais cette fois, on se restreint au déplacements dans le premier quadrant (on ne peut se trouver en un point dont une des coordonnées serait strictement négative).

4.2. Expressions régulières.

Exercice I.4.20. Donner une expression régulière du langage formé des mots de longueur au moins 2 sur $\{a,b\}$ pour lesquels tous les a éventuellement présents précèdent les b (éventuellement présents).

Exercice I.4.21. Même question que la précédente, mais cette fois, le mot vide appartient au langage.

Exercice I.4.22. Donner une expression régulière du langage formé des mots sur $\{a,b\}$ qui ne contiennent pas le facteur ba.

Exercice I.4.23. Donner une expression régulière du langage formé des mots sur $\{a,b\}$ qui contiennent simultanément le facteur aa et le facteur bb.

Exercice I.4.24. Donner une expression régulière du langage formé des mots sur $\{a,b\}$ qui contiennent le facteur aa ou le facteur bb, mais pas ces deux facteurs simultanément.

Exercice I.4.25. Donner une expression régulière du langage formé des mots sur $\{a,b\}$ qui contiennent exactement deux fois le facteur aa. (Suggestion : attention au facteur aaa).

Exercice I.4.26. Donner une expression régulière du langage formé des mots sur $\{a, b, c\}$ qui ne contiennent pas deux a consécutifs.

Exercice I.4.27. Donner une expression régulière du langage formé des mots sur $\{a, b\}$ qui contiennent le facteur aa exactement une fois.

Exercice I.4.28. Donner une expression régulière du langage formé des mots sur $\{a, b, c\}$ qui débutent par a, contiennent exactement deux b et se terminent par cc.

Exercice I.4.29. Donner une expression régulière du langage formé des mots sur $\{a, b, c\}$ qui contiennent un nombre de a divisible par 3.

Exercice I.4.30. Donner une expression régulière du langage formé des mots sur $\{a, b\}$ de longueur impaire et qui contiennent le facteur bb.

Exercice I.4.31. Donner une expression régulière du langage formé des mots sur $\{a, b\}$ ayant au plus trois a.

Exercice I.4.32. Donner une expression régulière du langage formé des mots sur $\{a, b, c\}$ qui contiennent un nombre impair d'occurences du facteur ab.

Exercice I.4.33. Donner une expression régulière du langage formé des représentations en base 3 des nombres pairs.

Exercice I.4.34. Donner une expression régulière du langage formé des représentations en base 10 des nombres multiples de 5.

Exercice I.4.35. Soit Σ un alphabet. Dans les expressions régulières données ci-dessous, on s'autorise à utiliser l'expression φ^+ (avec $\varphi \in \mathcal{R}_\Sigma$) qui est telle que $\mathcal{L}(\varphi^+) = (\mathcal{L}(\varphi))^+ = \cup_{i>0}(\mathcal{L}(\varphi))^i$. Démontrer que

- $(ba)^+(a^*b^* + a^*) = (ba)^* b\, a^+(b^* + e)$,
- $b^+(a^*b^* + e)b = b(b^*a^* + e)b^+$,
- $(a + b)^* = (a + b)^*b^*$,
- $(a + b)^* = (a^* + ba^*)^*$,
- $(a + b)^* = (b^*(a + e)b^*)^*$.

Exercice I.4.36. Soit $\Sigma = \{a, b\}$. Vérifier que

$$(ab)^+ = (a\Sigma^* \cap \Sigma^*b) \setminus (\Sigma^*aa\Sigma^* + \Sigma^*bb\Sigma^*).$$

4.3. Langages réguliers sur un alphabet unaire.

Exercice I.4.37. Exprimer $|\mathcal{L}(\varphi)|$ comme une union finie de progressions arithmétiques lorsque

- $\varphi = (ab)^* + bbb$,
- $\varphi = (a(ba^*) + a^*)^*$,
- $\varphi = (ab)(ac(a + b))^*$,
- $\varphi = ab(bbc)^*$.

Exercice I.4.38. Construire une expression régulière φ telle que

- $|\mathcal{L}(\varphi)| = 5 + \mathbb{N}.3$,
- $|\mathcal{L}(\varphi)| = \mathbb{N}.7 \cup (4 + \mathbb{N}.5)$.

Exercice I.4.39. Le langage $\{a^{n^3+2n+1} \mid n \in \mathbb{N}\}$ est-il régulier ? Justifier.

Exercice I.4.40. Le langage $\{a^{2n+1} \mid n \in \mathbb{N}\}$ est-il régulier ? Justifier.

Exercice I.4.41. Le langage $\{a^n \mid n \in \mathcal{P}\}$ où $\mathcal{P}$ désigne l'ensemble des nombres premiers est-il régulier ? Justifier.

Remarque I.4.42. Le résultat obtenu à l'exercice précédent n'est en rien incompatible avec le célèbre théorème de Dirichlet qui stipule que si a et b sont premiers entre eux (i.e., $\text{pgcd}(a, b) = 1$), alors la progression arithmétique $a + \mathbb{N}.b$ contient une infinité de nombres premiers.

CHAPITRE II

Automates

Nous avons vu au premier chapitre que les expressions régulières permettent de *générer* ce que l'on a décidé d'appeler les langages réguliers. Les automates que nous allons introduire ici sont des "machines" permettant de *reconnaître* exactement ces langages. En d'autres termes, l'ensemble des langages acceptés par automate fini coïncide avec l'ensemble des langages réguliers.

1. Automates finis déterministes

Définition II.1.1. Un *automate fini déterministe* (ou AFD) est la donnée d'un quintuple

$$\mathcal{A} = (Q, q_0, F, \Sigma, \delta)$$

où

- Q est un ensemble fini dont les éléments sont les *états* de $\mathcal{A}$,
- $q_0 \in Q$ est un état privilégié appelé *état initial*,
- $F \subseteq Q$ désigne l'ensemble des *états finals*,
- Σ est l'alphabet de l'automate,
- $\delta : Q \times \Sigma \to Q$ est la *fonction de transition* de $\mathcal{A}$.

Nous supposerons que δ est une fonction totale, i.e., que δ est défini pour tout couple $(q, \sigma) \in Q \times \Sigma$ (on parle alors d'AFD *complet*).

Nous représentons un AFD $\mathcal{A}$ de la manière suivante. Les états de $\mathcal{A}$ sont les sommets d'un graphe orienté et sont représentés par des cercles. Si $\delta(q, \sigma) = q'$, $q, q' \in Q$, $\sigma \in \Sigma$, alors on trace un arc orienté de q vers q' et de label σ

$$q \xrightarrow{\sigma} q'.$$

Les états finals sont repérés grâce à un double cercle et l'état initial est désigné par une flèche entrante sans label. Enfin, si deux lettres σ et σ' sont telles que $\delta(q, \sigma) = q'$ et $\delta(q, \sigma') = q'$, on s'autorise à dessiner un unique arc portant deux labels séparés par une virgule,

$$q \xrightarrow{\sigma,\sigma'} q'.$$

Cette convention s'adapte aisément à plus de deux lettres.

Exemple II.1.2. L'automate $\mathcal{A} = (Q, q_0, F, \Sigma, \delta)$ où $Q = \{1, 2, 3\}$, $q_0 = 1$, $F = \{1, 2\}$, $\Sigma = \{a, b\}$ et où la fonction de transition est donnée par

δ	a	b
1	1	2
2	1	3
3	3	2

est représenté à la figure II.1.

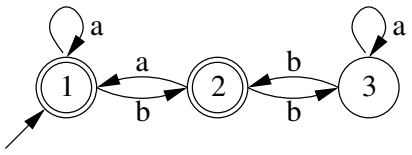


FIGURE II.1. Un AFD.

Définition II.1.3. Soit $\mathcal{A} = (Q, q_0, F, \Sigma, \delta)$ un AFD. On étend naturellement la fonction de transition δ à $Q \times \Sigma^*$ de la manière suivante :

$$\delta(q, \varepsilon) = q$$

et

$$\delta(q, \sigma w) = \delta(\delta(q, \sigma), w), \ \ \sigma \in \Sigma, w \in \Sigma^*.$$

Le *langage accepté par* $\mathcal{A}$ est alors

$$L(\mathcal{A}) = \{w \in \Sigma^* \mid \delta(q_0, w) \in F\}.$$

Si $w \in L(\mathcal{A})$, on dit encore que $\mathcal{A}$ accepte le mot w (ou que w est accepté par $\mathcal{A}$).

Ainsi, le rôle fondamental d'un automate est d'accepter ou de rejeter des mots. Un automate partitionne l'ensemble des mots sur Σ en deux sous-ensembles

$$L(\mathcal{A}) \ \text{ et } \ \Sigma^* \setminus L(\mathcal{A}).$$

Exemple II.1.4. Si on poursuit l'exemple précédent, l'automate $\mathcal{A}$ de la figure II.1 accepte le mot *abbab* car on a, partant de l'état initial, le parcours suivant au sein de $\mathcal{A}$:

$$1 \xrightarrow{a} 1 \xrightarrow{b} 2 \xrightarrow{b} 3 \xrightarrow{a} 3 \xrightarrow{b} 2 \in F.$$

Par contre, *bba* n'est pas accepté par $\mathcal{A}$ car

$$1 \xrightarrow{b} 2 \xrightarrow{b} 3 \xrightarrow{a} 3 \notin F.$$

Exemple II.1.5. L'automate $\mathcal{A}$ représenté à la figure II.2 accepte exactement le langage formé des mots sur l'alphabet $\{a, b\}$ et contenant un nombre impair de a.

$$L(\mathcal{A}) = \{w \in \{a, b\}^* \ : \ |w|_a \equiv 1 \pmod 2\}.$$

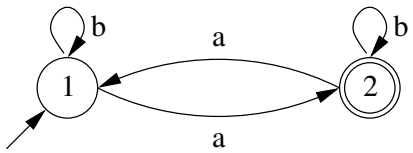


FIGURE II.2. Un automate fini déterministe.

Remarque II.1.6. Pour simplifier les notations, on s'autorise à écrire

$$q.w$$

au lieu de $\delta(q, w)$ si aucune confusion n'est possible.

Définition II.1.7. A tout mot w de Σ^* correspond une unique suite d'états de $\mathcal{A}$ correspondant au chemin parcouru lors de la lecture de w dans $\mathcal{A}$. Cette suite s'appelle l'*exécution*[1] de w sur $\mathcal{A}$. Ainsi, si $w = w_0 \cdots w_k$, avec $w_i \in \Sigma$, alors l'exécution de w sur $\mathcal{A}$ est

$$(q_0,\, q_0.w_0,\, q_0.w_0w_1, \ldots,\, q_0.w_0 \cdots w_{k-1},\, q_0.w_0 \cdots w_k).$$

Remarque II.1.8. On aurait pu aussi introduire des automates "infinis" en n'imposant pas la restriction à l'ensemble d'états Q d'être fini (mais nous supposerons quand même que l'alphabet reste fini). Les notions d'exécution ou de langage accepté se transposent sans peine à ce cadre plus général. (Nous verrons un peu plus loin que la notion d'automate "minimal" ne spécifie *a priori* rien sur le caractère fini de l'ensemble d'états.)

2. Automates non déterministes

Le modèle d'automate fini non déterministe généralise le cas des AFD. Comme nous le verrons bientôt, le non-déterminisme permet une plus grande souplesse bien utile dans certaines situations.

Définition II.2.1. Un *automate fini non déterministe* (AFND) est la donnée d'un quintuple

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

où

- Q est un ensemble fini dont les éléments sont les *états* de $\mathcal{A}$,
- $I \subseteq Q$ est l'ensemble des *états initiaux*,
- $F \subseteq Q$ désigne l'ensemble des *états finals*,
- Σ est l'alphabet de l'automate,
- $\Delta \subset Q \times \Sigma^* \times Q$ est une *relation de transition* (qu'on supposera finie).

On peut dès à présent noter plusieurs différences entre les AFD et AFND. Dans le cas non déterministe, il est possible d'avoir plus d'un état intial; les labels des arcs ne sont plus nécessairement des lettres mais bien des mots

[1] La terminologie anglo-saxonne consacre le mot "*run*".

de Σ^* et enfin, on n'a plus une fonction de transition mais une relation de transition. Pour représenter les AFND, nous utilisons les mêmes conventions que pour les AFD.

Exemple II.2.2. L'automate de la figure II.3 est un AFND ayant 1 et 3

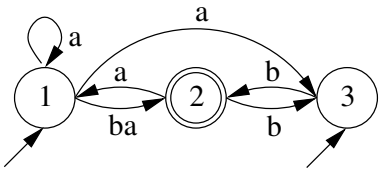


FIGURE II.3. Un AFND.

comme états initiaux, 2 comme état final et la relation de transition est

$$\Delta = \{(1, a, 1), (1, a, 3), (1, ba, 2), (2, a, 1), (2, b, 3), (3, b, 2)\}$$

Définition II.2.3. Un mot $w = w_1 \cdots w_k$ est *accepté* par un AFND $\mathcal{A} = (Q, I, F, \Sigma, \Delta)$ s'il existe $q_0 \in I$, $\ell \in \mathbb{N} \setminus \{0\}$, $v_1, \ldots, v_\ell \in \Sigma^*$, $q_1, \ldots, q_\ell \in Q$ tels que

$$(q_0, v_1, q_1), (q_1, v_2, q_2), \ldots, (q_{\ell-1}, v_\ell, q_\ell) \in \Delta,$$
$$w = v_1 \cdots v_\ell \quad \text{et} \quad q_\ell \in F.$$

En d'autres termes, cette condition signifie qu'il existe un chemin dans le graphe associé à $\mathcal{A}$ débutant dans un état initial, de label w et se terminant dans un état final. Naturellement, le langage *accepté* par un AFND $\mathcal{A}$ est l'ensemble des mots acceptés par $\mathcal{A}$ et se note encore $L(\mathcal{A})$. Enfin, deux AFND $\mathcal{A}$ et $\mathcal{B}$ sont dits *équivalents* si $L(\mathcal{A}) = L(\mathcal{B})$.

Exemple II.2.4. Si nous poursuivons l'exemple II.2.2, le mot ab est accepté car $1 \in I$, $(1, a, 3) \in \Delta$, $(3, b, 2) \in \Delta$ et $2 \in F$. Ceci se note schématiquement,

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

A un mot, il peut correspondre plus d'un chemin. Par exemple, au mot baa, il correspond les chemins

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

et

$$1 \xrightarrow{ba} 2 \xrightarrow{a} 1.$$

Ce sont les trois seules possibilités partant d'un état initial. Le mot baa n'est donc pas accepté par l'automate.

Remarque II.2.5. Dans la définition d'un AFND, rien n'empêche d'avoir des transitions "vides" du type

$$(q, \varepsilon, q') \in \Delta.$$

On parle parfois de ε-transitions. En particulier, on suppose implicitement que pour tout état q d'un AFND, on a

$$(q, \varepsilon, q) \in \Delta.$$

Exemple II.2.6. Considérons l'AFND suivant. Cet automate accepte le

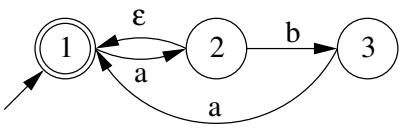


FIGURE II.4. Un AFND avec ε-transitions.

mot a car on a le chemin

$$1 \xrightarrow{a} 2 \xrightarrow{\varepsilon} 1 \in F.$$

Observons encore qu'il n'est pas possible depuis l'état initial de lire des mots débutant par b. Donc, contrairement à la situation déterministe où, à chaque mot correspondait exactement une exécution, ici, on peut avoir pour un mot donné plus d'un chemin dans le graphe, voire même aucun.

Définition II.2.7. Un AFND $\mathcal{A} = (Q, I, F, \Sigma, \Delta)$ est qualifié d'*élémentaire* si pour tout $(q, w, q') \in \Delta$,

$$|w| \leq 1.$$

Comme le montre le lemme suivant, on peut dans le cadre des AFND se restreindre au cas d'automates élémentaires. En effet, tout AFND est équivalent à un AFND élémentaire.

Lemme II.2.8. *Tout langage accepté par un AFND est accepté par un AFND élémentaire.*

Démonstration. Soit $\mathcal{A} = (Q, I, F, \Sigma, \Delta)$ un AFND. S'il n'est pas élémentaire, il existe au moins un mot $w = w_1 \cdots w_k$ ($k \geq 2$, $w_i \in \Sigma$) et des états q, q' tels que

$$(q, w, q') \in \Delta.$$

Considérons de nouveaux états $q_1, \ldots, q_{k-1}$ n'appartenant pas Q. Il est clair que l'automate $\mathcal{B} = (Q', I, F, \Sigma, \Delta')$ où

$$Q' = Q \cup \{q_1, \ldots, q_{k-1}\}$$

et

$$\begin{aligned} \Delta' \;=\; & \big(\Delta \setminus \{(q, w, q')\}\big) \\ & \bigcup \big\{(q, w_1, q_1), (q_1, w_2, q_2), \ldots, (q_{k-2}, w_{k-1}, q_{k-1}), (q_{k-1}, w_k, q')\big\} \end{aligned}$$

est tel que $L(\mathcal{A}) = L(\mathcal{B})$. En répétant cette procédure pour chaque transition $(q, w, q') \in \Delta$ telle que $|w| > 1$, on obtient (en un nombre fini d'étapes) un automate élémentaire acceptant le même langage.

■

Exemple II.2.9. Appliquons la méthode de construction donnée dans la preuve du lemme précédent à un exemple. Soit l'AFND non élémentaire $\mathcal{A}$ représenté à la figure II.5.

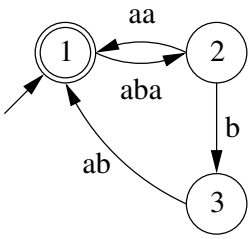


FIGURE II.5. Un AFND non élémentaire $\mathcal{A}$.

On obtient alors un automate élémentaire équivalent à $\mathcal{A}$ représenté à la figure II.6, les états supplémentaires ne portant pas de numéro.

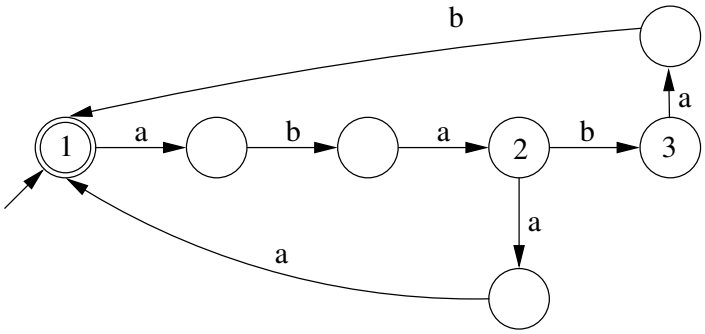


FIGURE II.6. Un AFND élémentaire équivalent à $\mathcal{A}$.

Remarque II.2.10. Soit $\mathcal{A} = (Q, I, F, \Sigma, \Delta)$ un AFND. Si $R \subseteq Q$ et $w \in \Sigma^*$, on note

$$R.w$$

Il s'agit d'une extension de la notation $q.w$ introduite à la remarque II.1.6

l'ensemble des états atteints à partir des états de R en lisant w. Par exemple, avec l'automate de la figure II.4, $\{1\}.a = \{1, 2\}$ car

$$1 \xrightarrow{a} \mathbf{2} \quad \text{et} \quad 1 \xrightarrow{a} 2 \xrightarrow{\varepsilon} \mathbf{1}.$$

On a aussi pour cet automate, $\{1\}.b = \emptyset$.

Dans le cas particulier de $w = \varepsilon$, on a toujours

$$R.\varepsilon \supseteq R.$$

En effet, on suppose implicitement que pour tout état q, $(q, \varepsilon, q) \in \Delta$ (cf. remarque II.2.5). Autrement dit, $R.\varepsilon$ est l'ensemble des états atteints depuis les états de R sans lire de lettres.

Si L est un langage sur Σ, on pose

$$R.L = \bigcup_{w \in L} R.w.$$

Le résultat suivant stipule que tout AFND est équivalent à un AFD. En d'autres termes, lorsqu'on s'intéresse à l'acceptation des mots d'un langage, un AFND n'est pas "*plus puissant*" qu'un AFD.

Proposition II.2.11 (Rabin et Scott[2]). *Tout langage accepté par un AFND est accepté par un AFD.*

Démonstration. Vu le lemme II.2.8, on peut supposer disposer d'un AFND élémentaire $\mathcal{A} = (Q, I, F, \Sigma, \Delta)$. Considérons l'automate fini déterministe

$$\mathcal{B} = (2^Q, Q_0, \mathfrak{F}, \Sigma, \beta)$$

ayant comme ensemble d'états, l'ensemble des parties[3] de Q et tel que

- l'état initial Q_0 est égal à $I.\varepsilon$, i.e., à l'ensemble des états de $\mathcal{A}$ atteints à partir des états initiaux sans consommer de lettres;
- l'ensemble des états finals est

  $$\mathfrak{F} = \{G \subseteq Q \mid G \cap F \neq \emptyset\},$$

  i.e., les états finals de $\mathcal{B}$ sont les parties de Q contenant au moins un état final;
- si $G \subseteq Q$ est un état de $\mathcal{B}$ et w un mot non vide sur Σ, alors on définit la fonction de transition de $\mathcal{B}$ par

  $$\beta(G, w) = G.w.$$

  De plus, on pose $\beta(G, \varepsilon) = G$.

$G.w$ désigne l'ensemble des états atteints à partir des états de G en lisant w.

Tout d'abord, pour vérifier qu'il s'agit bien d'un AFD, il suffit de montrer que β est bien une fonction de transition, i.e., que pour tous $u, v \in \Sigma^*$,

$$\beta(G, uv) = \beta(\beta(G, u), v).$$

Si au moins un des deux mots u ou v est nul, la conclusion est immédiate. Sinon, nous devons montrer que

$$G.uv = (G.u).v.$$

Il est clair que le membre de droite est inclus dans le membre de gauche. L'autre inclusion résulte du fait que l'automate est élémentaire. En effet, cette propriété est indispensable. En guise d'illustration, l'automate représenté à la figure II.7 n'est pas élémentaire et $\{1\}.ab = \{3, 4\}$, $\{1\}.a =$

[2] M. O. Rabin, D. Scott, Finite automata and their decision problems, *IBM J. of Research and Development* **3** (1959), 114–125.

[3] $\mathcal{B}$ est fini car $\#2^Q = 2^{\#Q}$ et $\mathcal{A}$ est fini.

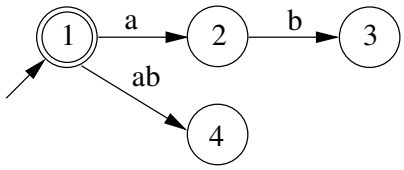


FIGURE II.7. Un automate non élémentaire.

$\{2\}$ et $(\{1\}.a).b = \{2\}.b = 3$ donc

$$\{1\}.ab \not\subset (\{1\}.a).b.$$

Pour un AFND élémentaire, les arcs ayant des labels de longueur au plus un, une telle situation ne peut se produire.

Il nous reste à montrer que $L(\mathcal{A}) = L(\mathcal{B})$. On procède par double inclusion. Soit w appartenant à $L(\mathcal{A})$. Cela signifie qu'il existe dans $\mathcal{A}$ un chemin de label w débutant dans un état initial de I (et même dans un état de $I.\varepsilon$) et aboutissant dans un état final de F. En d'autres termes, par définition de Q_0,

$$Q_0.w \cap F \neq \emptyset$$

et $\beta(Q_0, w)$ appartient donc à $\mathfrak{F}$.

Soit w appartenant à $L(\mathcal{B})$. Dans $\mathcal{B}$, le chemin de label w débutant dans Q_0 aboutit dans un état final de $\mathfrak{F}$, i.e.,

$$\beta(Q_0, w) \in \mathfrak{F}.$$

Donc

$$(I.\varepsilon).w \cap F \neq \emptyset$$

ce qui signifie que w est accepté par $\mathcal{A}$.

■

Remarque II.2.12. La démonstration précédente nous fournit une méthode (un algorithme) permettant de rechercher un automate déterministe acceptant exactement le même langage qu'un AFND donné. On parle souvent de la *construction par sous-ensembles* (en anglais, "*subset construction*"). Comme nous allons le montrer sur des exemples, il est inutile de considérer toutes les parties de Q. Seuls les états qui peuvent être atteints depuis Q_0 méritent d'être considérés.

Exemple II.2.13. Soit l'AFND représenté à la figure II.8. On remarque qu'il est élémentaire. S'il ne l'était pas, il faudrait tout d'abord le rendre élémentaire. On construit le tableau suivant de proche en proche, en l'initialisant avec $I.\varepsilon$ qui est ici

$$\{1\}.\varepsilon = \{1\}.$$

A chaque étape, pour un sous-ensemble X d'états non encore traité, on détermine les valeurs de $X.\sigma$ pour tout $\sigma \in \Sigma$. La construction se termine

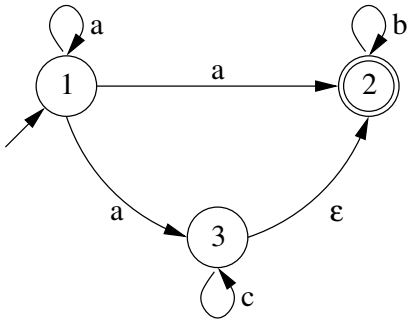


FIGURE II.8. Un automate fini non déterministe.

une fois que tous les sous-ensembles d'états apparaissant ont été pris en compte.

X	$X.a$	$X.b$	$X.c$
$\{1\}$	$\{1,2,3\}$	$\emptyset$	$\emptyset$
$\{1,2,3\}$	$\{1,2,3\}$	$\{2\}$	$\{2,3\}$
$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$
$\{2\}$	$\emptyset$	$\{2\}$	$\emptyset$
$\{2,3\}$	$\emptyset$	$\{2\}$	$\{2,3\}$

La construction d'un tel tableau peut être facilitée en établissant au préalable la table de transition de l'automate. Ici, pour $\mathcal{A}$, cette table est

q	$q.a$	$q.b$	$q.c$
1	$\{1,2,3\}$		
2		2	
3		2	$\{2,3\}$

Si on pose $A = \{1\}$, $B = \{1,2,3\}$, $C = \emptyset$, $D = \{2\}$ et $E = \{2,3\}$, alors on a l'automate représenté à la figure II.9. Bien évidemment, les états finals de

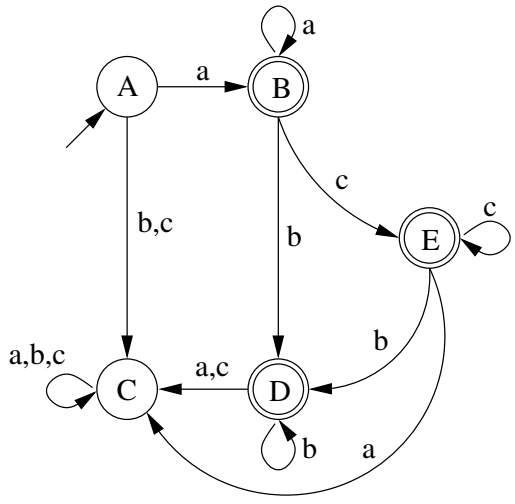


FIGURE II.9. AFD équivalent à l'AFND de la figure II.8.

cet automate sont les sous-ensembles d'états de $\mathcal{A}$ qui contiennent 2 (le seul état final de $\mathcal{A}$).

Exemple II.2.14. Considérons un AFND (élémentaire) acceptant, comme il est facile de le vérifier, le langage $a(ba)^* \cup a^*$. Cet automate est représenté à la figure II.10. Ici, $\{1\}.\varepsilon = \{1, 2, 4\}$. Ainsi, la construction du tableau

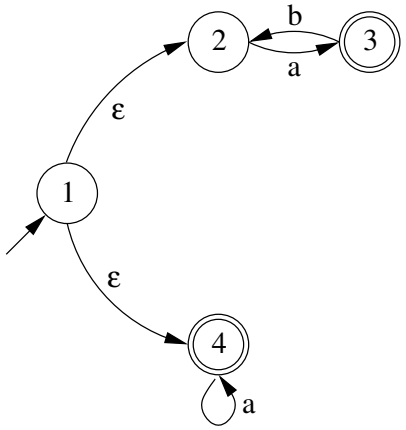


FIGURE II.10. un ANFD acceptant $a(ba)^* \cup a^*$.

donne

X	$X.a$	$X.b$
$A = \{1, 2, 4\}$	$\{3, 4\}$	$\emptyset$
$B = \{3, 4\}$	$\{4\}$	$\{2\}$
$C = \emptyset$	$\emptyset$	$\emptyset$
$D = \{4\}$	$\{4\}$	$\emptyset$
$E = \{2\}$	$\{3\}$	$\emptyset$
$F = \{3\}$	$\emptyset$	$\{2\}$

et on trouve l'automate de la figure II.11.

Remarque II.2.15. Il est clair que tout AFD est un cas particulier d'AFND. Par conséquent, tout langage accepté par un AFD $\mathcal{A}$ est trivialement accepté par un AFND (à savoir $\mathcal{A}$ lui-même). De la proposition II.2.11, nous concluons donc que les langages acceptés par les AFD et les AFND coïncident. Nous pourrons dès lors, par la suite, parler d'un langage accepté par un automate fini (sans autre précision).

2.1. A propos de l'explosion exponentielle. Le nombre d'états peut croître de manière exponentielle lorsqu'on rend déterministe un AFND. Dans certaines situations, cette explosion du nombre d'états est inévitable et ce, même dans le cas d'un alphabet unaire.

Définition II.2.16. On définit un AFND $\mathcal{A}_k$ sur un alphabet unaire $\{a\}$ comme suit. Cet automate possède un unique état initial à partir duquel on peut se déplacer dans k boucles disjointes. Pour $i = 1, \ldots, k$, la i-ième

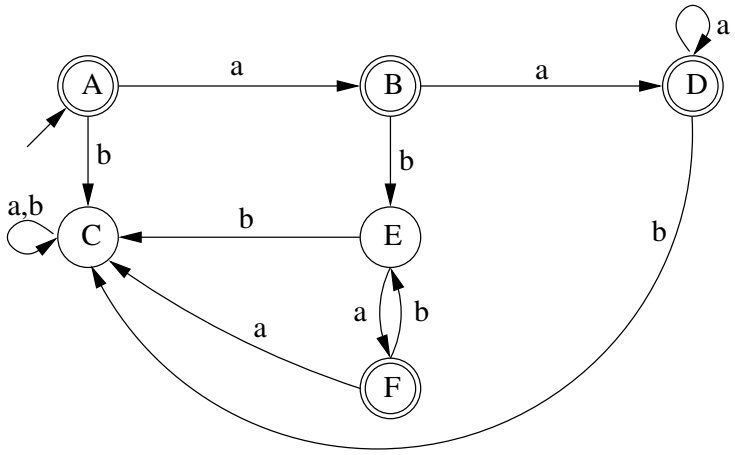


FIGURE II.11. un AFD acceptant $a(ba)^* \cup a^*$.

boucle est un cycle de longueur p_i où p_i représente le i-ème nombre premier. Les états finals sont l'état initial et un état par cycle, de manière telle que le langage accepté par $\mathcal{A}_k$ soit

$$L(\mathcal{A}_k) = \{a^n \mid n \in \mathbb{N}.p_1 \cup \mathbb{N}.p_2 \cup \ldots \cup \mathbb{N}.p_k\} =: L_k.$$

Un exemple, dans le cas $k = 3$, est représenté à la figure II.12.

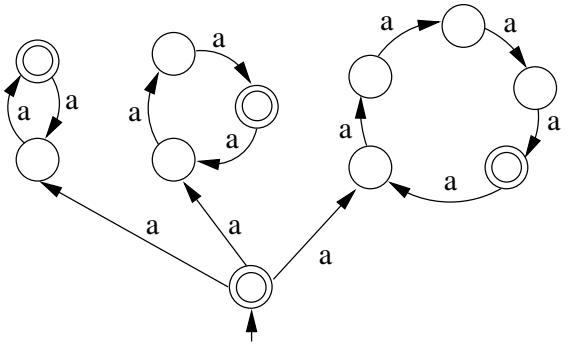


FIGURE II.12. L'automate $\mathcal{A}_3$.

Proposition II.2.17. *Le langage L_k accepté par l'AFND $\mathcal{A}_k$ possédant $1 + p_1 + p_2 + \cdots + p_k$ états est accepté par un AFD ayant $N = p_1 p_2 \cdots p_k$ états et aucun AFD ayant moins de N états n'accepte ce langage.*

Démonstration. Tout d'abord, un AFD composé d'un unique cycle de longueur N convient. En effet, si on numérote les états de cet automate $0, \ldots, N-1$, alors l'état initial est 0 et les états finals correspondent aux indices i pour lesquels il existe $j \in \{1, \ldots, k\}$ tel que

$$i \equiv 0 \pmod{p_j}.$$

Il est clair qu'un tel AFD accepte exactement L_k. Par exemple, dans le cas où $k = 3$, on a l'AFD représenté à la figure II.13. Dans cet automate, est final tout état dont l'indice est multiple de 2, 3 ou 5.

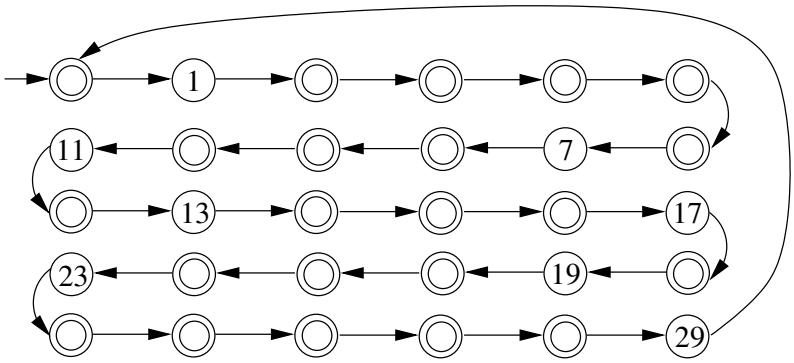


FIGURE II.13. Un AFD acceptant L_3.

A présent, supposons que $\mathcal{B}$ est un AFD acceptant L_k et possèdant moins de N états. Puisque l'alphabet est unaire, cet automate est de la forme suivante :

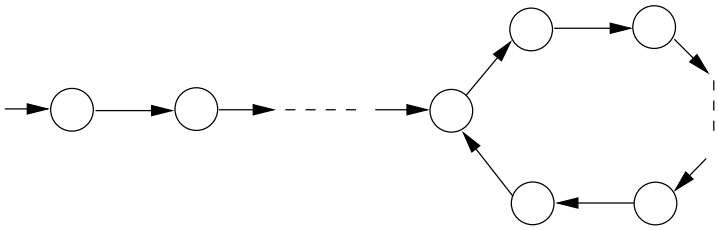


FIGURE II.14. Un AFD sur un alphabet unaire.

On parle parfois de manière imagée, d'automate "*poële à frire*" (*frying pan automaton*). Le cycle est de longueur $\ell \geq 1$ et le chemin menant au cycle est de longueur $c \geq 0$. Par hypothèse, on a $\ell < N$. Par conséquent, il existe $j \in \{1, \ldots, k\}$ tel que p_j soit premier avec ℓ (en effet, sinon, $p_1, \ldots, p_k$ apparaîtraient tous dans la décomposition en facteurs premiers de ℓ et dès lors, on aurait $\ell \geq N$). Par le théorème de Bezout, il existe $f, g \in \mathbb{Z}$ tels que

$$f\, p_j + g\, \ell = 1.$$

En d'autres termes,

$$f\, p_j \equiv 1 \pmod{\ell}$$

et donc

$$\{m\, p_j \pmod{\ell} \mid m \in \mathbb{N}\} = \{0, \ldots, \ell - 1\}.$$

On a bien sûr aussi, quel que soit c,

$$\{m\, p_j - c \pmod{\ell} \mid m \in \mathbb{N}\} = \{0, \ldots, \ell - 1\}. \tag{3}$$

Cela signifie que pour accepter les mots de la forme a^n avec $n = c + c'$ multiple de p_j, le cycle de l'automate $\mathcal{B}$ doit avoir tous ses états finals. En effet, c' est de la forme $m\,p_j - c$, $m \in \mathbb{N}$, et donc, vu (3), la lecture d'un tel mot a^n peut aboutir dans un état quelconque du cycle. Dès lors, cet automate $\mathcal{B}$ accepte tout mot a^t pour $t \geq c$ et en particulier, il accepte les mots de longueur

$$p_{k+1}^n$$

pour n suffisamment grand. Or il est facile de voir que ces derniers mots n'appartiennent pas à L_k. En effet, les puissances de p_{k+1} ne peuvent être multiples de $p_1, \ldots, p_k$. Ainsi, l'automate $\mathcal{B}$ ne peut accepter L_k.

■

Remarque II.2.18. E. Bach et J. Shallit montrent[4] que

$$\sum_{i=1}^{k} p_i \sim \frac{1}{2} k^2 \log k$$

alors qu'une minoration grossière donne

$$p_1 \cdots p_k \geq 2^k.$$

3. Stabilité des langages acceptés par automate

Nous montrons tout d'abord que l'ensemble des langages acceptés par un automate fini est stable pour les opérations rationnelles (i.e., l'union, la concaténation et l'étoile de Kleene).

Proposition II.3.1. *Si L et M sont les langages acceptés par deux automates finis, alors $L \cup M$ est aussi accepté par un automate fini.*

Démonstration. Représentons symboliquement un automate fini $\mathcal{A}$ par le schéma donné à la figure II.15. On représente uniquement les états finals

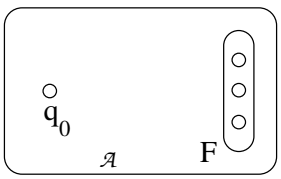


FIGURE II.15. Représentation symbolique d'un automate.

et l'état initial. Pour ne pas alourdir le dessin, on ne représente aucune des transitions. De plus, rien n'empêche l'état initial d'être final. Considérer un seul état initial n'est pas en soi une restriction. En effet, on peut ajouter un nouvel état q_0 à un automate. Cet état est considéré comme le seul état initial et on place une ε-transition depuis q_0 vers chacun des anciens états

[4]cf. E. Bach et J. Shallit, *algorithmic number theory, Vol.1, Efficient algorithms*, Foundations of Computing Series, MIT Press, 1996.

initiaux (qui perdent alors ce statut particulier). De la sorte, on obtient un automate équivalent avec un seul état initial.

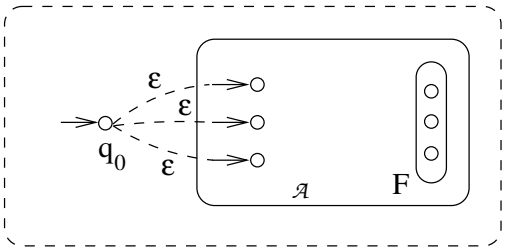


FIGURE II.16. Considérer un unique état initial n'est pas une restriction.

Soient $\mathcal{A}$ et $\mathcal{B}$ deux automates finis[5]. L'automate fini non déterministe représenté à la figure II.17 accepte $L(\mathcal{A}) \cup L(\mathcal{B})$. L'état initial de cet automate est un nouvel état et les états finals sont ceux de $\mathcal{A}$ et de $\mathcal{B}$.

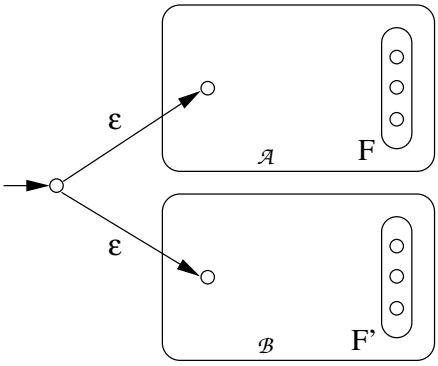


FIGURE II.17. Automate acceptant $L(\mathcal{A}) \cup L(\mathcal{B})$.

■

Proposition II.3.2. *Si L et M sont les langages acceptés par deux automates finis, alors LM est aussi accepté par un automate fini.*

Démonstration. On utilise les mêmes conventions que dans la preuve de la proposition précédente. Soient $\mathcal{A}$ et $\mathcal{B}$ deux automates finis. L'automate non déterministe représenté à la figure II.18 accepte $L(\mathcal{A})L(\mathcal{B})$. L'état initial de cet automate est l'état initial de $\mathcal{A}$ et les états finals sont ceux de $\mathcal{B}$.

■

Proposition II.3.3. *Si L est accepté par un automate fini, alors L^* l'est aussi.*

[5]Sans autre précision, on peut considérer des AFD ou des AFND.

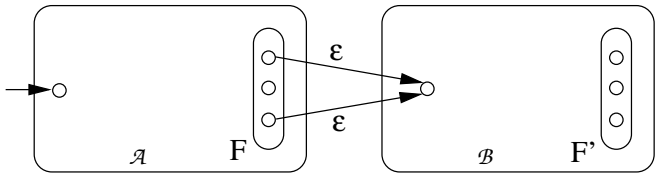


FIGURE II.18. Automate acceptant $L(\mathcal{A})L(\mathcal{B})$.

Démonstration. On emploie les mêmes conventions que dans la preuve de la proposition II.3.1. Soit $\mathcal{A}$ un automate fini. L'automate non déterministe représenté à la figure II.19 accepte $(L(\mathcal{A}))^*$. L'état initial de cet automate est un nouvel état. Les états finals sont ceux de $\mathcal{A}$ ainsi que le nouvel état initial (cela permet l'acceptation du mot vide).

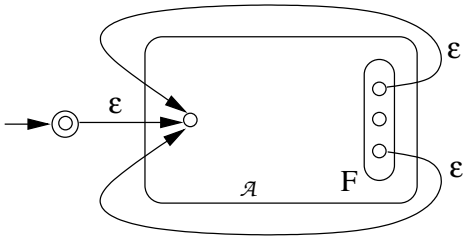


FIGURE II.19. Automate acceptant $(L(\mathcal{A}))^*$.

■

Les opérations rationnelles ne sont pas les seules à assurer la stabilité de l'ensemble des langages acceptés par automate fini.

Proposition II.3.4. *Soient $L \subseteq \Sigma^*$ un langage accepté par un automate fini et $f : \Sigma^* \to \Gamma^*$ un morphisme de monoïdes. Le langage $f(L) \subseteq \Gamma^*$ est aussi accepté par un automate fini.*

Démonstration. Soit $\mathcal{A}$ un automate fini. Sans perte de généralité, on suppose $\mathcal{A}$ élémentaire[6]. Le morphisme f est complètement défini par les valeurs qu'il attribue aux éléments de Σ. Le langage $f(L)$ est accepté par l'automate $\mathcal{A}'$ construit à partir de $\mathcal{A}$ où l'on remplace chaque arc de la forme

$$q \xrightarrow{\sigma} q'$$

par

$$q \xrightarrow{f(\sigma)} q'.$$

Rien n'assure que l'automate $\mathcal{A}'$ soit encore déterministe (cela dépend du morphisme). Cependant, il est clair que $f(L(\mathcal{A})) = L(\mathcal{A}')$. En effet, si

[6]Nous laissons au lecteur le soin de vérifier que la construction proposée dans cette preuve peut aussi être utilisée dans le cas d'un automate non élémentaire.

$w = w_1 \cdots w_k$ est accepté par $\mathcal{A}$, il existe un chemin débutant dans l'état initial

$$q_0 \xrightarrow{w_1} q_1 \xrightarrow{w_2} q_2 \longrightarrow \cdots \longrightarrow q_{k-1} \xrightarrow{w_k} q_k$$

tel que q_k soit un état final de $\mathcal{A}$. A ce chemin, il correspond dans $\mathcal{A}'$ le chemin

$$q_0 \xrightarrow{f(w_1)} q_1 \xrightarrow{f(w_2)} q_2 \longrightarrow \cdots \longrightarrow q_{k-1} \xrightarrow{f(w_k)} q_k$$

et donc, $f(w_1 \cdots w_k) = f(w_1) \cdots f(w_k)$ est accepté par $\mathcal{A}'$. Réciproquement, à tout mot v accepté par $\mathcal{A}'$, il correspond au moins un mot w accepté par $\mathcal{A}$ tel que $f(w) = v$.

■

Proposition II.3.5. *Si $L \subseteq \Sigma^*$ est accepté par un automate fini, alors $\Sigma^* \setminus L$ est aussi accepté par un automate fini.*

Démonstration. Soit $\mathcal{A}$ un automate fini déterministe acceptant L (vu la proposition II.2.11, il ne s'agit pas d'une véritable restriction). Si on inverse les statuts final/non final de chacun des états de $\mathcal{A}$, on obtient un nouvel automate acceptant exactement $\Sigma^* \setminus L(\mathcal{A})$.

■

Remarque II.3.6. Comme le montre l'exemple suivant, la méthode prescrite dans la preuve précédente requiert un automate déterministe. En effet, l'automate de la figure II.20 est non déterministe et accepte le langage $a(ba)^*$. Par contre, si on inverse le statut final/non final des états, on obtient un

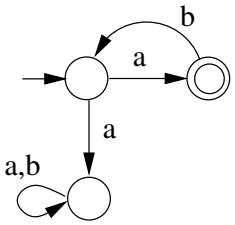


FIGURE II.20. Un AFND acceptant $a(ba)^*$.

automate acceptant $\{\varepsilon\} \cup a\{a, b\}^*$. Ce dernier langage n'est évidemment pas le complémentaire de $a(ba)^*$.

Proposition II.3.7. *Si L est un langage accepté par un automate fini, alors L^R est aussi accepté par un automate fini.*

Démonstration. Soit $\mathcal{A} = (Q, I, F, \Sigma, \Delta)$ un AFND acceptant L. L'automate

$$A^R = (Q, F, I, \Sigma, \Delta^R)$$

où Δ^R est défini par

$$(q, w, q') \in \Delta \Leftrightarrow (q', w^R, q) \in \Delta^R,$$

est un automate acceptant L^R. En effet, si un mot est accepté par $\mathcal{A}$, cela signifie qu'il existe un chemin de label w débutant dans un état de I et aboutissant dans un état de F. Ainsi, par définition dans $\mathcal{A}^R$, on a un chemin de label w^R débutant dans un état de F (ensemble des états initiaux de $\mathcal{A}^R$) et aboutissant dans un état de I (ensemble des états finals de $\mathcal{A}^R$). Ainsi, w^R est accepté par $\mathcal{A}^R$. La réciproque s'obtient de manière analogue.

■

Remarque II.3.8. Grossièrement, on observe que $\mathcal{A}^R$ est l'automate construit sur $\mathcal{A}$ où on a retourné tous les arcs et inversé les statuts initial/final des états. Si $\mathcal{A}$ est un AFD, alors en général, $\mathcal{A}^R$ est non déterministe. En effet, si dans un AFD, on dispose de trois états p, q, r tels que $\delta(p, a) = r$ et $\delta(q, a) = r$, alors dans l'automate miroir, on a

$$(r, a, p) \in \Delta \quad \text{et} \quad (r, a, q) \in \Delta.$$

Proposition II.3.9. *Si L est accepté par un automate fini, alors* $\mathrm{Pref}(L)$ *est aussi accepté par un automate fini.*

Démonstration. Soit $\mathcal{A} = (Q, q_0, F, \Sigma, \delta)$ un AFD acceptant L. Un état $q \in Q$ est dit *coaccessible* s'il existe un mot $w \in \Sigma^*$ tel que $q.w \in F$. Nous laissons au lecteur le soin de vérifier que l'automate $\mathcal{A}' = (Q, q_0, F', \Sigma, \delta)$, où F' est l'ensemble des états coaccessibles de $\mathcal{A}$, accepte $\mathrm{Pref}(L)$.

■

Proposition II.3.10. *Si L est accepté par un automate fini, alors* $\mathrm{Suff}(L)$ *est aussi accepté par un automate fini.*

Démonstration. Soit $\mathcal{A} = (Q, q_0, F, \Sigma, \delta)$ un AFD acceptant L. Un état $q \in Q$ est dit *accessible* s'il existe un mot $w \in \Sigma^*$ tel que $q = q_0.w$. Nous laissons au lecteur le soin de vérifier que l'AFND $\mathcal{A}' = (Q, I, F, \Sigma, \delta)$, où I est l'ensemble des états simultanément accessibles et coaccessibles de $\mathcal{A}$, accepte $\mathrm{Suff}(L)$.

■

Remarque II.3.11. Une autre démonstration de cette dernière proposition consiste à remarquer que

$$\mathrm{Suff}(L) = (\mathrm{Pref}(L^R))^R$$

et à utiliser la proposition II.3.7

4. Produit d'automates

Proposition II.4.1. *Si L et M sont les langages acceptés par deux automates finis, alors $L \cap M$ est aussi accepté par un automate fini.*

Démonstration. Supposons que les automates finis déterministes $\mathcal{A}$ et $\mathcal{B}$ possèdent le même alphabet[7] Σ. Ainsi,

$$\mathcal{A} = (Q^{(a)}, q_0^{(a)}, F^{(a)}, \Sigma, \delta^{(a)}) \text{ et } \mathcal{B} = (Q^{(b)}, q_0^{(b)}, F^{(b)}, \Sigma, \delta^{(b)}).$$

Considérons l'automate $\mathcal{P}$ ayant

- $Q^{(a)} \times Q^{(b)}$ comme ensemble fini d'états,
- $(q_0^{(a)}, q_0^{(b)})$ comme état initial,
- $F^{(a)} \times F^{(b)}$ comme ensemble d'états finals

et dont la fonction de transition π est définie par

$$\pi : (Q^{(a)} \times Q^{(b)}) \times \Sigma \to (Q^{(a)} \times Q^{(b)}) : ((q, q'), \sigma) \mapsto (\delta^{(a)}(q, \sigma), \delta^{(b)}(q', \sigma)).$$

Les mots acceptés par $\mathcal{P}$ sont exactement les mots $w \in \Sigma^*$ tels que

$$\pi((q_0^{(a)}, q_0^{(b)}), w) \in F^{(a)} \times F^{(b)};$$

ceci est équivalent à

$$\delta^{(a)}(q_0^{(a)}, w) \in F^{(a)} \text{ et } \delta^{(b)}(q_0^{(b)}, w) \in F^{(b)}$$

et signifie que le langage accepté par $\mathcal{P}$ est $L(\mathcal{A}) \cap L(\mathcal{B})$.

■

Proposition II.4.2. *Si L et M sont les langages acceptés par deux automates finis, alors $L \sqcup\!\sqcup M$ est aussi accepté par un automate fini.*

Démonstration. Soient

$$\mathcal{A} = (Q^{(a)}, q_0^{(a)}, F^{(a)}, \Sigma^{(a)}, \delta^{(a)}) \text{ et } \mathcal{B} = (Q^{(b)}, q_0^{(b)}, F^{(b)}, \Sigma^{(b)}, \delta^{(b)})$$

deux automates finis déterministes. Supposons dans un premier temps que

$$\Sigma^{(a)} \cap \Sigma^{(b)} = \emptyset.$$

Comme dans la preuve précédente, considérons un automate $\mathcal{P}$ ayant

- $Q^{(a)} \times Q^{(b)}$ comme ensemble fini d'états,
- $(q_0^{(a)}, q_0^{(b)})$ comme état initial,
- $F^{(a)} \times F^{(b)}$ comme ensemble d'états finals,
- $\Sigma^{(a)} \cup \Sigma^{(b)}$ comme alphabet

et dont la fonction de transition $\pi : (Q^{(a)} \times Q^{(b)}) \times (\Sigma^{(a)} \cup \Sigma^{(b)}) \to (Q^{(a)} \times Q^{(b)})$ est définie par

$$\pi : ((q, q'), \sigma) \mapsto \begin{cases} (\delta^{(a)}(q, \sigma), q') & \text{si } \sigma \in \Sigma^{(a)} \\ (q, \delta^{(b)}(q', \sigma)) & \text{si } \sigma \in \Sigma^{(b)} \end{cases}.$$

Par construction, il est clair que $L(\mathcal{P}) = L(\mathcal{A}) \sqcup\!\sqcup L(\mathcal{B})$. En effet, en lisant un mot w, on ne peut atteindre un état final de $\mathcal{P}$ que si après avoir lu toutes les lettres de $\Sigma^{(a)}$ (resp. de $\Sigma^{(b)}$) apparaissant dans w, on se trouve dans un état de $\mathcal{P}$ dont la première (resp. seconde) composante est dans $F^{(a)}$ (resp. $F^{(b)}$).

[7] Ceci est toujours possible car s'ils avaient des alphabets différents, il suffirait de considérer comme alphabet commun, l'union des deux alphabets.

Il nous reste à envisager le cas où les alphabets ne sont pas disjoints. Dans cette situation, on peut remplacer, par exemple, $\Sigma^{(b)} = \{\sigma_1, \ldots, \sigma_n\}$ par un nouvel alphabet $\underline{\Sigma}^{(b)} = \{\underline{\sigma}_1, \ldots, \underline{\sigma}_n\}$ de telle manière que $\Sigma^{(a)} \cap \underline{\Sigma}^{(b)} = \emptyset$. On applique dès lors la construction présentée ci-dessus. Pour terminer, il suffit d'appliquer le morphisme $f : (\Sigma^{(a)} \cup \underline{\Sigma}^{(b)})^* \to (\Sigma^{(a)} \cup \Sigma^{(b)})^*$ défini par

$$f(\underline{\sigma}_i) = \sigma_i, \ \forall \underline{\sigma}_i \in \underline{\Sigma}^{(b)} \text{ et } f(\sigma) = \sigma, \ \forall \sigma \in \Sigma^{(a)}.$$

On conclut en utilisant la proposition II.3.4.

■

Exemple II.4.3. Considérons les langages a^*b^* et $(cd)^*$ acceptés respectivement par les deux AFD de la figure II.21. Les tables de transition sont

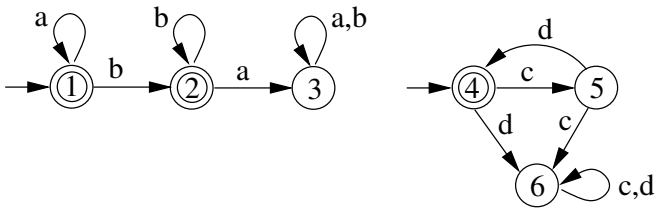


FIGURE II.21. AFD acceptant a^*b^* et $(cd)^*$.

q	$q.a$	$q.b$
1	1	2
2	3	2
3	3	3

q	$q.c$	$q.d$
4	5	6
5	6	4
6	6	6

Recherchons la table de transition de l'automate acceptant le langage $(a^*b^*) \sqcup\!\sqcup (cd)^*$,

q	$q.a$	$q.b$	$q.c$	$q.d$
$(1,4)$	$(1,4)$	$(2,4)$	$(1,5)$	$(1,6)$
$(1,5)$	$(1,5)$	$(2,5)$	$(1,6)$	$(1,4)$
$(1,6)$	$(1,6)$	$(2,6)$	$(1,6)$	$(1,6)$
$(2,4)$	$(3,4)$	$(2,4)$	$(2,5)$	$(2,6)$
$(2,5)$	$(3,5)$	$(2,5)$	$(2,6)$	$(2,4)$
$(2,6)$	$(3,6)$	$(2,6)$	$(2,6)$	$(2,6)$
$(3,4)$	$(3,4)$	$(3,4)$	$(3,5)$	$(3,6)$
$(3,5)$	$(3,5)$	$(3,5)$	$(3,6)$	$(3,4)$
$(3,6)$	$(3,6)$	$(3,6)$	$(3,6)$	$(3,6)$

Les états finals sont $(1,4)$ et $(2,4)$, l'état initial est $(1,4)$. Si on renumérote les états de 1 à 9 dans l'ordre du tableau, on a l'AFD repris à la figure II.22.

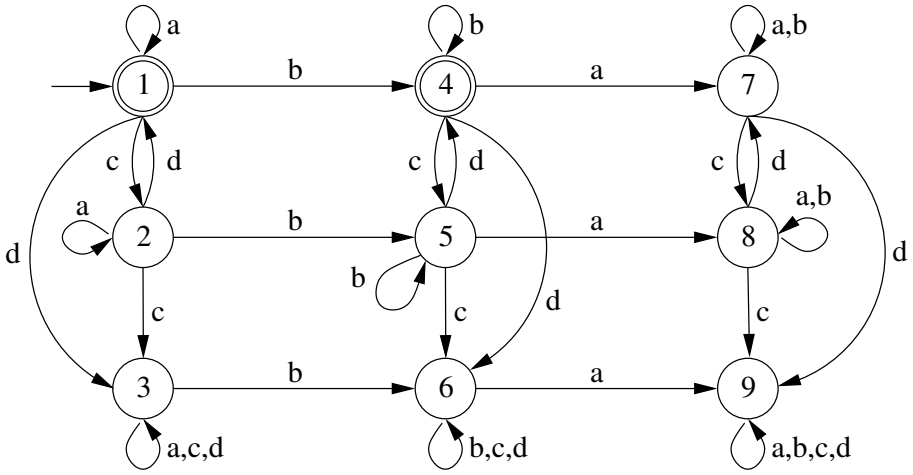


FIGURE II.22. AFD "shuffle".

5. Exercices

Exercice II.5.1. Modéliser au moyen d'un automate fini déterministe le *problème du chou, de la chèvre et du loup.* Un berger doit faire traverser une rivière au moyen d'une barque à un chou, une chèvre et un loup. La barque étant petite pour les transporter tous, à chaque trajet sur la rivière, il ne peut emporter qu'un seul des trois protagonistes. On ne peut laisser la chèvre et le chou (resp. le loup et la chèvre) seuls sur une rive. Comment doit faire le berger pour faire traverser les trois protagonistes sous les contraintes indiquées. Avec la modélisation choisie, quel est le langage des déplacements acceptables ?

Exercice II.5.2. Soit l'AFD $\mathcal{A} = (Q, 1, F, \Sigma, \delta)$ où $Q = \{1, 2, 3\}$, $\Sigma = \{a, b\}$, $F = \{3\}$ et où la fonction de transition est donnée par

δ	a	b
1	1	2
2	3	2
3	3	1

.

Tracer le diagramme d'états de $\mathcal{A}$. Donner l'exécution de $\mathcal{A}$ sur les mots

- *abba*,
- *bbbabb*,
- *bababa*,
- *bbbaa*.

Quel est le langage accepté par $\mathcal{A}$ (en donner une expression régulière) ?

Exercice II.5.3. Soient les langages a^*b^* et $\{c\}$. Construire un AFD acceptant $a^*b^* \sqcup\!\sqcup \{c\}$. Pour ce faire, on construira au préalable un AFD sur $\{a, b\}$ acceptant le langage a^*b^* et un AFD sur $\{c\}$ acceptant le langage $\{c\}$. Si on note ρ_L la fonction qui à $n \in \mathbb{N}$ associe le nombre de mots de longueur

n dans L,

$$\rho_L : \mathbb{N} \to \mathbb{N} : n \mapsto \#(L \cap \Sigma^n).$$

Que vaut $\rho_{a^*b^*}(n)$? Même question pour $\rho_{a^*b^* \sqcup\!\sqcup \{c\}}(n)$.

Exercice II.5.4. Soient les deux AFD représentés à la figure II.23. Construire

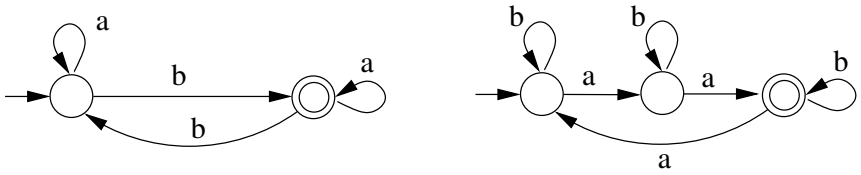


FIGURE II.23. Deux automates finis déterministes.

un AFD acceptant le shuffle des langages acceptés par ces deux automates.

Exercice II.5.5. Soit l'AFND représenté à la figure II.24.

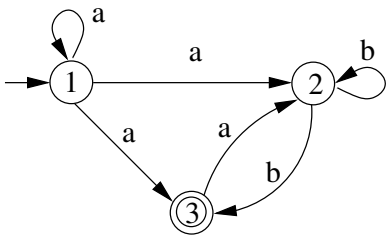


FIGURE II.24. Un AFND.

- Enumérer les éléments de la relation de transition Δ de l'automate.
- Quelles sont toutes les exécutions possibles du mot *aaabb* dans cet automate (en démarrant de l'unique état initial).
- Le mot *aaabb* est-il accepté ?
- Rendre cet automate déterministe au moyen de la construction par sous-ensembles d'états.
- Donner une expression régulière du langage accepté par l'automate.

Exercice II.5.6. Rendre déterministe l'automate repris à la figure II.25. (Prendre garde aux ε-transitions.)

Exercice II.5.7. Remplacer l'automate représenté à la figure II.26 par un automate équivalent possédant un unique état initial et un unique état final.

Exercice II.5.8. Construire un AFND acceptant

$$(ab)^* + a^*.$$

Si l'automate obtenu n'est pas déterministe, le rendre déterministe.

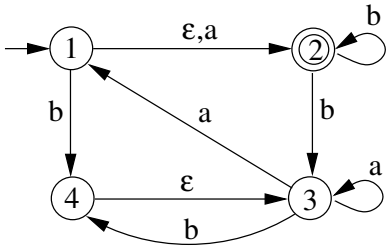


FIGURE II.25. Un AFND à rendre déterministe.

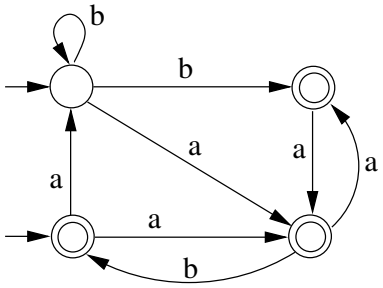


FIGURE II.26. Un AFND.

Exercice II.5.9 (Cet exercice pourra être passé en première lecture, et repris après avoir vu la notion d'automate minimal). Montrer que pour tout $n \geq 1$, le langage $(a+b)^*b(a+b)^{n-1}$ peut être reconnu par un AFND à $n+1$ états, mais que tout AFD acceptant ce langage possède au moins 2^n états.

Exercice II.5.10. Construire un AFND acceptant

$$(abc)^*a^*.$$

Si l'automate obtenu n'est pas déterministe, le rendre déterministe.

Exercice II.5.11. En utilisant l'exercice précédent, construire un AFD acceptant le langage

$$((abc)^*a^*)^R.$$

Exercice II.5.12. Construire un AFND acceptant

$$(ba + bb)^* + (ab + aa)^*.$$

Si l'automate obtenu n'est pas déterministe, le rendre déterministe.

Exercice II.5.13. Construire un AFND acceptant

$$(ab^+a)^+.$$

Si l'automate obtenu n'est pas déterministe, le rendre déterministe.

Exercice II.5.14. Construire un AFD acceptant exactement les mots sur $\{a, b\}$ qui contiennent le facteur $abba$.

Exercice II.5.15. Construire un AFD acceptant exactement les mots sur $\{a, b\}$ pour lesquels tout facteur de longueur 4 contient au moins un a.

Exercice II.5.16. Soit $L \subset \{a, b, c\}^*$ un langage dont aucun mot ne commence par a. Montrer que L est un langage accepté par un AFD si et seulement si a^*L est aussi accepté par un AFD.

Exercice II.5.17. Soit $\Sigma = \{a, b\}$. Construire un AFD acceptant le langage suivant

- $\{w \in \Sigma^* : |w| \equiv 0 \pmod 3\}$,
- $\{w \in \Sigma^* : |w|_a \equiv 0 \pmod 3\}$,
- $\{w \in \Sigma^* : |w| \equiv 1 \pmod 3\}$,
- $\{w \in \Sigma^* : |w| \not\equiv 0 \pmod 3\}$,
- $\{w \in \Sigma^* : |w|_a \leq 4\}$,

Exercice II.5.18. Construire un AFD acceptant le langage

$$\{a^i b^j \mid i \equiv j \pmod 2\}.$$

Définition II.5.19. Soit $p \geq 2$, tout entier $n \geq 1$ se décompose de manière unique comme

$$n = \sum_{i=0}^{\ell} c_i \, p^i, \qquad \text{avec } c_i \in \{0, \ldots, p-1\} \text{ et } p_\ell \neq 0.$$

Le mot $c_\ell \cdots c_0 \in \{0, \ldots, p-1\}^*$ est *la représentation* en base p de l'entier n. Par convention, zéro est représenté par le mot vide. Cette manière de procéder fournit une bijection entre $\mathbb{N}$ et le langage

$$L_p = \{\varepsilon\} \cup \{1, \ldots, p-1\}\{0, \ldots, p-1\}^*.$$

Exercice II.5.20. Soient $k, m \geq 2$. Démontrer que $\{k^n \pmod m \mid n \in \mathbb{N}\}$ est ultimement périodique.

Exercice II.5.21. Construire un AFD acceptant exactement les représentations binaires des nombres pairs. (On suppose que 0 est représenté par le mot vide et pour des raisons de simplification, on autorise les zéros de tête dans les représentations, i.e., 000101 est par exemple une représentation de 5.) Si besoin est, on permet de considérer les représentations miroir.

Exercice II.5.22. Même question avec les représentations binaires des multiples de 4, 5, 6 ou 7.

Exercice II.5.23. Donner la table de transition d'un automate fini déterministe reconnaissant les écritures décimales des multiples de 6 (ou leur miroir, si vous jugez la construction plus simple).

Remarque II.5.24. Ces trois derniers exercices montrent que tout critère de divisibilité peut toujours être reconnu par un automate fini et ce, quelle que soit la base choisie pour les représentations des entiers.

Exercice II.5.25. Soit $\Sigma = \{0, 1\}$. Si $u \in \Sigma^*$, alors on note $\pi_2(u)$ l'entier représenté par u en base 2. Par exemple,

$$\pi_2(1101) = 13,\ \pi_2(001010) = 10.$$

On considère l'alphabet $\Gamma = \Sigma \times \Sigma$. Construire un automate sur Γ qui reconnaît le langage des couples (u, v) de mots de même longueur tels que

$$\pi_2(v) = 2\pi_2(u).$$

Pour obtenir des mots de même longueur, on s'autorise toujours à placer des zéros de tête dans les représentations. Par exemple,

$$\begin{pmatrix} 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 \end{pmatrix}$$

appartient au langage accepté. Comme dans les exercices précédents, par souci de simplification, on pourra dans un premier temps considérer les représentations miroir.

Exercice II.5.26. Dans le même contexte que l'exercice précédent, on note $\Gamma = \Sigma^3$. Construire un automate sur Γ qui reconnaît les triplets (u, v, w) de mots de même longueur tels que

$$\pi_2(u) + \pi_2(v) = \pi_2(w).$$

Par exemple,

$$\begin{pmatrix} 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 \end{pmatrix}$$

appartient au langage accepté. Comme dans les exercices précédents, par souci de simplification, on pourra dans un premier temps considérer les représentations miroir.

Exercice II.5.27. Même question qu'à l'exercice II.5.25, mais cette fois, on impose

$$\pi_2(v) = 3\pi_2(u).$$

Exercice II.5.28. Montrer que le langage des représentations binaires des nombres entiers divisibles par 4 est régulier, en donnant une expression régulière.

Montrer que le langage des représentations binaires des nombres entiers divisibles par 3 est régulier, en fournissant un automate fini déterministe acceptant ce langage (ou son miroir, au choix).

Déduire des deux premiers points que le langage des représentations binaires des nombres entiers divisibles par 12 est régulier ? Justifier votre réponse.

CHAPITRE III

Langages réguliers et automates

Le but premier de ce chapitre est de montrer que l'ensemble des langages réguliers coïncide exactement avec l'ensemble des langages acceptés par automate fini. Nous allons donc faire le lien entre les notions introduites aux deux premiers chapitres.

1. Des expressions aux automates

A toute expression régulière φ, on peut associer un automate fini $\mathcal{A}$ de telle sorte que $\mathcal{L}(\varphi) = L(\mathcal{A})$. On procède par récurrence sur la longueur de φ.

- Si $\varphi = 0$, les automates suivants acceptent tous deux le langage $\emptyset$.

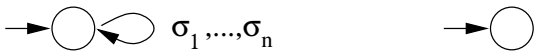


FIGURE III.1. AFD et AFND acceptant $\emptyset$.

- Si $\varphi = e$, les automates suivants acceptent le langage $\{\varepsilon\}$.

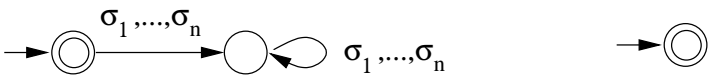


FIGURE III.2. AFD et AFND acceptant $\{\varepsilon\}$.

- Si $\varphi = \sigma$, $\sigma \in \Sigma$, les automates suivants acceptent le langage $\{\sigma\}$.

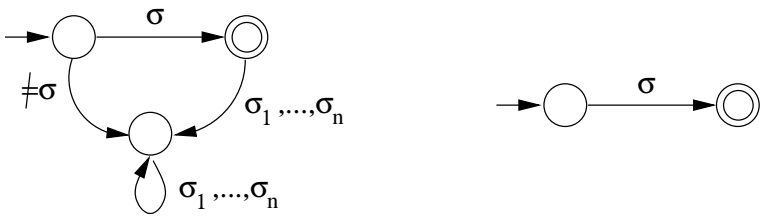


FIGURE III.3. AFD et AFND acceptant $\{\sigma\}$.

- Si $\varphi = (\psi + \mu)$, avec ψ et μ des expressions régulières de longueur inférieure à celle de φ, alors, par hypothèse de récurrence, on dispose de deux automates finis $\mathcal{A}_\psi$ et $\mathcal{A}_\mu$ acceptant respectivement $\mathcal{L}(\psi)$ et $\mathcal{L}(\mu)$. On conclut en utilisant la proposition II.3.1.

- Si $\varphi = (\psi.\mu)$, on utilise les mêmes raisonnements et la proposition II.3.2.
- Enfin, si $\varphi = \psi^*$, on tire la même conclusion en utilisant cette fois la proposition II.3.3.

Ainsi, de proche en proche, on peut, étant donné une expression régulière, construire un automate acceptant le langage généré par l'expression.

Exemple III.1.1. Soit l'expression régulière $\varphi = (a^*ba^*b)^*a^*$. Des automates acceptant $\mathcal{L}(a) = \{a\}$ et $\mathcal{L}(b) = \{b\}$ sont donnés par :

FIGURE III.4. AFND acceptant $\{a\}$ et $\{b\}$.

En utilisant la proposition II.3.3, on construit un automate acceptant a^* :

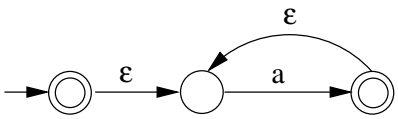


FIGURE III.5. AFND acceptant $\{a\}^*$.

Pour des raisons de simplifications évidentes, nous allons considérer un automate équivalent acceptant aussi a^* :

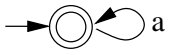


FIGURE III.6. AFND équivalent acceptant $\{a\}^*$.

En utilisant la proposition II.3.2, on construit un automate acceptant a^*b :

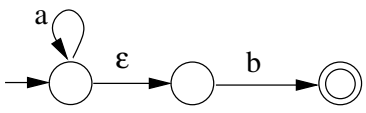


FIGURE III.7. AFND acceptant $\{a\}^*\{b\}$.

En utilisant cette proposition une seconde fois, on obtient un automate acceptant a^*ba^*b :

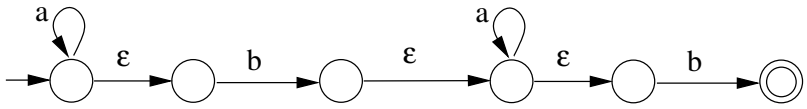


FIGURE III.8. AFND acceptant $\{a\}^*\{b\}\{a\}^*\{b\}$.

Et en simplifiant quelque peu, on a même

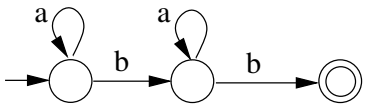


FIGURE III.9. AFND équivalent acceptant $\{a\}^*\{b\}\{a\}^*\{b\}$.

Appliquons à présent la proposition II.3.3 à ce dernier automate pour obtenir

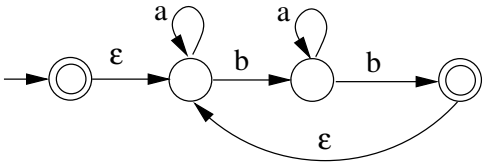


FIGURE III.10. AFND acceptant $(\{a\}^*\{b\}\{a\}^*\{b\})^*$.

La dernière étape consiste à combiner l'automate ci-dessus avec celui acceptant a^* au moyen de la proposition II.3.2.

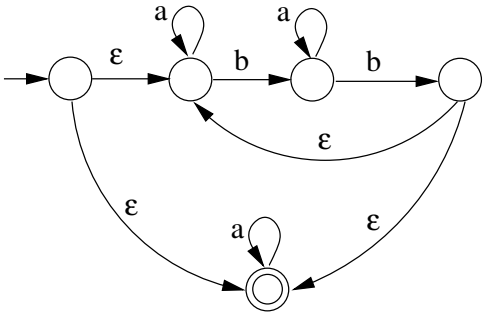


FIGURE III.11. AFND acceptant $(\{a\}^*\{b\}\{a\}^*\{b\})^*\{a\}^*$.

2. Des automates aux expressions régulières

Nous définissons tout d'abord des automates généralisés dont les arcs ont comme label non pas des lettres de l'alphabet Σ mais des expressions régulières. Pour rappel, on note $\mathcal{R}_\Sigma$, l'ensemble des expressions régulières sur Σ.

Définition III.2.1. Un *automate fini étendu*[1] (AFE) est la donnée d'un quintuple

$$\mathcal{A} = (Q, q_0, F, \Sigma, \delta)$$

où

- Q est un ensemble fini d'états,
- $q_0 \in Q$ est l'état initial,
- $F \subseteq Q$ est l'ensemble des états finals,
- $\delta : Q \times Q \to \mathcal{R}_\Sigma$ est la *fonction d'étiquetage* des transitions.

Si aucun transition entre q et q' n'est explicitement définie, on pose $\delta(q, q') = 0$ si $q \neq q'$ et $\delta(q, q') = e$ si $q = q'$.

Exemple III.2.2. L'automate représenté à la figure III.12 est un AFE. On a $\delta(1,2) = ab^*$, $\delta(2,2) = (a + ab)$, $\delta(2,3) = bab$, $\delta(1,1) = \delta(3,3) = e$ et

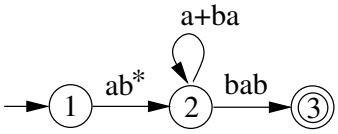


FIGURE III.12. Un automate fini étendu (AFE).

$\delta(i,j) = 0$ pour $(i,j) \in \{(1,3),(2,1),(3,1),(3,2)\}$.

Définition III.2.3. Un mot w est *accepté* par un AFE $\mathcal{A} = (Q, q_0, F, \Sigma, \delta)$ si il existe des mots $w_1, \ldots, w_n \in \Sigma^*$ et des états $q_1, \ldots, q_n \in Q$ tels que $w = w_1 \cdots w_n$,

$$w_1 \in \mathcal{L}(\delta(q_0, q_1)), \ldots, w_n \in \mathcal{L}(\delta(q_{n-1}, q_n))$$

et $q_n \in F$.

Par exemple, pour l'AFE donné dans l'exemple précédent, le mot $w = abbbbbabab$ est accepté car si on pose $w_1 = abbbb$, $w_2 = ba$ et $w_3 = bab$, on s'aperçoit que $w_1 \in \mathcal{L}(ab^*)$, $w_2 \in \mathcal{L}(a + ba)$ et $w_3 \in \mathcal{L}(bab)$.

Définition III.2.4. Le *langage accepté* par un AFE est l'ensemble des mots qu'il accepte. Deux AFE sont dits *équivalents* s'ils acceptent le même langage.

Remarque III.2.5. Un AFD est un cas particulier d'AFE où toutes les transitions sont des expressions régulières de la forme σ, $\sigma \in \Sigma$. Ainsi, les techniques décrites ci-après peuvent s'appliquer au départ d'un AFD.

[1] En anglais, on trouve la dénomination "*extended finite automaton*".

Dans les lignes qui suivent, nous allons expliquer comment, au départ d'un AFE arbitraire, obtenir un AFE équivalent possédant uniquement deux états (un initial et un final). De cette manière, il sera aisé d'en déduire une expression régulière du langage accepté.

Le pivotage (Elimination d'un état qui n'est ni initial, ni final).
Soit $\mathcal{A} = (Q, q_0, F, \Sigma, \delta)$ un AFE. Pour tous $p, q \in Q$, on note r_{pq} l'expression régulière $\delta(p, q)$. Soit q un état de $\mathcal{A}$ tel que $q \neq q_0$ et $q \notin F$. Définissons l'AFE

$$\mathcal{A}' = (Q', q_0, F, \Sigma, \delta')$$

où $Q' = Q \setminus \{q\}$ et pour tous $p, s \in Q'$,

$$\delta'(p, s) = r_{ps} + r_{pq}\, r_{qq}^*\, r_{qs}.$$

Par construction, il est clair que $\mathcal{A}'$ est équivalent à $\mathcal{A}$.

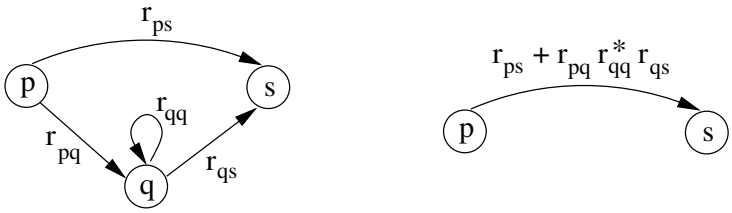


FIGURE III.13. Le pivotage.

Exemple III.2.6. Considérons l'AFE donné à la figure III.14. Avec les

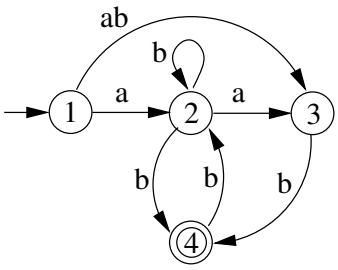


FIGURE III.14. Un AFE avant élimination de l'état 2.

notations précédentes, si on désire éliminer l'état 2, on obtient

$$\begin{aligned}
\delta'(1,1) &= r_{11} + r_{12}\, r_{22}^*\, r_{21} = e + a\, b^*\, 0 = e \\
\delta'(1,3) &= r_{13} + r_{12}\, r_{22}^*\, r_{23} = ab + a\, b^*\, a \\
\delta'(1,4) &= r_{14} + r_{12}\, r_{22}^*\, r_{24} = 0 + a\, b^*\, b = ab^*b \\
\delta'(3,1) &= r_{31} + r_{32}\, r_{22}^*\, r_{21} = 0 + 0\, b^*\, 0 = 0 \\
\delta'(3,3) &= r_{33} + r_{32}\, r_{22}^*\, r_{23} = e + 0\, b^*\, a = e \\
\delta'(3,4) &= r_{34} + r_{32}\, r_{22}^*\, r_{24} = b + 0\, b^*\, b = b \\
\delta'(4,1) &= r_{41} + r_{42}\, r_{22}^*\, r_{21} = 0 + b\, b^*\, 0 = 0 \\
\delta'(4,3) &= r_{42} + r_{42}\, r_{22}^*\, r_{23} = 0 + b\, b^*\, a = bb^*a \\
\delta'(4,4) &= r_{44} + r_{42}\, r_{22}^*\, r_{24} = e + b\, b^*\, b = bb^*b + e
\end{aligned}$$

En ne représentant que les transitions différentes de 0 et différentes de $\delta(q,q) = e$, on obtient l'AFE équivalent représenté à la figure III.15.

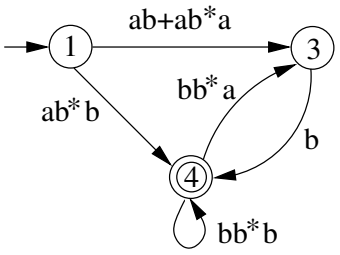


FIGURE III.15. AFE équivalent après élimination de l'état 2.

L'algorithme complet[2]
Soit $\mathcal{A} = (Q, q_0, F, \Sigma, \delta)$ un AFE.
(1.a) Obtention d'un état initial non final et au quel on ne peut aboutir. Si l'état initial q_0 est final ou si il existe $q \in Q$ tel que[3] $\delta(q, q_0) \neq 0$, alors on ajoute un nouvel état q_0' à l'ensemble Q d'états et on pose $\delta(q_0', q_0) = e$. On redéfinit q_0' comme le nouvel état initial.
(1.b) Obtention d'un unique état final. Si $\#F > 1$, c'est-à-dire, s'il y a plus d'un état final, on ajoute un nouvel état f' et on pose $\delta(q, f') = e$ pour tout $q \in F$. Ensuite, on redéfinit $\{f'\}$ comme nouvel ensemble d'états finals.

Ainsi, à la fin de l'étape **1**, on peut supposer disposer d'un AFE équivalent $\mathcal{A}' = (Q', q_0', \{f'\}, \Sigma, \delta')$ possédant un unique état initial q_0' (non final et auquel n'aboutit aucune transition) et un unique état final f'.

(2) Fin ? Si $Q' = \{q_0', f'\}$, alors une expression régulière du langage accepté par $\mathcal{A}'$ est

$$r_{q_0'f'}(r_{f'f'})^*$$

où $r_{q_0'f'} = \delta'(q_0', f')$ et $r_{f'f'} = \delta'(f', f')$. L'algorithme s'achève. Sinon, on passe à l'étape **3**.
(3) Elimination d'un état. Il existe $q \in Q' \setminus \{q_0', f'\}$. On élimine q de $\mathcal{A}'$ par la méthode du pivot présentée ci-dessus. Après pivotage, l'ensemble d'états est $Q \setminus \{q\}$. On recommence le point **2**. A chaque étape, le nombre d'états décroît strictement. Par conséquent, l'algorithme s'achève toujours.

[2]Il s'agit de l'algorithme de McNaughton-Yamada.

[3]On ne tient pas compte du cas trivial $\delta(q_0, q_0) = e$. Par contre, si il existe $r \neq e$ tel que $\delta(q_0, q_0) = r$, alors on effectue la modification de l'automate.

Exemple III.2.7. Poursuivons l'exemple III.2.6. Si on élimine le sommet 3 de l'AFE représenté à la figure III.15, il vient

$$\begin{array}{rcl}\delta'(1,1) & = & r_{11} + r_{13}\, r_{33}^*\, r_{31} = e + (ab + ab^*a)\, e\, 0 = e \\ \delta'(1,4) & = & r_{14} + r_{13}\, r_{33}^*\, r_{34} = ab^*b + (ab + ab^*a)\, e\, b \\ \delta'(4,1) & = & r_{41} + r_{43}\, r_{33}^*\, r_{31} = 0 + (bb^*b)\, e\, 0 = 0 \\ \delta'(4,4) & = & r_{44} + r_{43}\, r_{33}^*\, r_{34} = bb^*b + (bb^*a)\, e\, b\end{array} .$$

On obtient l'automate représenté à la figure III.16. Finalement une expres-

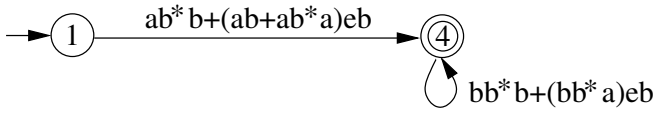


FIGURE III.16. AFE équivalent après élimination de l'état 3.

sion régulière du langage accepté par l'automate de départ est

$$(ab^*b + (ab + ab^*a)\, e\, b)(bb^*b + (bb^*a)\, e\, b)^*.$$

Puisqu'à toute expression régulière φ, correspond un automate acceptant le langage $\mathcal{L}(\varphi)$ et qu'à tout langage L accepté par un automate correspond une expression régulière ψ telle que $L = \mathcal{L}(\psi)$, nous avons le résultat suivant.

Théorème III.2.8 (Kleene). *Un langage est régulier si et seulement si il est accepté par un automate fini.*

■

Remarque III.2.9. D'une certaine manière, on peut dire que les expressions régulières sont les *générateurs* des langages réguliers, alors que les automates finis en sont les *accepteurs*.

3. Stabilité de la régularité

Théorème III.3.1. *L'ensemble des langages réguliers est stable par union, concaténation, étoile de Kleene, image par morphisme, miroir, passage au complémentaire, intersection et shuffle.*

Démonstration. Cela résulte immédiatement des résultats démontrés au chapitre II concernant les langages acceptés par automate fini et du théorème de Kleene.

■

Le résultat suivant est souvent utilisé pour vérifier que certains langages ne sont pas réguliers. Il s'agit simplement d'une redite du corollaire I.3.10.

Corollaire III.3.2. *Si L est un langage régulier sur $\Sigma = \{\sigma_1, \ldots, \sigma_n\}$, alors $|L| = \{|w| : w \in L\} \subseteq \mathbb{N}$ est une union finie de progressions arithmétiques.*

Démonstration. Soit $\Gamma = \{\gamma\}$ un alphabet unaire. L'application

$$f : \Sigma^* \to \Gamma^* : \sigma_i \mapsto \gamma, \ \forall i \in \{1, \ldots, n\}$$

est un morphisme de monoïdes préservant les longueurs, i.e., pour tout mot $w \in \Sigma^*$, $|f(w)| = |w|$. Par conséquent,

$$|f(L)| = |L|.$$

Puisque L est régulier, par le théorème III.3.1, $f(L)$ est un langage régulier sur un alphabet unaire. Au vu de la proposition I.3.9, $|f(L)|$ est une union finie de progressions arithmétiques.

■

4. Critère de non-régularité

Lemme III.4.1 (Lemme de la pompe). [4] *Soit $L \subseteq \Sigma^*$ un langage régulier. Il existe un entier ℓ tel que pour tout mot w de L satisfaisant $|w| \geq \ell$, il existe $x, y, z \in \Sigma^*$ tels que $w = xyz$ et*

- $|xy| \leq \ell$,
- $y \neq \varepsilon$,
- $xy^*z \subset L$.

Démonstration. Puisque L est régulier, il est accepté par un AFD $\mathcal{A} = (Q, q_0, F, \Sigma, \delta)$ possédant ℓ états. Un mot $w = w_1 \cdots w_n \in L$ de longueur n correspond à une exécution passant par $n+1$ états $q_0, q_1, \ldots, q_n$,

$$q_0 \xrightarrow{w_1} q_1 \xrightarrow{w_2} q_2 \cdots q_{n-1} \xrightarrow{w_n} q_n \in F.$$

Puisque $\mathcal{A}$ possède ℓ états, si $n \geq \ell$, alors au moins deux états dans la suite d'états sont égaux. Soient q_i et q_j deux tels états (on suppose que l'on considère la première répétition de deux états, i.e., $q_i = q_j$, $0 \leq i < j \leq n$ et $q_0, \ldots, q_{j-1}$ sont deux à deux distincts). On a donc pour tout $t \geq 0$, l'exécution suivante

$$q_0 \underbrace{\xrightarrow{w_1} \cdots \xrightarrow{w_i}}_{x} \left[q_i \underbrace{\xrightarrow{w_{i+1}} \cdots \xrightarrow{w_j}}_{y} \right]^t q_j \underbrace{\xrightarrow{w_{j+1}} \cdots \xrightarrow{w_n}}_{z} q_n \in F$$

où $[\cdot]^t$ signifie que la boucle est empruntée t fois. En posant x, y, z comme indiqués sur la figure III.17, la conclusion en découle.

■

Le lemme de la pompe est très souvent utilisé pour démontrer que certains langages ne sont pas réguliers.

Exemple III.4.2. Considérons une fois encore le langage

$$L = \{a^{n^2} \mid n \in \mathbb{N}\}.$$

[4] En anglais, on trouve souvent l'expression "*pumping lemma*". En français, on rencontre parfois, pour des raisons évidentes, la dénomination "*lemme de l'étoile*".

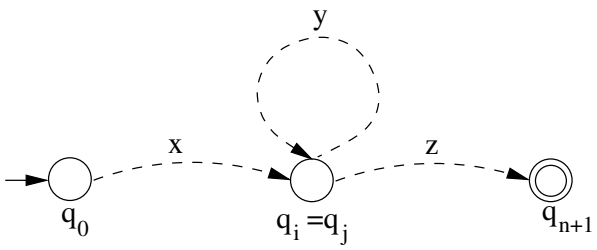


FIGURE III.17. Le lemme de la pompe.

Nous avons déjà montré dans l'exemple I.3.13 que ce langage n'était pas régulier (en utilisant la proposition I.3.9). Utilisons ici le lemme de la pompe. Si L était régulier, il serait accepté par un AFD $\mathcal{A}$ ayant k états. Dès lors, le mot a^{k^2} est accepté par $\mathcal{A}$ et cet automate comprend donc une boucle de longueur $i > 0$ (car $k^2 \geq k$). Par conséquent, tout mot de longueur

$$k^2 + n\,i,\ n \in \mathbb{N}$$

est accepté par $\mathcal{A}$. Or, l'ensemble des carrés parfaits ne contient aucune progression arithmétique infinie. On en tire que le langage L ne peut être régulier.

Remarque III.4.3. Attirons l'attention du lecteur sur le fait que des langages non réguliers peuvent néanmoins satisfaire la condition du lemme de la pompe. En effet, soit $L \subset b^*$ un langage non régulier arbitraire. Le langage

$$\{a\}^+ L \cup \{b\}^*$$

satisfait le lemme de la pompe. Il suffit de prendre avec les notations du lemme, $\ell = 1$.

La version suivante du lemme de la pompe fournit une condition nécessaire et suffisante pour qu'un langage soit régulier.

Lemme III.4.4 (Lemme de la pompe, version forte). [5] *Un langage $L \subseteq \Sigma^*$ est régulier si et seulement si il existe une constante $k > 0$ telle que pour tout mot $w \in \Sigma^*$, si $|w| \geq k$, alors il existe $x, y, z \in \Sigma^*$ tels que $w = xyz$, $y \neq \varepsilon$ et pour tout $i \geq 0$ et pour tout $v \in \Sigma^*$,*

$$wv \in L \Leftrightarrow xy^i zv \in L.$$

Démonstration. La condition est nécessaire. Supposons que le langage L est accepté par un AFD $\mathcal{A} = (Q, q_0, F, \Sigma, \delta)$ possédant k états. Tout mot $w = w_1 \cdots w_\ell$ de longueur $\ell \geq k$ fournit une exécution de la forme

$$q_0 \xrightarrow{w_1} q_1 \xrightarrow{w_2} \cdots \xrightarrow{w_\ell} q_\ell$$

où q_0 est l'état initial. Par un raisonnement analogue à celui développé dans la preuve précédente, il existe $0 \leq i < j \leq \ell$ tels que $q_i = q_j$ et l'automate $\mathcal{A}$

[5] Ce résultat est dû à J. Jaffe (*SIGACT News*, 1978). Nous avons repris ici une preuve extraite de S. Yu, *Regular Languages*, Handbook of formal languages, Springer, 1997.

a donc une boucle. On pose $x = w_1 \cdots w_i$, $y = w_{i+1} \cdots w_j$ et $z = w_{j+1} \cdots w_\ell$ (si $i = 0$, $x = \varepsilon$ et si $j = \ell$, $z = \varepsilon$). Dès lors, pour tout $i \geq 0$,

$$\delta(q_0, xy^i z) = q_\ell$$

et ainsi, pour tout $i \geq 0$,

$$\delta(q_0, xy^i zv) = \delta(q_0, xyzv) = \delta(q_0, wv)$$

ce qui signifie que

$$wv \in L \Leftrightarrow xy^i zv \in L.$$

Passons à la réciproque et supposons qu'il existe une constante $k > 0$ telle que le langage L satisfasse les propriétés énoncées. Nous devons montrer que L est régulier. Pour ce faire, nous allons construire un AFD $\mathcal{A}$ et vérifier que $L = L(\mathcal{A})$. Soit $\mathcal{A} = (Q, q_0, F, \Sigma, \delta)$ où chaque état de Q correspond à un mot $w \in \Sigma^*$ de longueur strictement inférieure à k, i.e.,

$$Q = \{q_w \mid w \in \Sigma^* \text{ et } |w| < k\}.$$

L'état initial de $\mathcal{A}$ est q_ε et $F = \{q_w \mid w \in L\}$. La fonction de transition est définie par

- si $|w| < k - 1$, alors pour tout $\sigma \in \Sigma$,

  $$\delta(q_w, \sigma) = q_{w\sigma}$$

- si $|w| = k-1$, alors pour tout $\sigma \in \Sigma$, $w\sigma$ est un mot de longueur k et par hypothèse, il peut se décomposer en xyz avec y non vide et tel que pour tout $v \in \Sigma^*$, $xyzv \in L$ si et seulement si $xzv \in L$. Il peut y avoir plus d'une telle décomposition (mais il y en a toujours au moins une). S'il y a plus d'une décomposition, on choisit celle pour laquelle xy est le plus court (et si une ambiguïté subsiste encore, on choisit parmi les décompositions ayant xy de longueur minimum, celle où y est le plus court). On pose

  $$\delta(q_w, \sigma) = q_{xz}.$$

  (On remarque que $|xz| < k$ puisque y est non vide.)

Il nous reste à montrer que $L(\mathcal{A}) = L$. On procède par récurrence sur la longueur d'un mot $w \in \Sigma^*$. Par définition de l'automate $\mathcal{A}$, il est clair qu'un mot w de longueur strictement inférieure à k appartient à L si et seulement si il appartient à $L(\mathcal{A})$. Soit $n \geq k$. Supposons la propriété satisfaite pour les mots de longueur inférieure à n et vérifions-la pour les mots w tels que $|w| = n$. Dès lors, il existe w_0 et v tels que

$$w = w_0 v, \quad \text{avec} \quad |w_0| = k.$$

Par définition de $\mathcal{A}$, il existe $x, z \in \Sigma^*$ tels que

$$\delta(q_0, w_0) = \delta(q_0, xz) = q_{xz}, \quad \text{avec} \quad w_0 = xyz,\ y \neq \varepsilon$$

et en particulier, $w_0 v = w$ appartient à L si et seulement si xzv appartient à L. De plus, on a

$$\delta(q_0, w_0 v) = \delta(q_0, xzv) = \delta(q_{xz}, v),$$

ce qui signifie que $w_0 v = w$ appartient à $L(\mathcal{A})$ si et seulement si xzv appartient à $L(\mathcal{A})$ (en effet, on atteint le même état de $\mathcal{A}$). Or $|xzv| < n$ (car y non vide) et donc, par hypothèse de récurrence, xzv appartient à $L(\mathcal{A})$ si et seulement si il appartient à L. En conclusion, $w \in L(\mathcal{A}) \Leftrightarrow w \in L$.

■

Remarque III.4.5. Nous voulons faire observer au lecteur que cette dernière proposition nécessite une décomposition de w en xyz qui doit pouvoir être appliquée pour tout mot wv, $v \in \Sigma^*$.

5. Exercices

5.1. Langages réguliers.

Exercice III.5.1. Soit le langage

$$L = \{ab^2a^3b^4 \cdots a^{2n-1}b^{2n} \mid n \in \mathbb{N}\}.$$

Ce langage est-il régulier ? Justifier.

Exercice III.5.2. Le langage $\{a^n b^n \mid n \in \mathbb{N}\}$ est-il régulier ?

Exercice III.5.3. Le langage $\{a^n b^{2n} \mid n \in \mathbb{N}\}$ est-il régulier ?

Exercice III.5.4. Le langage $\{w \in \{a,b\}^* : |w|_a < |w|_b\}$ est-il régulier ?

Exercice III.5.5. Le langage $\{a^{2^n} \mid n \in \mathbb{N}\}$ est-il régulier ?

Exercice III.5.6. Soit le morphisme $f : \{a,b\}^* \to \{a,b\}$ tel que

$$f(a) = b \quad \text{et} \quad f(b) = a.$$

Le langage $L = \{wf(w) \mid w \in \{a,b\}^*\}$ est-il régulier ?

Exercice III.5.7. Soit $\mathcal{A}$ un AFD possédant k états, $k \geq 1$. Démontrer que si le langage accepté par $\mathcal{A}$ ne contient aucun mot de longueur strictement inférieure à k, alors le langage accepté par $\mathcal{A}$ est vide.

Exercice III.5.8. Soit $\mathcal{A}$ un AFD possédant k états, $k \geq 1$. Démontrer que si le langage accepté par $\mathcal{A}$ est fini, alors tout mot accepté w est tel que $|w| < k$.

Exercice III.5.9. Soit Σ un alphabet de taille au moins 2. Le langage des palindromes sur Σ est-il régulier ? Que se passe-t-il dans le cas particulier d'un alphabet unaire ?

Exercice III.5.10. Le langage $\{a^n b^m a^{n+m} \mid m,n \in \mathbb{N}\}$ est-il régulier ?

Exercice III.5.11. Le langage formé des mots sur $\{a, b\}$ qui contiennent deux fois plus de a que de b, i.e.,

$$L = \{w \in \{a, b\}^* : |w|_a = 2|w|_b\},$$

est-il régulier ? Que vaut $\psi(L)$?

Exercice III.5.12. Soient les alphabets $\Sigma = \{a, b, c\}$ et $\Gamma = \{e, f\}$ et un langage L sur Σ. On donne le morphisme $h : \Sigma \to \Gamma$ tel que

$$h(a) = h(b) = e \quad \text{et} \quad h(c) = f.$$

Si $h(L) \subset \Gamma^*$ est un langage régulier, peut-on en déduire que L est lui-même régulier, justifier ?

5.2. Langage accepté par un automate.

Exercice III.5.13. Déterminer une expression régulière du langage accepté par l'automate repris en figure III.18.

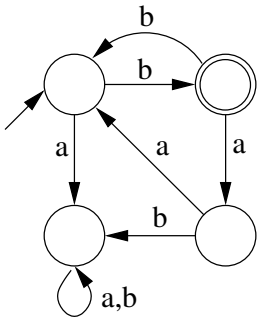


FIGURE III.18. Expression régulière du langage accepté.

Exercice III.5.14. Même question que l'exercice précédent pour l'AFD représenté à la figure III.19. Si les mots acceptés sont considérés comme des

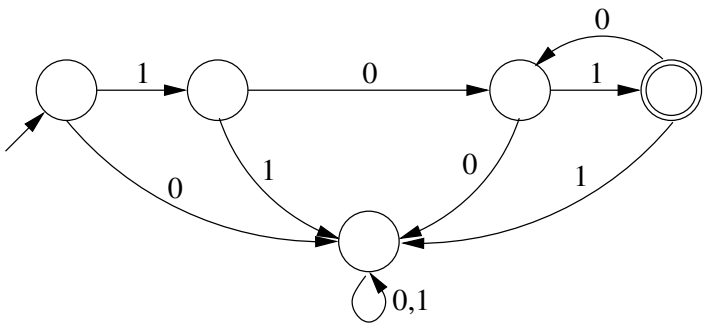


FIGURE III.19. Expression régulière du langage accepté.

représentations en base 2 d'entiers, en déduire les propriétés arithmétiques de l'ensemble d'entiers accepté.

CHAPITRE IV

Automate minimal

1. Introduction

Nous savons à présent qu'un langage est régulier si et seulement si il est accepté par un automate fini (et en particulier, déterministe). Cependant, plusieurs AFD peuvent accepter le même langage. La question posée ici est de rechercher parmi des automates équivalents, un automate qui serait, selon un sens encore à définir, *canonique*. Par exemple, les automates suivants acceptent tous le langage formé des mots ne comprenant pas deux a consécutifs.

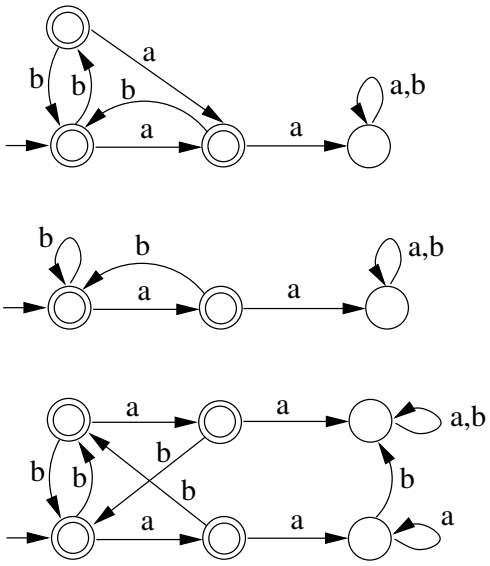


FIGURE IV.1. Trois AFD équivalents.

Il paraît naturel de vouloir minimiser le nombre d'états d'un AFD acceptant un langage régulier donné. En effet, lors de constructions comme le produit d'automates, il est préférable d'avoir peu d'états à traiter pour diminuer la taille de l'automate résultant. Nous allons montrer qu'à isomorphisme près, il n'existe qu'un seul AFD acceptant un langage donné et possédant un nombre minimum d'états. Notons encore que la notion d'automate minimal peut être définie pour un langage quelconque et pas uniquement pour un langage régulier.

2. Congruence syntaxique

Définition IV.2.1. Soit $L \subseteq \Sigma^*$ un langage arbitraire. Si w est un mot sur Σ, on dénote par $w^{-1}.L$ l'ensemble des mots qui, concaténés avec w, appartiennent à L, i.e.,

$$\boxed{w^{-1}.L = \{u \in \Sigma^* \mid wu \in L\}.}$$

On définit une relation sur Σ^*, notée $\sim_L$, de la manière suivante. Pour tous $x, y \in \Sigma^*$,

$$x \sim_L y \Leftrightarrow x^{-1}.L = y^{-1}.L.$$

En d'autres termes, $x \sim_L y$ si et seulement si pour tout mot $w \in \Sigma^*$, $xw \in L \Leftrightarrow yw \in L$.

Notons que la notation la plus répandue dans la littérature est $w^{-1}L$.

Remarque IV.2.2. Avec une telle définition, la formule suivante est alors immédiate (où la somme représente l'union),

$$L = \sum_{\sigma \in \Sigma} \sigma\left(\sigma^{-1}.L\right) + \delta(L), \qquad \text{avec } \delta(L) = \begin{cases} \varepsilon & \text{, si } \varepsilon \in L \\ \emptyset & \text{, sinon.} \end{cases}$$

et J. H. Conway d'écrire "*both Taylor's theorem and the mean value theorem*".

Proposition IV.2.3. *Soit $L \subseteq \Sigma^*$ un langage. La relation $\sim_L$ est une relation d'équivalence. Il s'agit même d'une congruence*[1] *à droite, i.e.,*

$$\forall z \in \Sigma^*,\ x \sim_L y \Rightarrow xz \sim_L yz.$$

Démonstration. C'est immédiat.

■

Remarque IV.2.4. On parle souvent pour $\sim_L$ de la *congruence de Nerode*. On note $[w]_L$ la classe d'équivalence du mot w pour la relation $\sim_L$,

$$[w]_L = \{u \in \Sigma^* \mid u \sim_L w\}.$$

Exemple IV.2.5. Soit le langage

$$L = \{w \in \{a, b\}^* \,:\, |w|_a \equiv 0 \pmod 3\}.$$

Pour ce langage, on a par exemple

$abbaba \sim_L aaa$	car $abbaba^{-1}.L = aaa^{-1}.L = L$
$b \not\sim_L ab$	car pour $u = aa$, $bu \notin L$ et $abu \in L$
$aba \not\sim_L bab$	car pour $u = a$, $abau \in L$ et $babu \notin L$
$a \sim_L ababaa$	car $a^{-1}.L = ababaa^{-1}.L = \{w \in \{a, b\}^* \,:\, \|w\|_a \equiv 2 \pmod 3\}$.

[1] Pour rappel, une congruence est une relation d'équivalence qui préserve les opérations de la structure algébrique considérée.

En effet, pour $w \in \{a,b\}^*$,

$$\begin{array}{lllll} \text{si } |w|_a \equiv_3 0, & \text{alors} & w^{-1}.L & = & \{u \in \{a,b\}^* : |u| \equiv_3 0\} \\ & \text{et} & [w]_L & = & \{u \in \{a,b\}^* : |u| \equiv_3 0\}, \\ \text{si } |w|_a \equiv_3 1, & \text{alors} & w^{-1}.L & = & \{u \in \{a,b\}^* : |u| \equiv_3 2\} \\ & \text{et} & [w]_L & = & \{u \in \{a,b\}^* : |u| \equiv_3 1\}, \\ \text{si } |w|_a \equiv_3 2, & \text{alors} & w^{-1}.L & = & \{u \in \{a,b\}^* : |u| \equiv_3 1\} \\ & \text{et} & [w]_L & = & \{u \in \{a,b\}^* : |u| \equiv_3 2\}. \end{array}$$

Cet exemple nous montre qu'en général, $w^{-1}.L \neq [w]_L$.

Définition IV.2.6. Dans le cas d'un automate déterministe (fini ou non) $\mathcal{A} = (Q, q_0, F, \Sigma, \delta)$, par analogie avec la notation $w^{-1}.L$, on utilise la notation suivante. Si $q \in Q$ est un état de $\mathcal{A}$ et si $G \subseteq Q$ est un sous-ensemble d'états, on note $q^{-1}.G$, l'ensemble des mots qui sont labels des chemins débutant en q et aboutissant dans un état de G, i.e.,

$$\boxed{q^{-1}.G = \{w \in \Sigma^* \mid \delta(q,w) \in G\}.}$$

On définit sur Q une relation d'équivalence comme suit : si $p, q \in Q$, alors

$$p \sim_{\mathcal{A}} q \Leftrightarrow p^{-1}.F = q^{-1}.F.$$

Remarque IV.2.7. Avec la notation que nous venons d'introduire, le langage accepté par l'automate déterministe $\mathcal{A} = (Q, q_0, F, \Sigma, \delta)$ est simplement

$$q_0^{-1}.F.$$

Lemme IV.2.8. *Soit $\mathcal{A} = (Q, q_0, F, \Sigma, \delta)$ un automate déterministe acceptant un langage L. Si $q \in Q$ et $w \in \Sigma^*$ sont tels que $\delta_L(q_0, w) = q$, alors*

$$\boxed{q^{-1}.F = w^{-1}.L.}$$

Démonstration. En effet, par définition, $q^{-1}.F = \{u \in \Sigma^* \mid \delta(q,u) \in F\}$. Or $\delta(q_0, w) = q$. Ainsi, pour tout $u \in q^{-1}.F$, on a

$$\delta(q_0, wu) = \delta(\delta(q_0, w), u) = \delta(q, u) \in F$$

et donc wu appartient à $L(\mathcal{A}) = L$, c'est-à-dire, u appartient à $w^{-1}.L$ et réciproquement.

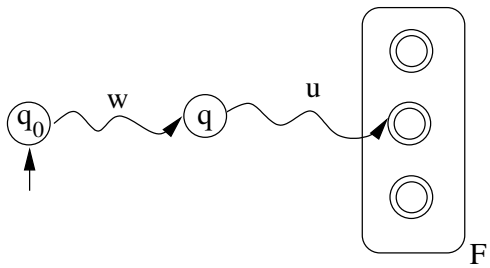


FIGURE IV.2. $q^{-1}.F = w^{-1}.L$ si $\delta(q_0, w) = q$.

■

Lemme IV.2.9. *Soient $L \subseteq \Sigma^*$ un langage et u, v deux mots sur Σ. On a*

$$\boxed{(uv)^{-1}.L = v^{-1}.(u^{-1}.L).}$$

Démonstration. Si w appartient à $(uv)^{-1}.L$, cela signifie que uvw appartient à L. En d'autres termes, vw appartient à $u^{-1}.L$ et ainsi w appartient à $v^{-1}.(u^{-1}.L)$. La démonstration de l'autre inclusion est identique.

■

Remarque IV.2.10. Pour l'opération $\sigma^{-1}.L$ (σ étant une lettre), on trouve parfois une terminologie rappelant le calcul différentiel[2]. Soit σ une lettre. On parle parfois de *dérivé* et l'on note $D_\sigma L$ pour $\sigma^{-1}.L$. La raison en est simple, il est clair que

$$D_\sigma(L + M) = D_\sigma L + D_\sigma M$$

et

$$D_\sigma(LM) = (D_\sigma L)\, M + \delta(L)\, D_\sigma M$$

où, une fois encore, somme et produit représentent respectivement l'union et la concaténation.

3. Automate minimal

Nous allons tirer parti de la congruence de Nerode introduite à la section précédente pour définir un automate particulier, à savoir l'automate minimal du langage L. La définition peut à première vue sembler artificielle, mais nous allons montrer qu'ainsi introduit, l'automate minimal jouit de propriétés fort intéressantes.

Définition IV.3.1. On définit l'*automate minimal*

$$\mathcal{A}_L = (Q_L, q_{0,L}, F_L, \Sigma, \delta_L)$$

d'un langage $L \subseteq \Sigma^*$ comme suit :

- $Q_L = \{w^{-1}.L \mid w \in \Sigma^*\}$,
- $q_{0,L} = \varepsilon^{-1}.L = L$,
- $F_L = \{w^{-1}.L \mid w \in L\} = \{q \in Q_L \mid \varepsilon \in q\}$,
- $\delta_L(q, \sigma) = \sigma^{-1}.q$, pour tous $q \in Q_L, \sigma \in \Sigma$.

Grâce au lemme IV.2.9, la fonction de transition de l'automate s'étend à $Q_L \times \Sigma^*$ par

$$\delta_L(q, w) = w^{-1}.q \quad , \forall q \in Q_L, w \in \Sigma^*.$$

Nous devons vérifier que cette définition a un sens en montrant que la fonction de transition ne dépend pas du représentant choisi. Ainsi, si un état de $\mathcal{A}_L$ est de la forme $x^{-1}.L = y^{-1}.L$ ($x, y \in \Sigma^*$), alors $x \sim_L y$.

[2] cf. J. A. Brzozowski, Derivatives of regular expressions, *J. of the Assoc. for Comp. Machinery* **11** (1964), 481–494.

Puisque $\sim_L$ est une congruence à droite, pour tout $\sigma \in \Sigma$, $x\sigma \sim_L y\sigma$ et donc $(x\sigma)^{-1}.L = (y\sigma)^{-1}.L$. En appliquant le lemme IV.2.9, on trouve bien $\sigma^{-1}.(x^{-1}.L) = \sigma^{-1}.(y^{-1}.L)$.

Remarque IV.3.2. Au vu de la définition de $\sim_L$, il est clair que l'ensemble des états de $\mathcal{A}$, $\{w^{-1}.L \mid w \in \Sigma^*\}$, est en bijection avec l'ensemble quotient $\Sigma^*/\sim_L = \{[w]_L \mid w \in \Sigma^*\}$. En effet, à chaque classe d'équivalence $[w]_L$ pour $\sim_L$ correspond un état $w^{-1}.L$ de l'automate minimal $\mathcal{A}_L$ et réciproquement. C'est pour cette raison que, dans la littérature, on trouve également une définition de l'automate minimal en termes des classes d'équivalence de $\sim_L$. Ainsi, on aurait pu définir l'automate minimal comme suit :

- $Q_L = \{[w]_L \mid w \in \Sigma^*\}$
- $q_{0,L} = [\varepsilon]_L$
- $F_L = \{[w]_L \mid w \in L\}$
- $\delta_L([w]_L, \sigma) = [w\sigma]_L$.

Cette dernière définition est équivalente à celle donnée en IV.3.1 car si $[w]_L$ correspond à $w^{-1}.L$, alors $[w\sigma]_L$ correspond à $(w\sigma)^{-1}.L = \sigma^{-1}.(w^{-1}.L)$. Dans la suite, nous utiliserons principalement la définition de l'automate minimal donnée en IV.3.1.

Remarquons encore que si $x \sim_L y$, alors

$$\delta_L(q_{0,L}, x) = \delta_L(q_{0,L}, y)$$

car il suffit de se rappeler que $q_{0,L} = L$ et dès lors, il vient

$$\delta_L(q_{0,L}, x) = x^{-1}.L = y^{-1}.L = \delta_L(q_{0,L}, y).$$

Exemple IV.3.3. Poursuivons l'exemple IV.2.5. Il est facile de voir que pour le langage L formé des mots sur $\{a, b\}$ contenant un nombre de a multiple de trois, la congruence de Nerode possède trois classes d'équivalence

$$[\varepsilon]_L,\ [a]_L \text{ et } [aa]_L.$$

Dit autrement, l'automate minimal $\mathcal{A}_L$ a trois états

$$\varepsilon^{-1}.L,\ a^{-1}.L \text{ et } aa^{-1}.L.$$

Pour définir la fonction de transition, on a

$$\begin{array}{ll}
\delta_L(\varepsilon^{-1}.L, a) = a^{-1}.(\varepsilon^{-1}.L) = a^{-1}.L & \\
\delta_L(\varepsilon^{-1}.L, b) = b^{-1}.(\varepsilon^{-1}.L) = b^{-1}.L = \varepsilon^{-1}.L & \text{car } \varepsilon \sim_L b \\
\delta_L(a^{-1}.L, a) = a^{-1}.(a^{-1}.L) = aa^{-1}.L & \\
\delta_L(a^{-1}.L, b) = b^{-1}.(a^{-1}.L) = ab^{-1}.L = a^{-1}.L & \text{car } a \sim_L ab \\
\delta_L(aa^{-1}.L, a) = a^{-1}.(aa^{-1}.L) = aaa^{-1}.L = \varepsilon^{-1}.L & \text{car } \varepsilon \sim_L aaa \\
\delta_L(aa^{-1}.L, b) = b^{-1}.(aa^{-1}.L) = aab^{-1}.L = aa^{-1}.L & \text{car } aa \sim_L aab.
\end{array}$$

Si on note $1, 2, 3$ les trois langages $\varepsilon^{-1}.L = L, a^{-1}.L, aa^{-1}.L$, on obtient l'automate représenté à la figure IV.3.

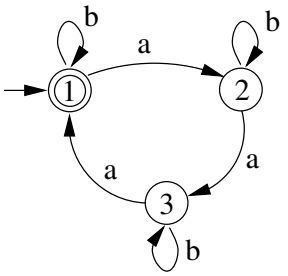


FIGURE IV.3. Un automate minimal.

Remarque IV.3.4. On observe que, dans la définition de $\mathcal{A}_L$, les états de l'automate minimal de L sont des ensembles de mots. Dans l'exemple précédent, on a un nombre fini d'états et chaque état correspond à un ensemble infini de mots.

Exemple IV.3.5. Considérons le langage L formé des mots sur $\{a, b\}$ ayant même nombre de a que de b, i.e.,

$$L = \{w \in \{a, b\}^* \,:\, |w|_a = |w|_b\}.$$

Une application immédiate du lemme de la pompe montre que ce langage n'est pas régulier. On peut néanmoins rechercher son automate minimal puisque la relation $\sim_L$ est définie pour tout langage L. On s'aperçoit que le nombre de classes d'équivalence pour la relation $\sim_L$ est infini. En effet, pour tout $n \in \mathbb{Z}$,

$$c_n := [a^i b^j]_L, \text{ avec } i - j = n$$

est une classe d'équivalence et il est clair que si $m \neq n$, alors $c_m \neq c_n$. De plus,

$$\delta_L((a^i b^j)^{-1}.L, a) = (a^{i+1} b^j)^{-1}.L = (a^i b^{j-1})^{-1}.L$$

et

$$\delta_L((a^i b^j)^{-1}.L, b) = (a^i b^{j+1})^{-1}.L = (a^{i-1} b^j)^{-1}.L.$$

(Dans les expressions ci-dessus, on ne considère que les expressions pour lesquelles les exposants sont positifs ou nuls.) Le seul état final de l'automate est $(a^i b^i)^{-1}.L = L$. L'automate minimal de L est représenté à la figure IV.4.

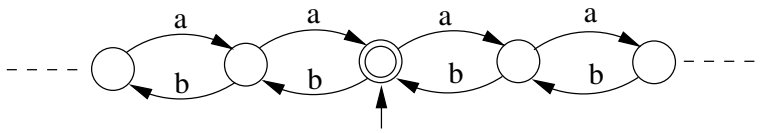


FIGURE IV.4. L'automate minimal d'un langage non régulier.

On peut d'ores-et-déjà remarquer que pour ce langage non régulier, le nombre d'états de l'automate minimal est infini.

Proposition IV.3.6. *L'automate minimal d'un langage $L \subseteq \Sigma^*$ accepte L.*

Démonstration. En effet, soit $w \in \Sigma^*$,

$$w \in L(\mathcal{A}_L) \Leftrightarrow \delta_L(q_{0,L}, w) \in F_L \Leftrightarrow w^{-1}.L \in F_L \Leftrightarrow w \in L.$$

On a utilisé le fait que

$$\delta_L(q_{0,L}, w) = \delta_L(\varepsilon^{-1}.L, w) = w^{-1}.(\varepsilon^{-1}.L) = (\varepsilon w)^{-1}.L.$$

■

Définition IV.3.7. Un automate déterministe $\mathcal{A} = (Q, q_0, F, \Sigma, \delta)$ est *accessible* si pour tout état $q \in Q$, il existe un mot $w \in \Sigma^*$ tel que $\delta(q_0, w) = q$.

Un automate déterministe $\mathcal{A} = (Q, q_0, F, \Sigma, \delta)$ est *réduit* si pour tous $p, q \in Q$,

$$p^{-1}.F = q^{-1}.F \text{ entraîne } p = q.$$

En d'autres termes, un AFD est réduit, si les langages acceptés depuis deux états distincts sont distincts ou encore si chaque classe d'équivalence pour la relation $\sim_{\mathcal{A}}$ sur Q est un singleton.

Le résultat suivant justifie l'appellation "minimal".

Théorème IV.3.8. *Soient $L \subseteq \Sigma^*$ un langage et $\mathcal{A}_L = (Q_L, q_{0,L}, F_L, \Sigma, \delta_L)$ son automate minimal. Si $\mathcal{B} = (Q, q_0, F, \Sigma, \delta)$ est un automate accessible et déterministe acceptant L, alors il existe une application $\Phi : Q \to Q_L$ telle que*

- *Φ est surjectif,*
- *$\Phi(q_0) = q_{0,L}$,*
- *$\forall \sigma \in \Sigma, \forall q \in Q : \Phi(\delta(q, \sigma)) = \delta_L(\Phi(q), \sigma)$,*
- *$\Phi(F) = F_L$.*

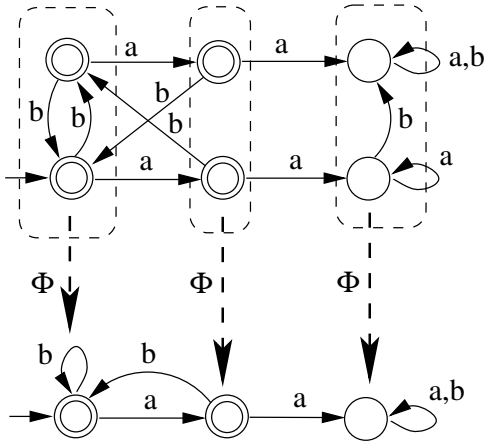


FIGURE IV.5. Une application Φ satisfaisant les propriétés du théorème IV.3.8.

Démonstration. Puisque $\mathcal{B}$ est accessible, pour tout état $q \in Q$, il existe un mot $w \in \Sigma^*$ tel que $\delta(q_0, w) = q$. Supposons tout d'abord qu'une application Φ satisfaisant les propriétés énoncées existe. Dans ce cas[3],

On effectue d'abord l'analyse.

$$\Phi(q) = \Phi(\delta(q_0, w)) = \delta_L(\Phi(q_0), w) = \delta_L(q_{0,L}, w) = w^{-1}.L = q^{-1}.F$$

où pour la dernière égalité, on a appliqué le lemme IV.2.8. Montrons à présent que l'application

Passons à la synthèse.

$$\Phi : Q \to Q_L : q \mapsto q^{-1}.F$$

possède les propriétés indiquées :

- il est clair que Φ est à valeurs dans Q_L car $\mathcal{B}$ étant accessible, il est toujours possible d'écrire $q^{-1}.F$ sous la forme $w^{-1}.L$ pour un certain $w \in \Sigma^*$.
- On a $\Phi(q_0) = q_0^{-1}.F = L = q_{0,L}$.
- Soient $\sigma \in \Sigma$ et $q \in Q$. Par définition de Φ, on a tout d'abord

  $$\Phi(\delta(q, \sigma)) = (\delta(q, \sigma))^{-1}.F$$

  Si $w \in \Sigma^*$ est tel que $\delta(q_0, w) = q$, alors $\delta(q_0, w\sigma) = \delta(q, \sigma)$ et par le lemme IV.2.8,

  $$(\delta(q, \sigma))^{-1}.F = (w\sigma)^{-1}.L.$$

  Par le lemme IV.2.9, $(w\sigma)^{-1}.L = \sigma^{-1}.(w^{-1}.L)$ et si on applique à nouveau le lemme IV.2.8, $w^{-1}.L = q^{-1}.F$. Par conséquent,

  $$\Phi(\delta(q, \sigma)) = \sigma^{-1}.(q^{-1}.F) = \sigma^{-1}.\Phi(q) = \delta_L(\Phi(q), \sigma).$$

- Montrons que Φ est surjectif. Soit $q \in Q_L$. Cet état est de la forme $w^{-1}.L$ pour un mot $w \in \Sigma^*$. Soit r l'état de $\mathcal{B}$ tel que $\delta(q_0, w) = r$. Il vient

  $$\Phi(r) = r^{-1}.F = w^{-1}.L = q.$$

- Un état q de $\mathcal{B}$ est final si et seulement si il existe $w \in L$ tel que $\delta(q_0, w) = q$. Soit q un tel état. Ainsi,

  $$\Phi(q) = q^{-1}.F = w^{-1}.L \in F_L \quad \text{et} \quad \Phi(F) \subseteq F_L.$$

  Considérons à présent un état q de $\mathcal{A}_L$ tel que $q \in F_L$. Puisque Φ est surjectif, il existe un état $p \in Q$ de $\mathcal{B}$ tel que $\Phi(p) = p^{-1}.F = q$. Par définition de l'automate minimal, $p^{-1}.F$ appartient à F_L si et seulement si $\varepsilon \in p^{-1}.F$ ce qui signifie que p est un état final de $\mathcal{B}$.

■

Corollaire IV.3.9. *Si L est un langage régulier accepté par un AFD $\mathcal{B}$, alors le nombre d'états de $\mathcal{B}$ est minoré par le nombre d'états de $\mathcal{A}_L$.*

Démonstration. Cela découle immédiatement de la surjectivité de l'application Φ introduite à la proposition précédente.

■

[3] En particulier, ceci prouve que si une telle application Φ existe, alors elle est unique.

Proposition IV.3.10. *Soit $L \subseteq \Sigma^*$ un langage.*

(i) *L'automate minimal $\mathcal{A}_L = (Q_L, q_{0,L}, F_L, \Sigma, \delta_L)$ de L est accessible et réduit.*

(ii) *Soit $\mathcal{B} = (Q, q_0, F, \Sigma, \delta)$ un automate déterministe accessible acceptant L. Cet automate est réduit si et seulement si l'application $\Phi : Q \to Q_L$ définie au théorème IV.3.8 est une bijection. Dans ce cas, les automates $\mathcal{A}_L$ et $\mathcal{B}$ sont isomorphes.*

Démonstration. L'automate minimal est accessible car un état quelconque de $\mathcal{A}_L$ est de la forme $w^{-1}.L$ pour un mot $w \in \Sigma^*$ et

$$\delta_L(q_{0,L}, w) = w^{-1}.L.$$

Par définition de l'ensemble d'états Q_L, il est clair que $\mathcal{A}_L$ est réduit.

Si $\mathcal{B}$ est un automate accessible, l'application $\Phi : Q \to Q_L$ introduite au théorème IV.3.8 est surjective. Cette application est injective si et seulement si pour tous $p, q \in Q$,

$$\Phi(p) = \Phi(q) \Rightarrow p = q$$

ce qui se réécrit

$$p^{-1}.F = q^{-1}.F \Rightarrow p = q$$

et qui signifie que $\mathcal{B}$ est réduit.

■

Proposition IV.3.11. *Un langage $L \subseteq \Sigma^*$ est régulier si et seulement si son automate minimal $\mathcal{A}_L$ est fini.*

Démonstration. Si $\mathcal{A}_L$ est fini, au vu de la proposition IV.3.6, L est accepté par $\mathcal{A}_L$ qui est en particulier un AFD. Par le théorème de Kleene, L est régulier.

Passons à la réciproque et supposons L régulier et accepté par un AFD $\mathcal{A}$ que l'on peut, sans aucune restriction, supposer accessible. Dès lors, au vu du théorème IV.3.8, l'automate minimal de L est fini.

■

Ce dernier résultat peut se réénoncer comme suit.

Théorème IV.3.12 (Myhill-Nerode). *Un langage L est régulier si et seulement si la congruence $\sim_L$ est d'indice fini (i.e., possède un nombre fini de classes d'équivalence).*

Démonstration. Cela résulte immédiatement de la proposition précédente et de la remarque IV.3.2.

■

4. Construction de l'automate minimal

La proposition IV.3.10 fournit un moyen de construire l'automate minimal d'un langage régulier L à partir d'un AFD $\mathcal{A}$ acceptant L. En effet, il suffit de pouvoir trouver un AFD accessible et réduit équivalent. Tout d'abord, il est facile de rendre un AFD donné accessible. Il suffit de passer en revue les états qui peuvent être atteints depuis l'état initial et d'éliminer les autres états (inaccessibles). Classiquement, un algorithme de *recherche en profondeur* suffit. On construit un arbre ayant pour racine l'état initial de $\mathcal{A}$. Dans cet arbre, les fils d'un noeud sont les états accessibles depuis celui-ci et on arrête la construction lorsqu'à un niveau de l'arbre, il n'apparaît plus de nouveaux états par rapport aux niveaux précédents.

La question qui se pose est donc de pouvoir déterminer si un automate fini déterministe donné $\mathcal{A} = (Q, q_0, F, \Sigma, \delta)$ est réduit. Par définition de la relation $\sim_{\mathcal{A}}$ sur Q, l'automate est réduit si pour tout couple (p, q) d'états avec $p \neq q$,

$$p \not\sim_A q.$$

En particulier, $p \not\sim_A q$ s'il existe un mot $w \in \Sigma^*$ tel que

$$\delta(p, w) \in F \text{ et } \delta(q, w) \notin F$$

ou

$$\delta(p, w) \notin F \text{ et } \delta(q, w) \in F.$$

On dit alors qu'un tel mot *distingue* les états p et q ou encore que le couple (p, q) est distingué. Dans l'algorithme qui suit, on notera $\mathcal{N}_\ell$ l'ensemble des couples d'états qui sont distingués par un mot de longueur ℓ et qui ne sont distingués par aucun mot plus court.

Algorithme de recherche des états équivalents

(1) *Initialisation* : lors de cette étape, on détermine les couples d'états distingués par le mot vide (seul mot de longueur $\ell = 0$).

- On pose $\ell := 0$.
- Pour tout $p \in F$ et tout $q \in Q \backslash F$, la paire (p, q) est distinguée car le mot vide appartient à $p^{-1}.F$ mais pas à $q^{-1}.F$. Soit $\mathcal{N}_0$ l'ensemble de ces paires.

(2) *Incrémentation* : on détermine les couples d'états distingués par un mot de longueur $\ell + 1$ et non distingués par un mot de longueur ℓ.

- Si $\mathcal{N}_\ell = \emptyset$, l'algorithme s'achève[4].

[4]On doit remarquer que si $\mathcal{N}_\ell$ est vide, alors il en est de même pour $\mathcal{N}_{\ell+1}$ et donc aussi pour tous les suivants. En effet, supposons au contraire que $\mathcal{N}_\ell = \emptyset$ et $\mathcal{N}_{\ell+1} \neq \emptyset$. Il existe donc $(r, s) \in \mathcal{N}_{\ell+1}$ distingué par un mot σw de longueur $\ell + 1$. Dès lors, le mot w de longueur ℓ distingue les états $\delta(r, \sigma) = r'$ et $\delta(s, \sigma) = s'$. Puisque $\mathcal{N}_\ell = \emptyset$, on en conclut que r' et s' doivent être distingués par un mot w' de longueur $< \ell$. Mais dans ce cas, r et s sont aussi distingués par $\sigma w'$ de longueur $\leq \ell$, ce qui est absurde.

- Sinon, pour chaque paire $(p, q) \in \mathcal{N}_\ell$, on passe en revue les paires (r, s) avec $r \neq s$ qui n'appartiennent pas à $\mathcal{N}_0 \cup \cdots \cup \mathcal{N}_\ell$. S'il existe $\sigma \in \Sigma$ tel que

$$\delta(r, \sigma) = p \text{ et } \delta(s, \sigma) = q$$

ou

$$\delta(s, \sigma) = p \text{ et } \delta(r, \sigma) = q,$$

alors la paire (r, s) est distinguée par un mot de longueur $\ell + 1$. Soit $\mathcal{N}_{\ell+1}$ l'ensemble de ces paires.
- Remplacer ℓ par $\ell + 1$ et répéter **(2)**.

Exemple IV.4.1. Appliquons l'algorithme précédent à l'AFD représenté à la figure IV.6.

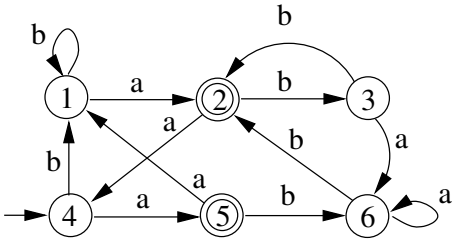


FIGURE IV.6. Un AFD dont on recherche les états équivalents pour $\sim_{\mathcal{A}}$.

La première étape donne le tableau suivant. Dans ce tableau, on dénote simplement par **i** l'appartenance d'un couple à l'ensemble $\mathcal{N}_i$, $i \in \mathbb{N}$. De plus, par raison de symétrie, on s'intéressera uniquement à la partie située au dessus de la diagonale principale.

	1	2	3	4	5	6
1	$*$	**0**			**0**	
2		$*$	**0**	**0**		**0**
3			$*$		**0**	
4				$*$	**0**	
5					$*$	**0**
6						$*$

Puisque $(1.a, 3.a) = (2, 6)$, $(1.b, 6.b) = (1, 2)$, $(3.a, 4.a) = (6, 5)$ et $(4, 6) = (5, 6)$, on a, à la deuxième étape, le tableau ci-dessous.

	1	2	3	4	5	6
1	$*$	**0**	**1**		**0**	**1**
2		$*$	**0**	**0**		**0**
3			$*$	**1**	**0**	
4				$*$	**0**	**1**
5					$*$	**0**
6						$*$

L'algorithme s'achève car $(1.a, 4.a) = (2, 5)$, $(1.b, 4.b) = (1, 1)$, $(2.a, 5.a) = (1, 4)$, $(2.b, 5.b) = (3, 6)$, $(3.a, 6.a) = (6, 6)$ et $(3.b, 6.b) = (2, 2)$. On en conclut que $1 \sim_{\mathcal{A}} 4$, $2 \sim_{\mathcal{A}} 5$ et $3 \sim_{\mathcal{A}} 6$.

Puisque nous pouvons supposer avoir un AFD $\mathcal{A}$ accessible, le théorème IV.3.8 nous affirme que l'automate $\mathcal{A}$ se projette au moyen de l'application Φ sur l'automate minimal du langage L accepté par $\mathcal{A}$ et que des états de $\mathcal{A}$ équivalents pour $\sim_{\mathcal{A}}$ sont envoyés sur un même état de $\mathcal{A}_L$. Ainsi, les états de $\mathcal{A}_L$ vont correspondre aux classes d'équivalence de $\sim_{\mathcal{A}}$.

Toujours en vertu du théorème IV.3.8, les transitions de l'automate minimal sont définies par

$$\delta_L(\Phi(q), \sigma) = \Phi(\delta(q, \sigma))$$

si δ (resp. δ_L) est la fonction de transition de $\mathcal{A}$ (resp. $\mathcal{A}_L$). Traduit en termes d'états équivalents, cela signifie que si un état de $\mathcal{A}_L$ correspond à une classe d'équivalence $[q]_{\mathcal{A}}$ pour la relation $\sim_{\mathcal{A}}$, alors la lecture de σ depuis cet état dans $\mathcal{A}_L$ conduit à l'état correspondant à la classe $[q.\sigma]_{\mathcal{A}}$.

Exemple IV.4.2. Si nous continuons l'exemple précédent, on a $[1]_{\mathcal{A}} = \{1, 4\}$, $[2]_{\mathcal{A}} = \{2, 5\}$ et $[3]_{\mathcal{A}} = \{3, 6\}$. Puisque dans l'automate de départ, $1.a = 2$ et $1.b = 1$, on a $\delta_L(\Phi(1), a) = \Phi(2)$ et $\delta_L(\Phi(1), b) = \Phi(1)$. Ceci signifie que, dans l'automate minimal, la lecture de a (resp. b) depuis l'état correspondant à $\{1, 4\}$ conduit à $[2]_{\mathcal{A}} = \{2, 5\}$ (resp. $[1]_{\mathcal{A}} = \{1, 4\}$). En continuant de la sorte, on obtient l'automate de la figure IV.7.

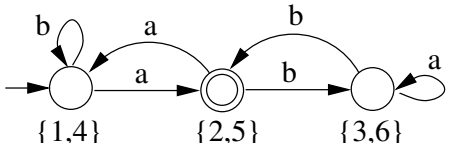


FIGURE IV.7. Un automate minimal.

Exemple IV.4.3. Soit l'AFD accessible $\mathcal{A}$ représenté à la figure IV.8. Nous allons lui appliquer l'algorithme de recherche des états équivalents pour

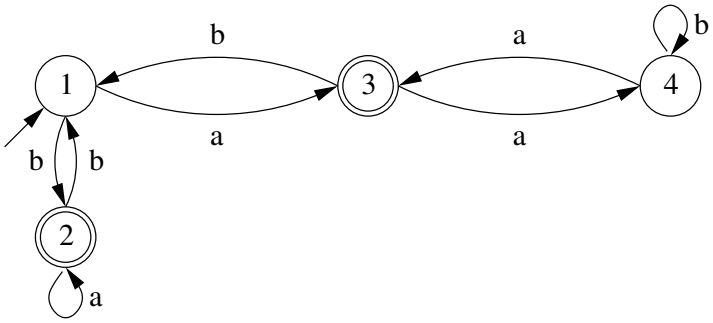


FIGURE IV.8. Un AFD accessible $\mathcal{A}$.

montrer qu'il est réduit (et qu'il s'agit donc d'un automate minimal puisqu'il est visiblement accessible). Avec les mêmes notations que précédemment, on obtient rapidement le tableau suivant.

	1	2	3	4
1	$*$	**0**	**0**	**1**
2		$*$	**1**	**0**
3			$*$	**0**
4				$*$

4.1. Une autre procédure de minimisation.

Proposition IV.4.4. *Soit $\mathcal{A}$ un AFD acceptant un langage L. Si pour tout automate $\mathcal{B}$, $\mu(\mathcal{B})$ désigne l'automate déterministe équivalent à $\mathcal{B}^R$ obtenu par construction des sous-ensembles d'états, alors $\mu(\mu(\mathcal{A}))$ est l'automate minimal de L.*

Démonstration. Si $\mathcal{B}$ est un AFD acceptant M, il est clair que $\mu(\mathcal{B})$ accepte M^R et qu'il est accessible. En effet, dans la procédure de construction par sous-ensembles, on ne considère que les états accessibles car on recherche de proche en proche les états atteints depuis l'état initial. Il suffit dès lors de montrer que si $\mathcal{B}$ est un AFD accessible acceptant un langage M, alors $\mu(\mathcal{B})$ est un AFD accessible et réduit acceptant M^R. Dans ce cas, $\mu(\mathcal{A})$ sera un AFD accessible acceptant L^R et $\mu(\mu(\mathcal{A}))$ sera un AFD accessible et réduit acceptant $(L^R)^R = L$. On conclut alors par la proposition IV.3.10.

Soit $\mathcal{B}$ un AFD accessible. Montrons que $\mu(\mathcal{B})$ est réduit. Soient P, Q deux états de $\mu(\mathcal{B})$. Supposons que $P^{-1}.\mathcal{F} = Q^{-1}.\mathcal{F}$ (où $\mathcal{F}$ désigne l'ensemble des états finals de $\mu(\mathcal{B})$). De par la construction par sous-ensembles, l'état P (resp. Q) est constitué d'états[5] $p_1, \ldots, p_r$ (resp. $q_1, \ldots, q_s$) de $\mathcal{B}^R$ et un état est final s'il est un sous-ensemble d'états de $\mathcal{B}^R$ contenant un état final de $\mathcal{B}^R$ (donc ici contenant l'état initial q_0 de $\mathcal{B}$, i.e., l'unique état final de $\mathcal{B}^R$).

Si w appartient à $P^{-1}.\mathcal{F}$, cela signifie que, dans $\mu(\mathcal{B})$, w est le label d'un chemin débutant dans P et aboutissant dans un état final. Encore un fois, de par la construction par sous-ensembles, cela signifie que dans $\mathcal{B}^R$ on a un chemin de label w débutant dans un des états p_i et aboutissant dans q_0. Ou de manière équivalente, dans $\mathcal{B}$, w^R est le label d'un chemin débutant dans q_0 et aboutissant dans p_i. Réciproquement, pour tout $i \in \{1, \ldots, r\}$, si w est label d'un chemin dans $\mathcal{B}$ débutant dans q_0 et aboutissant dans p_i, alors dans $\mu(\mathcal{B})$, w^R appartient à $P^{-1}.F$. Autrement dit, on a

$$P^{-1}.\mathcal{F} = (q_0^{-1}.\{p_1, \ldots, p_r\})^R$$

où dans le membre de gauche (resp. de droite), on considère l'automate $\mu(\mathcal{B})$ (resp. $\mathcal{B}$).

Dans $\mathcal{B}$, pour tout $i \in \{1, \ldots, r\}$, si $q_0.w = p_i$, cela signifie qu'il appartient à $q_0^{-1}.\{p_1, \ldots, p_r\}$ et puisque nous avons supposer que $P^{-1}.\mathcal{F} = Q^{-1}.\mathcal{F}$,

[5] Les automates $\mathcal{B}$ et $\mathcal{B}^R$ ont le même ensemble d'états.

on en déduit que w appartient à $q_0^{-1}.\{q_1, \ldots, q_s\}$. Il existe $j \in \{1, \ldots, s\}$ tel que, dans $\mathcal{B}$, $q_0.w = p_j$. Or puisque $\mathcal{B}$ est déterministe, on trouve $p_i = q_j$ et $P \subset Q$. On procède de même pour l'autre inclusion et ainsi, $P = Q$.

■

Exemple IV.4.5. Appliquons la proposition précédente pour obtenir l'automate minimal du langage accepté par l'automate représenté à la figure IV.9. Tout d'abord, l'automate miroir $\mathcal{A}^R$ est donné par l'automate de la

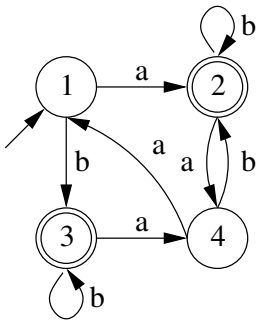


FIGURE IV.9. Un AFD $\mathcal{A}$.

figure IV.10. Pour rendre $\mathcal{A}^R$ déterministe, on utilise la construction par

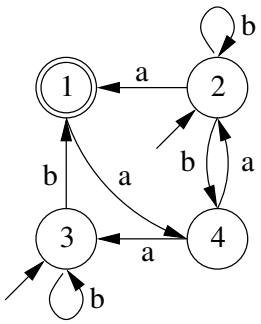


FIGURE IV.10. L'automate $\mathcal{A}^R$.

sous-ensembles. On trouve les ensembles d'états

$$\{2,3\},\ \{1\},\ \{1,2,3,4\},\ \{4\},\ \emptyset.$$

Si on renomme ces derniers $1, \ldots, 5$, on obtient l'AFD accessible $\mu(\mathcal{A})$ représenté à la figure IV.11. Considérons à présent le miroir de ce dernier automate pour obtenir celui de la figure IV.12. Pour conclure, il nous reste à rendre cet automate déterministe en utilisant une fois encore la construction par sous-ensembles. Les ensembles d'états sont ici,

$$\{2,3\},\ \{1,3\},\ \{3,4\}.$$

Une fois ces ensembles renommés $1, 2, 3$, on obtient l'automate de la figure IV.13 qui est l'automate minimal du langage accepté par l'automate $\mathcal{A}$ de départ.

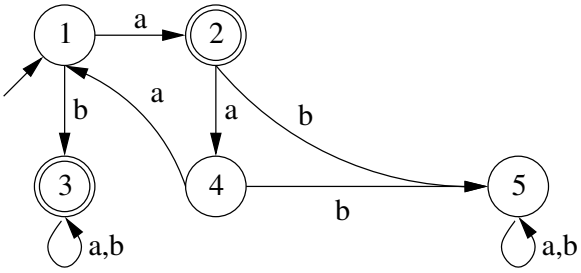


FIGURE IV.11. L'automate $\mu(\mathcal{A})$.

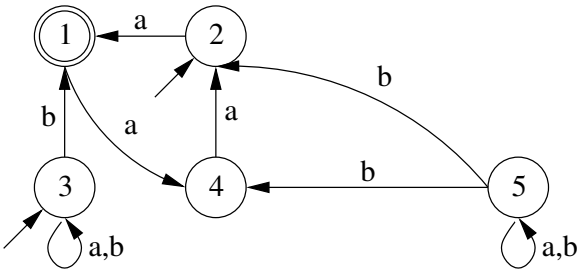


FIGURE IV.12. L'automate $(\mu(\mathcal{A}))^R$.

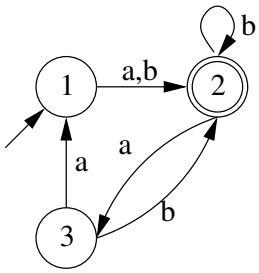


FIGURE IV.13. L'automate $\mu(\mu(\mathcal{A}))$.

5. Applications

Nous allons utiliser la théorie de l'automate minimal pour montrer que l'ensemble des langages réguliers est stable par morphisme inverse. Nous montrons également que l'ensemble des préfixes ou des suffixes d'un langage régulier est régulier. Enfin, la racine n-ième (que nous définirons le moment voulu) d'un langage régulier est encore un langage régulier.

Proposition IV.5.1. *Soit $f : \Sigma^* \to \Gamma^*$ un morphisme de monoïdes. Si $M \subset \Gamma^*$ est un langage régulier alors $f^{-1}(M)$ est aussi un langage régulier.*

Démonstration. Il suffit de montrer que l'automate minimal du langage $f^{-1}(M) \subset \Sigma^*$ est fini. Soit $w \in \Sigma^*$. On a

$$\begin{aligned} w^{-1}.f^{-1}(M) &= \{u \in \Sigma^* \mid wu \in f^{-1}(M)\} \\ &= \{u \in \Sigma^* \mid f(wu) \in M\} \\ &= \{u \in \Sigma^* \mid f(w)f(u) \in M\} \\ &= \{u \in \Sigma^* \mid f(u) \in f(w)^{-1}.M\} \\ &= f^{-1}(f(w)^{-1}.M) \end{aligned}$$

Or M est régulier donc son automate minimal est fini et $f(w)^{-1}.M$ ne peut prendre qu'un nombre fini de valeurs. On en conclut que, quel que soit $w \in \Sigma^*$, $w^{-1}.f^{-1}(M)$ ne peut prendre qu'un nombre fini de valeurs. Ainsi, l'automate minimal du langage $f^{-1}.M$ est fini.

■

Remarque IV.5.2. Profitons du résultat précédent pour énoncer, sans démonstration[6], un résultat assez étonnant concernant la représentation des langages réguliers. *Pour tout langage régulier L sur un alphabet quelconque, il existe des morphismes h_1, h_2, h_3, h_4 tels que*

$$L = h_4(h_3^{-1}(h_2(h_1^{-1}(a^*b)))).$$

On pourra comparer ce résultat avec le théorème de représentation de Chomsky-Schützenberger pour les langages algébriques (cf. théorème VI.10.3).

Ce résultat n'est pas nouveau! Mais la preuve est élégante.

Proposition IV.5.3. *Si $L \subseteq \Sigma^*$ est un langage régulier, alors* $\mathrm{Pref}(L)$ *est aussi régulier.*

Démonstration. Il suffit de montrer que l'automate minimal du langage $\mathrm{Pref}(L)$ est fini. Soit $w \in \Sigma^*$. Il vient

$$\begin{aligned} w^{-1}.\mathrm{Pref}(L) &= \{u \in \Sigma^* \mid wu \in \mathrm{Pref}(L)\} \\ &= \{u \in \Sigma^* \mid \exists v \in \Sigma^* : wuv \in L\} \\ &= \{u \in \Sigma^* \mid \exists v \in \Sigma^* : uv \in w^{-1}.L\} \\ &= \mathrm{Pref}(w^{-1}.L). \end{aligned}$$

Le langage L étant régulier, $w^{-1}.L$ ne peut prendre qu'un nombre fini de valeurs. Par conséquent, $\mathrm{Pref}(w^{-1}.L)$ ne prend aussi qu'un nombre fini de valeurs et l'ensemble

$$\{\mathrm{Pref}(w^{-1}.L) \mid w \in \Sigma^*\}$$

est fini.

■

[6] Voir par exemple, le *Handbook of formal languages*, vol. 1, pour des références.

Corollaire IV.5.4. *Si $L \subseteq \Sigma^*$ est un langage régulier, alors* $\mathrm{Suff}(L)$ *est aussi un langage régulier.*

Démonstration. Il suffit de remarquer que

$$\mathrm{Suff}(L) = (\mathrm{Pref}(L^R))^R.$$

Le résultat découle de la proposition précédente et du fait que l'ensemble des langages réguliers est stable par application du miroir.

■

Remarque IV.5.5. On pourra comparer ces deux derniers résultats avec les propositions II.3.9 et II.3.10 et leur preuve.

Définition IV.5.6. Soit $k \geq 1$. On définit la *racine k-ième* d'un langage $L \subseteq \Sigma^*$ par

$$\sqrt[k]{L} = \{u \in \Sigma^* \mid u^k \in L\}.$$

Exemple IV.5.7. Soit $L = a^*b^*$. Il est facile de vérifier que

$$\sqrt{L} = a^* \cup b^*.$$

On dispose du résultat suivant.

Proposition IV.5.8. *Si $L \subseteq \Sigma^*$ est un langage régulier, alors $\sqrt[k]{L}$ est aussi un langage régulier.*

Afin de démontrer ce résultat, nous avons besoin du lemme suivant.

Lemme IV.5.9. *Soit L un langage régulier. Si p est un état de l'automate minimal de L donné dans la définition IV.3.1 alors p et*

$$S(p) = \{w \in \Sigma^* \mid p = w^{-1}.L\}$$

sont deux langages réguliers.

Démonstration. Puisque p est un état de l'automate minimal $\mathcal{A}_L = (Q_L, q_{0,L}, F_L, \Sigma, \delta_L)$, il existe $w \in \Sigma^*$ tel que $p = w^{-1}.L$. En d'autres termes,

$$p = \{u \in \Sigma^* \mid wu \in L\} = \{u \in \Sigma^* \mid \delta_L(q_{0,L}, wu) \in F_L\}$$

ce qui signifie que ce langage est accepté par l'AFD

$$(Q_L, \delta_L(q_{0,L}, w), F_L, \Sigma, \delta_L)$$

et donc, ce langage est régulier.

Pour montrer que $S(p)$ est régulier, il suffit une fois encore de vérifier que son automate minimal est fini. Soit $u \in \Sigma^*$. Il vient

$$\begin{aligned} u^{-1}.S(p) &= \{v \in \Sigma^* \mid uv \in S(p)\} \\ &= \{v \in \Sigma^* \mid p = (uv)^{-1}.L\} \\ &= \{v \in \Sigma^* \mid p = v^{-1}.(u^{-1}.L)\}. \end{aligned}$$

Or L est régulier, son automate minimal est donc fini et $u^{-1}.L$ ne peut prendre qu'un nombre fini de valeurs distinctes. Par conséquent,

$$\{u^{-1}.S(p) \mid u \in \Sigma^*\}$$

est un ensemble fini.

■

Nous pouvons à présent démontrer la proposition IV.5.8.

Démonstration. Soit $\mathcal{A}_L$ l'automate minimal de L ayant Q_L pour ensemble d'états. Montrons tout d'abord que

$$\sqrt[k+1]{L} = \bigcup_{p \in Q_L} (S(p) \cap \sqrt[k]{p}).$$

Si u appartient à $\sqrt[k+1]{L}$, alors, par définition de la racine $(k+1)$-ième, uu^k appartient à L et donc u^k appartient à $u^{-1}.L$. Si on pose $p = u^{-1}.L$, cela signifie que u appartient à $\sqrt[k]{p}$ avec $p \in Q_L$. De plus, par définition même de $S(p)$, u appartient également à ce dernier ensemble.

Démontrons l'autre inclusion. Si u appartient au membre de droite, cela signifie que p s'écrit $u^{-1}.L$ et que u^k appartient à p. Par conséquent, u^k appartient à $u^{-1}.L$ et donc u^{k+1} appartient à L.

Pour conclure la preuve, on procède par récurrence sur k. Si $k = 1$, alors par hypothèse $\sqrt[1]{L} = L$ est régulier. Supposons à présent que, si L est régulier, $\sqrt[k]{L}$ $(k \geq 1)$ est régulier et montrons que $\sqrt[k+1]{L}$ l'est encore. Au vu du lemme précédent, pour tout état $p \in Q_L$, p et $S(p)$ sont réguliers. Par hypothèse de récurrence, $\sqrt[k]{p}$ est régulier et donc $S(p) \cap \sqrt[k]{p}$ est régulier car il s'agit de l'intersection de deux langages réguliers. La formule donnée ci-dessus ne fait ainsi intervenir qu'une union finie de langages réguliers (en effet, L est régulier et donc, Q_L est fini). Par conséquent, $\sqrt[k+1]{L}$ est régulier.

■

Voici à présent quelques remarques concernant la complexité des algorithmes à mettre en oeuvre pour rechercher l'automate minimal d'un langage régulier à partir d'un automate donné.

Remarque IV.5.10. On peut montrer que l'algorithme de recherche des états équivalents dans un AFD $\mathcal{A}$ est de complexité temporelle $\mathcal{O}(n^2)$, si n est le nombre d'états de l'automate $\mathcal{A}$. En effet, en implémentant l'algorithme de manière soigneuse, il suffit de passer en revue les n états de $\mathcal{A}$ au plus n fois. A la première étape, on tente de distinguer les états de $\mathcal{A}$ en utilisant le mot vide. A la deuxième étape, on passe à nouveau en revue les états que l'on tente de distinguer au moyen de mots de longueur 1 et on répète l'opération jusqu'aux mots de longueur $n-1$.

Il est possible[7] d'obtenir un algorithme de complexité temporelle en $\mathcal{O}(n \log n)$ en considérant quelques raffinements à propos de la relation d'équivalence $\sim_{\mathcal{A}}$. Ces raffinements sortent du cadre introductif de ce cours.

Remarque IV.5.11. La procédure de minimisation donnée à la proposition IV.4.4 peut s'avérer coûteuse car elle demande deux procédures de

[7] cf. J.E. Hopcroft, An $n \log n$ algorithm for minimizing states in a finite automaton, *Theory of Machines and Computations*, Academic Press, New-York, 189–196, (1971).

déterminisation et par conséquent, le nombre d'états peut subir une croissance doublement exponentielle dans le cas le moins favorable. Notons que si $L = L^R$, alors, dans la procédure donnée à la proposition IV.4.4, l'automate minimal de L est simplement $\mu(\mathcal{A})$ si $\mathcal{A}$ est un AFD accessible acceptant L.

Ainsi, en résumé, pour une expression régulière donnée, on construira tout d'abord un automate fini non déterministe acceptant le langage généré par l'expression. Cet AFND contiendra en général des ε-transitions. Rappelons une fois encore que le non déterminisme est un outil puissant permettant d'exprimer facilement des langages aux spécifications complexes. Ensuite, on rendra cet automate déterministe (avec le risque inévitable d'une explosion exponentielle du nombre d'états). La procédure de déterminisation fournit toujours un AFD accessible. Il ne reste plus alors qu'à réduire l'AFD en détectant les ensembles d'états équivalents.

6. Exercices

Exercice IV.6.1. L'automate de la figure IV.14 est-il accessible et réduit? Autrement dit, s'agit-il d'un automate minimal ? Même question

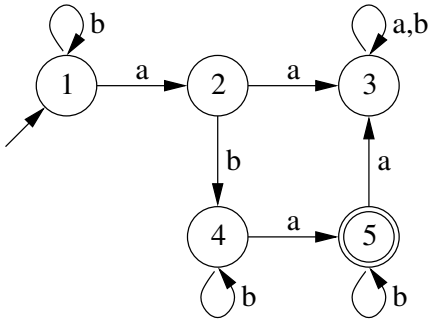


FIGURE IV.14. Un AFD.

avec l'automate de la figure IV.15. Pour ces deux automates, donner une expression régulière du langage accepté.

Exercice IV.6.2. Donner (en utilisant une méthode au choix) l'automate minimal des langages suivants :

- $a^*ba(bb)^*$,
- $(a+b)^*aba(a+b)^*$,
- $(ab+ba)^*$,
- le langage formé des mots contenant le facteur aa ou bb,
- le langage formé des mots contenant le facteur aa et bb,
- $(aab)^*(ba)^*$,
- le langage formé des mots de $(aab)^*(ba)^*$ qui sont de longueur paire.

Exercice IV.6.3. Soit $L = (ab+bab)^*$. Quels sont les différents ensembles de la forme

$$w^{-1}.L, \quad w \in \{a,b\}^*.$$

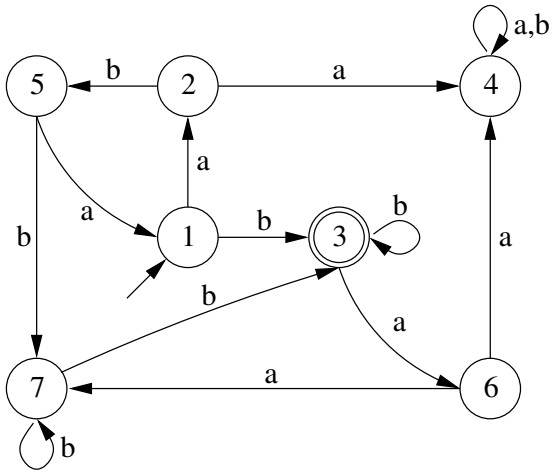


FIGURE IV.15. Un autre AFD.

En déduire l'automate minimal de L.

Exercice IV.6.4. Montrer que $\sim_L$ n'est en général pas une congruence à gauche, i.e., il existe $z \in \Sigma^*$ tel que $x \sim_L y$ et $zx \not\sim_L zy$.

Exercice IV.6.5. Soit $L = \{ab, aab, aba, ba, bb, aaa\}$. Quels sont les différents ensembles de la forme

$$w^{-1}.L, \quad w \in \{a, b\}^*.$$

En déduire l'automate minimal de L.

Exercice IV.6.6. Soit $L = (a + b)^* abaaba$. Quels sont les différents ensembles de la forme

$$w^{-1}.L, \quad w \in \{a, b\}^*.$$

En déduire l'automate minimal de L.

Exercice IV.6.7. Soit L, le langage sur $\{a, b\}$ des mots contenant exactement deux a. Quels sont les différents ensembles de la forme

$$w^{-1}.L, \quad w \in \{a, b\}^*.$$

En déduire l'automate minimal de L.

Exercice IV.6.8. Soit l'automate déterministe $\mathcal{A}$ représenté à la figure IV.16. Rechercher l'automate minimal du langage accepté par $\mathcal{A}$. On procédera par deux méthodes : la recherche des états équivalents et la procédure "$\mu(\mu(\mathcal{A}))$".

Exercice IV.6.9. Soit le langage

$$L = \{a^n\, b^m \mid n, m \in \mathbb{N} : n \leq m\}.$$

Caractériser les états de l'automate minimal de L et donner la table de transition de cet automate.

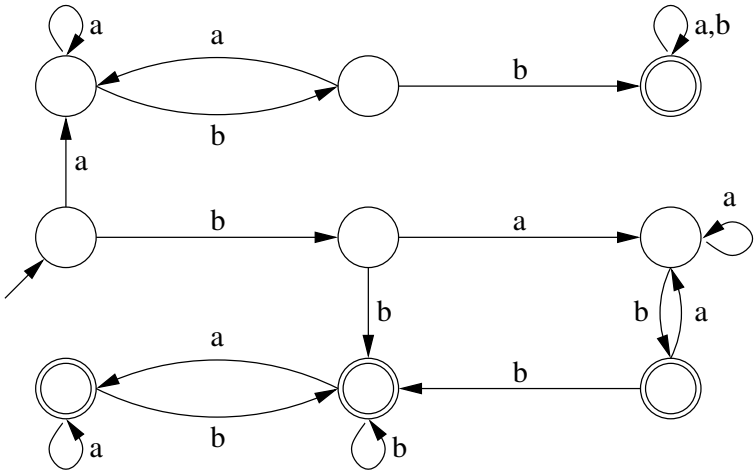


FIGURE IV.16. Un autre AFD dont on cherche le minimal.

Exercice IV.6.10. Soit l'automate fini déterministe $\mathcal{A}$ représenté à la figure IV.17. Rechercher les états équivalents pour la relation $\sim_{\mathcal{A}}$. En déduire

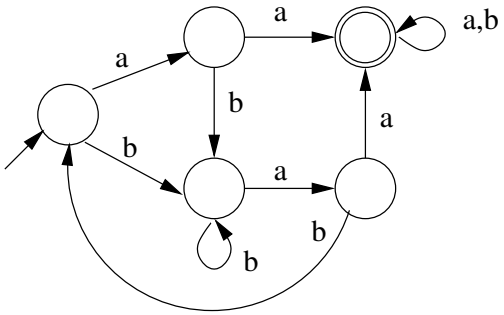


FIGURE IV.17. Recherche des états équivalents.

l'automate minimal du langage accepté par $\mathcal{A}$.

Exercice IV.6.11. On considère l'alphabet $\Sigma = \{a, b, c\}$.

- Donner l'automate minimal du langage $L = a^*b^*c^*$ (dans votre réponse, justifier en quoi l'automate que vous proposez est minimal).
- Quelles sont les classes d'équivalence de Σ^* pour la relation de Nerode $\sim_L$ et quels sont les différents ensembles de la forme $w^{-1}.L$, $w \in \Sigma^*$?
- La clôture commutative de L donnée par

$$\text{com}(L) = \{w \in \Sigma^* \mid \exists v \in L, \forall \sigma \in \Sigma : |w|_\sigma = |v|_\sigma\}$$

est-elle un langage régulier ? Justifier.

CHAPITRE V

Quelques compléments sur les langages réguliers

1. Transduction

Dans cette section, on définit la notion de transducteur qui, d'une certaine manière, peut être vue comme une généralisation des morphismes. Ensuite, nous montrons que l'ensemble des langages réguliers est stable pour l'image et l'image inverse par transduction.

Définition V.1.1. Un *transducteur* est un 6-uple $\mathcal{T} = (Q, q_0, \Sigma, \delta, \Delta, \tau)$ où Q, q_0, Σ, δ sont définis comme dans le cas des AFD, Δ est un alphabet fini appelé *alphabet de sortie* et $\tau : Q \times \Sigma \to \Delta^*$ est la *fonction de sortie*. (On supposera que τ est une fonction totale.) Un transducteur peut être vu comme un moyen pour définir des fonctions. Ainsi, à chaque *mot d'entrée* $w = w_1 \cdots w_\ell \in \Sigma^*$, $w_i \in \Sigma$, le transducteur $\mathcal{T}$ associe un *mot de sortie* $\mathcal{T}(w) \in \Delta^*$ donné par

$$\tau(q_0, w_1)\, \tau(\delta(q_0, w_1), w_2)\, \tau(\delta(q_0, w_1 w_2), w_3) \cdots \tau(\delta(q_0, w_1 \cdots w_{\ell-1}), w_\ell).$$

La représentation sagitale d'un transducteur se fait de la façon suivante. Pour tous $q, q' \in Q$, $\sigma \in \Sigma$, si $\delta(q, \sigma) = q'$ et $\tau(q, \sigma) = u \in \Delta^*$, alors on note

$$q \xrightarrow{\sigma/u} q'.$$

Exemple V.1.2. Voici un exemple de transducteur. Ici, $\Sigma = \{a, b\}$,

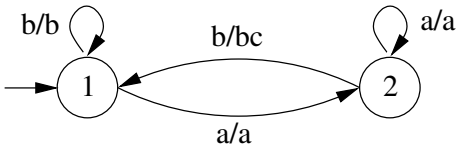


FIGURE V.1. Un transducteur.

l'alphabet de sortie est $\Delta = \{a, b, c\}$ et la fonction de sortie est définie par $\tau(1, a) = a$, $\tau(1, b) = b$, $\tau(2, a) = a$ et $\tau(2, b) = bc$. Considérant le mot $w = bab$, on a

$$1 \xrightarrow{b/b} 1 \xrightarrow{a/a} 2 \xrightarrow{b/bc} 1$$

et donc $\mathcal{T}(w) = babc$. Il est facile de voir que ce transducteur insère un c après chaque occurrence de ab dans le mot d'entrée.

Remarque V.1.3. La fonction sur Σ^* et à valeurs dans Δ^*, définie par le transducteur $\mathcal{T}$, est souvent appelée *fonction rationnelle*.

Exemple V.1.4. Si $f : \Sigma^* \to \Delta^*$ est un morphisme de monoïdes, cette fonction peut être aisément réalisée par un transducteur possédant un unique état. En effet, il suffit de considérer le transducteur représenté à la figure V.2.

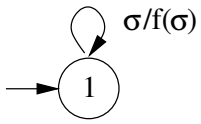


FIGURE V.2. Un transducteur calculant le morphisme f.

Remarque V.1.5. Dans la littérature, on trouve d'autres modèles plus généraux de transducteurs, comme par exemple, des transducteurs construits non pas sur un AFD mais sur un AFND. Dans ce cas, le transducteur ne définit plus une fonction de Σ^* dans Δ^* mais une relation (rationnelle), i.e., une partie de $\Sigma^* \times \Delta^*$. On peut aussi trouver des modèles dans lesquels on précise des états finals. Dans ce dernier cas, ne sont acceptés que les calculs dont la lecture du mot d'entrée conduit à un état final. Dans ce cours introductif, nous avons décidé de passer ces généralisations sous silence.

L'ensemble des langages réguliers est stable par transduction.

Théorème V.1.6. *Soient $L \subset \Sigma^*$ un langage régulier et $\mathcal{T}$ un transducteur. Le langage*

$$\mathcal{T}(L) = \{\mathcal{T}(w) \mid w \in L\} \subseteq \Delta^*$$

est régulier.

Démonstration. [1] Soient $\mathcal{A} = (Q, q_0, F, \Sigma, \delta)$ un AFD acceptant L et $\mathcal{T} = (Q', q_0', \Sigma, \delta', \Delta, \tau)$ le transducteur donné dans l'énoncé. Nous allons construire un AFND $\mathcal{B} = (Q'', q_0'', F'', \Delta, \delta'')$ acceptant exactement $\mathcal{T}(L)$. On le définit comme suit :

- $Q'' = Q \times Q' \times (\Sigma \cup \{\varepsilon\}) \times \{0, 1, \ldots, k\}$ où $k = \max_{\sigma \in \Sigma, q' \in Q'} |\tau(q', \sigma)|$,
- $q_0'' = (q_0, q_0', \varepsilon, 0)$,
- la relation de transition $\delta'' \subset Q'' \times \Delta^* \times Q''$ contient les éléments suivants. Pout tout $\sigma \in \Sigma$,

$$((q, q', \varepsilon, 0), \varepsilon, (q, q', \sigma, 0)) \in \delta''.$$

Si $\tau(q', \sigma) = y_1 \cdots y_j$, alors pour tout i tel que $1 \leq i \leq j$

$$((q, q', \sigma, i-1), y_i, (q, q', \sigma, i)) \in \delta''$$

et

$$((q, q', \sigma, j), \varepsilon, (\delta(q, \sigma), \delta'(q', \sigma), \varepsilon, 0)) \in \delta''.$$

[1] La preuve présentée ici est issue de : J.-P. Allouche, J. Shallit, *Automatic Sequences, Theory, Applications, Generalizations*, Cambridge University Press (2003).

En particulier, si $\tau(q', \sigma) = \varepsilon$ alors

$$((q, q', \sigma, 0), \varepsilon, (\delta(q, \sigma), \delta'(q', \sigma), \varepsilon, 0)) \in \delta''.$$

- Enfin, $F = \{(q, q', \varepsilon, 0) : q \in F\}$.

L'idée à la base de cette définition est la suivante : si $w \in \mathcal{T}(L)$, il existe $x \in L$ tel que $w = \mathcal{T}(x)$. Supposons que $x = x_1 \cdots x_r$, $x_t \in \Sigma$, et que $w_t \in \Delta^*$ soit la sortie correspondant à la lecture de x_t dans $\mathcal{T}$. En particulier, on a $w = w_1 \cdots w_r$. L'automate $\mathcal{B}$ peut deviner de manière non déterministe s'il existe un mot $x \in L$ produisant le mot w. En effet, la première composante de Q'' permet de simuler le comportement de $\mathcal{A}$. La deuxième composante simule le comportement de $\mathcal{T}$. La troisième composante de Q'' est utilisée pour mémoriser la lettre $\sigma = x_t$ du mot x qui vient d'être supposée. La quatrième composante permet de savoir combien de lettres de w_t ont déjà été rencontrées. Cette dernière composante sert de compteur, initialisé à zéro et incrémenté d'une unité à chaque fois qu'une lettre de w_t est lue. Lorsque ce compteur atteint $|w_t|$, on utilise une ε-transition pour réinitialiser la troisième composante à ε et la quatrième à 0. De plus, pour simuler le comportement de $\mathcal{A}$ et $\mathcal{T}$, la première composante passe à $\delta(q, \sigma)$ et la deuxième à $\delta'(q', \sigma)$. Il nous suffit de montrer que

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

Soit $w \in \mathcal{T}(L)$. Il existe $x = x_1 \cdots x_r \in L$ tel que $\mathcal{T}(x) = w$. Comme précédemment, w_t est la sortie correspondant à x_t et on note $k_t = |w_t|$. L'exécution de x dans $\mathcal{A}$ donne la suite d'états

$$q_0 = p_0, p_1, \ldots, p_r$$

où $p_r \in F$ car $x \in L$. De façon semblable, la lecture de x dans $\mathcal{T}$ conduit à la suite

$$q'_0 = p'_0 \xrightarrow{x_1/w_1} p'_1 \xrightarrow{x_2/w_2} p'_2 \longrightarrow \cdots \xrightarrow{x_r/w_r} p'_r.$$

On note $w_t = w_{t1} \cdots w_{tk_t}$. Ainsi, dans $\mathcal{B}$, la lecture de w peut conduire à la suite d'états

$$\begin{aligned}
&\underbrace{(p_0, p'_0, \varepsilon, 0)}_{=q''_0} \xrightarrow{\varepsilon} (p_0, p'_0, x_1, 0) \xrightarrow{w_{11}} (p_0, p'_0, x_1, 1) \longrightarrow \cdots \xrightarrow{w_{1k_1}} (p_0, p'_0, x_1, k_1) \\
&\xrightarrow{\varepsilon} \underbrace{(\delta(p_0, x_1), \delta'(p'_0, x_1), \varepsilon, 0)}_{=(p_1, p'_1, \varepsilon, 0)} \xrightarrow{\varepsilon} (p_1, p'_1, x_2, 0) \xrightarrow{w_{21}} \cdots \xrightarrow{w_{2k_2}} (p_1, p'_1, x_2, k_2) \\
&\xrightarrow{\varepsilon} (p_2, p'_2, \varepsilon, 0) \xrightarrow{\varepsilon} (p_2, p'_2, x_3, 0) \longrightarrow \cdots \\
&\vdots \\
&\xrightarrow{\varepsilon} (p_{r-1}, p'_{r-1}, \varepsilon, 0) \xrightarrow{\varepsilon} (p_{r-1}, p'_{r-1}, x_r, 0) \xrightarrow{w_{r1}} \cdots \xrightarrow{w_{rk_r}} (p_{r-1}, p'_{r-1}, x_r, k_r) \\
&\xrightarrow{\varepsilon} (p_r, p'_r, \varepsilon, 0) \in F''.
\end{aligned}$$

Ceci prouve que le mot $w = w_{11} \cdots w_{1k_1} w_{21} \cdots w_{2k_2} \cdots w_{r1} \cdots w_{rk_r}$ est accepté par $\mathcal{B}$. Pour l'autre inclusion, si $w \in L(\mathcal{B})$, alors cela signifie que partant de l'état initial q''_0, on dispose d'un chemin conduisant à un état

final de F''. Ainsi, par définition de $\mathcal{B}$, on retrouve un mot $x \in L$ tel que $\mathcal{T}(x) = w$ et par conséquent, w appartient bien à $\mathcal{T}(L)$.

■

Remarque V.1.7. Au vu de l'exemple V.1.4 et du théorème précédent, on retrouve comme cas particulier, le fait que l'ensemble des langages réguliers est stable par morphisme (cf. proposition II.3.4).

L'ensemble des langages réguliers est aussi stable par image inverse par transduction.

Proposition V.1.8. *Soient $L \subset \Delta^*$ un langage régulier et $\mathcal{T}$ un transducteur. Le langage*

$$\mathcal{T}^{-1}(L) = \{x \in \Sigma^* \mid \mathcal{T}(x) \in L\}$$

est régulier.

Démonstration. Il est aisé de construire un AFD acceptant $\mathcal{T}^{-1}(L)$ à partir d'un AFD $\mathcal{A} = (Q, q_0, F, \Delta, \delta)$ acceptant L et du transducteur $\mathcal{T} = (Q', q_0', \Sigma, \delta', \Delta, \tau)$ donné dans l'énoncé. Soit l'AFD $\mathcal{B} = (Q'', q_0'', F'', \Sigma, \delta'')$ défini par

- $Q'' = Q' \times Q$,
- $q_0'' = (q_0', q_0)$,
- $F'' = Q' \times F$ et
- pour tout $\sigma \in \Sigma$, $\delta''((q', q), \sigma) = (\delta'(q', \sigma), \delta(q, \tau(q', \sigma)))$.

La première composante simule le transducteur $\mathcal{T}$ et la seconde composante simule l'automate $\mathcal{A}$ sur la sortie produite par $\mathcal{T}$. Ainsi, il est clair que $L(\mathcal{B}) = \mathcal{T}^{-1}(L)$.

■

2. Recherche d'un mot dans un texte

Une application pratique des automates concerne la recherche d'un mot dans un texte. En effet, les traitements de textes que l'on peut trouver sur n'importe quelle plate-forme utilisent de manière interne des algorithmes basés sur la construction d'automates pour implémenter les fonctions bien utiles de recherche ("find", "find and replace", etc...). A titre indicatif[2], `Tcl`, `Perl`, `Python`, `GNU Emacs`, `ed`, `sed`, `vi`, la plupart des versions de `grep` et certaines versions de `egrep` et `awk` utilisent des AFND. Par contre, la majorité des versions de `egrep` et `awk`, `lex` et `flex` utilisent quant à eux des AFD.

Notre but est ici de rechercher les occurrences d'un mot u dans un texte T écrit sur l'alphabet Σ (un texte étant une suite finie de symboles de Σ,

[2]Pour plus de détails, voir par exemple, J.E.F. Friedl, *Mastering Regular Expressions*, O'Reilly.

il s'agit simplement d'un mot sur Σ). Ainsi, nous recherchons un AFD acceptant le langage

$$L = \Sigma^* u.$$

Pour ce faire, nous allons décrire l'automate minimal de ce langage. Les états sont de la forme $w^{-1}.L$ avec $w \in \Sigma^*$. Ainsi,

$$v \in w^{-1}.L \Leftrightarrow wv \in \Sigma^* u.$$

Pour décrire les ensembles $w^{-1}.L$, il est utile d'introduire, pour tout préfixe p de u, l'ensemble

$$\boxed{E_u(p) = \{\gamma \in \Sigma^* \mid \exists \alpha', \beta \in \Sigma^* : p = \beta\alpha', u = \alpha'\gamma\}}$$

formé des suffixes de u qui, complétés par un suffixe de p, donnent u. On remarque qu'avec cette définition, u appartient toujours à $E_u(p)$.

Soit v appartenant à $w^{-1}.L$. Si $|v| \geq |u|$, alors v appartient à L car v possède u comme suffixe. On en conclut donc que $w^{-1}.L \supseteq L$.

Sinon, $|v| < |u|$. Dans ce cas, on pose $\alpha_{w,u}$ comme étant le plus grand suffixe de w qui soit préfixe de u. Il est clair que $\alpha_{w,u}$ et $E_u(\alpha_{w,u})$ dépendent uniquement de u et w.

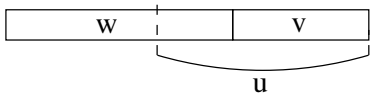


FIGURE V.3. wv appartient à $\Sigma^* u$.

Exemple V.2.1. Avec les notations précédentes, si

$$w = aabbab \quad \text{et} \quad u = babbaab,$$

alors

$$\underbrace{aabbab}_{w} baab \quad \text{et} \quad \underbrace{aabbab}_{w} abbaab$$

appartiennent à $L = \Sigma^* u$. Ici, $\alpha_{w,u} = bab$ car

$$\begin{array}{rc|c|c} u = & & bab & baab \\ w = & aab & bab & \end{array}.$$

De plus, on a

$$E_u(\alpha_{w,u}) = \{baab, abbaab, u\}$$

En effet, les suffixes de $\alpha_{w,u}$ sont ε, b, ab, bab. Parmi eux, bab et b sont préfixes de u et on a les factorisations suivantes,

$$u = \underbrace{\overbrace{bab}^{\alpha'}}_{\alpha_{w,u}} baab \quad \text{et} \quad u = \underbrace{\overbrace{ba}^{\beta}\overbrace{b}^{\alpha'}}_{\alpha_{w,u}} abbaab.$$

Ainsi, on se convainc aisément que

$$w^{-1}.L = L \cup E_u(\alpha_{w,u}).$$

Si $u = u_1 \cdots u_\ell$, les préfixes de u sont

$$p_0 = \varepsilon,\ p_1 = u_1,\ \ldots,\ p_\ell = u_1 \cdots u_\ell.$$

Les états de l'automate minimal de L sont donc les

$$L \cup E_u(p_i),\ i \in \{0, \ldots, \ell\}.$$

Au vu de ce qui précède, il est clair que

$$L \cup E_u(p_i) = p_i^{-1}.L.$$

Si on se rappelle la définition de l'automate minimal d'un langage, on retrouve les caractéristiques de celui-ci.

- L'état initial est tel que
$$p_i^{-1}.L = L,$$
et donc $i = 0$. En effet, si $0 < i \leq \ell$, $p_i^{-1}.L$ contient au moins un mot de longueur strictement inférieure à $|u|$, alors que L ne contient que des mots de longueur au moins $|u|$.
- Un état est final si et seulement si $\varepsilon \in p_i^{-1}.L$. Donc, le seul état final est $p_\ell^{-1}.L$.
- Recherchons la fonction de transition de l'automate. Si $\sigma \in \Sigma$, alors par définition de δ_L, on a
$$\delta_L(p_i^{-1}.L, \sigma) = (p_i\sigma)^{-1}.L.$$
De plus, si $\sigma = u_{i+1}$, alors $p_i\sigma = p_{i+1}$. Sinon, $\sigma \neq u_{i+1}$ et
$$(p_i\sigma)^{-1}.L = p_j^{-1}.L$$
où p_j est le plus grand préfixe de u qui soit suffixe de $p_i\sigma$. (Définition somme toute assez naturelle au vu de la défintion des ensembles $E_u(p)$.)

Ainsi, pour un mot u donné, il est facile de construire la table de l'automate. Nous convenons de noter i l'état correspondant à $p_i^{-1}.L$.

Exemple V.2.2. Soit $u = abbab$. On a

i	p_i	$\delta(i,a)$	$\delta(i,b)$
0	ε	1 car $\varepsilon a = p_1$	0 car p_0 suffixe de b
1	a	1 car p_1 suffixe de aa	2 car $ab = p_2$
2	ab	1 car p_1 suffixe de aba	3 car $abb = p_3$
3	abb	4 car $abba = p_4$	0 car p_0 suffixe de $abbb$
4	$abba$	1 car p_1 suffixe de $abbaa$	5 car $abbab = p_5$
5	$abbab$	1 car p_1 suffixe de $abbaba$	3 car p_3 suffixe de $abbabb$

et on trouve l'automate représenté à la figure V.4. Si on doit écrire un programme détectant la première occurrence de $abbab$ dans un texte fourni en entrée, il suffit de décréter que la procédure s'arrête une fois l'état 5

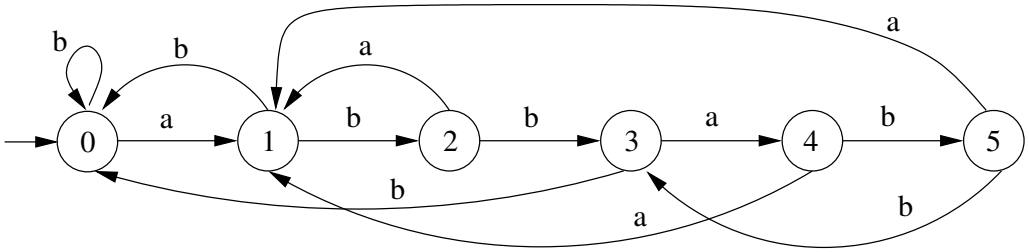


FIGURE V.4. Un automate détectant *abbab*.

atteint. Si on devait compter le nombre d'occurrence du facteur *abbab* dans un texte donné, on pourrait décider d'incrémenter un compteur d'une unité à chaque fois que l'état 5 serait atteint.

Remarque V.2.3. La construction de la table de transition de l'automate s'effectue en un temps proportionnel à $|u|$. En effet, le nombre d'états est $|u|+1$ et pour chaque état et chaque lettre de l'alphabet, une seule opération de comparaison de mots est nécessaire pour déterminer l'état atteint. Une fois la table de transition construite, la recherche d'un mot dans un texte T prend un temps proportionnel à $|T|$ puisque le texte T est lu lettre par lettre dans l'automate.

Exemple V.2.4. En appliquant la construction détaillée dans cette section, on peut construire aisément un automate reconnaissant la séquence génétique "agata". Cet automate est représenté à la figure V.5. De même,

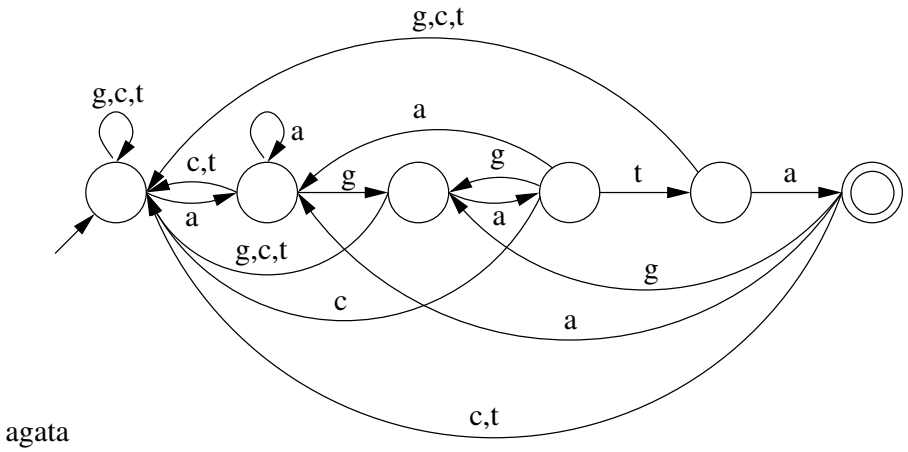


FIGURE V.5. Un automate détectant "agata".

pour rechercher le mot "ananas" dans un texte, on a l'automate de la figure V.6. Sur cette dernière, toutes les transitions non représentées aboutissent à l'état initial, l'alphabet étant $\{a, b, \ldots, z\}$.

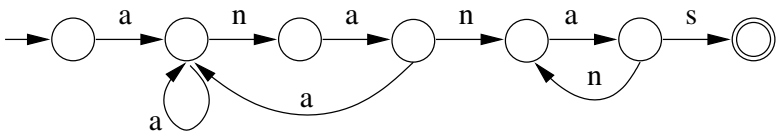


FIGURE V.6. Un automate détectant "ananas".

3. Fonction de complexité d'un langage régulier

Définition V.3.1. Soit $L \subseteq \Sigma^*$. La *fonction de complexité* du langage L est la fonction

$$\rho_L : \mathbb{N} \to \mathbb{N} : n \mapsto \#(L \cap \Sigma^n).$$

Cette fonction associe donc à n le nombre de mots de longueur n dans le langage L.

Le but de cette section est d'étudier la fonction de complexité d'un langage régulier. Le résultat principal est que la suite $(\rho_L(n))_{n\in\mathbb{N}}$ satisfait une relation de récurrence linéaire à coefficients constants.

Soit $L \subseteq \Sigma^*$ un langage régulier accepté par un AFD $\mathcal{A} = (Q, q_0, F, \Sigma, \delta)$. Il est clair que $\rho_L(n)$ est le nombre de chemins de longueur n débutant dans q_0 et se terminant dans un état final de F. Le problème posé se ramène donc à un problème de dénombrement de chemins dans un graphe.

Définition V.3.2. Soit $\mathcal{A} = (Q, q_0, F, \Sigma, \delta)$ un AFD. La *matrice d'adjacence* de $\mathcal{A}$ est la matrice donnée par

$$M_{q,r} = \#\{\sigma \in \Sigma \mid \delta(q, \sigma) = r\}, \quad q, r \in Q.$$

Exemple V.3.3. Considérons l'automate minimal du langage sur $\{a, b\}$ formé des mots ne contenant pas deux a consécutifs.

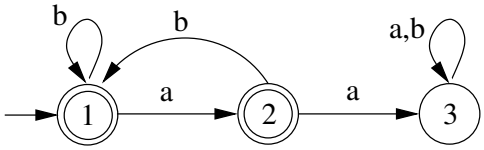


FIGURE V.7. AFD acceptant les mots ne contenant pas aa.

La matrice d'adjacence de $\mathcal{A}$ est

$$\begin{pmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 0 & 2 \end{pmatrix}.$$

Proposition V.3.4. *Soient $\mathcal{A} = (Q, q_0, F, \Sigma, \delta)$ un AFD et M sa matrice d'adjacence. Pour tous $q, r \in Q$ et tout $n \in \mathbb{N}$,*

$$[M^n]_{q,r}$$

est le nombre de chemins de longueur n joignant q à r.

Démonstration. On procède par récurrence sur n. Si $n = 0$ ou $n = 1$, le résultat est évident. Supposons la propriété satisfaite pour n et vérifions-la pour $n + 1$. Il vient

$$[M^{n+1}]_{q,r} = [M^n.M]_{q,r} = \sum_{s \in Q} [M^n]_{q,s} M_{s,r}.$$

Par hypothèse de récurrence, $[M^n]_{q,s}$ compte le nombre de chemins de longueur n joignant q à s. Or, il est clair que le nombre de chemins de longueur $n + 1$ joignant q à r s'obtient à partir des chemins de longueur n joignant q et s et des chemins de longueur 1 joignant s à r.

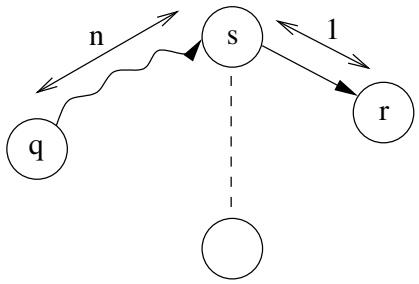


FIGURE V.8. Chemins de longueur $n + 1$ joignant q à r.

■

Corollaire V.3.5. *Soient $\mathcal{A} = (Q, q_0, F, \Sigma, \delta)$ un AFD acceptant L et M sa matrice d'adjacence. On a*

$$\rho_L(n) = \sum_{f \in F} [M^n]_{q_0,f}.$$

Démonstration. C'est évident.

■

Exemple V.3.6. Poursuivons l'exemple V.3.3. Les premières puissances de la matrice d'adjacence de $\mathcal{A}$ sont

$$M^2 = \begin{pmatrix} 2 & 1 & 1 \\ 1 & 1 & 2 \\ 0 & 0 & 4 \end{pmatrix}, \; M^3 = \begin{pmatrix} 3 & 2 & 3 \\ 2 & 1 & 5 \\ 0 & 0 & 8 \end{pmatrix}, \; M^4 = \begin{pmatrix} 5 & 3 & 8 \\ 3 & 2 & 11 \\ 0 & 0 & 16 \end{pmatrix}, \ldots$$

Ainsi, on peut remarquer qu'il y a 2 chemins (resp. 1 chemin) de longueur 2 de l'état 1 vers 1 (resp. 2). En sommant les deux, il y a donc 3 chemins de longueur 2 appartenant au langage accepté par l'automate. Ou encore, on trouve 8 mots de longueur 4 dans ce langage.

Nous allons à présent fournir une méthode générale permettant de calculer $\rho_L(n)$. En vertu du théorème de Cayley-Hamilton, toute matrice annule son polynôme caractéristique[3] $\det(M - \lambda I)$. Si $\#Q = k$, la matrice

[3]On peut faire tout le raisonnement qui suit en considérant non pas le polynôme caractéristique de M, mais son polynôme minimum.

M est une matrice carrée de dimension k et $\det(M - \lambda I)$ est un polynôme monique à coefficients entiers de degré k en λ. Ainsi, il existe $c_1, \ldots, c_k \in \mathbb{Z}$ tels que

$$M^k = c_1 M^{k-1} + \cdots + c_k I.$$

En multipliant les deux membres de cette égalité par M^{n-k}, on trouve pour tout $n \geq k$,

$$M^n = c_1 M^{n-1} + \cdots + c_k M^{n-k}.$$

Ceci signifie que les coefficients de M^n satisfont une relation de récurrence linéaire à coefficients constants, i.e., pour tous $q, r \in Q$ et tout $n \geq k$,

$$[M^n]_{q,r} = c_1 [M^{n-1}]_{q,r} + \cdots + c_k [M^{n-k}]_{q,r}.$$

En conséquence du corollaire V.3.5, il vient

$$\boxed{\rho_L(n) = c_1\, \rho_L(n-1) + \cdots + c_k\, \rho_L(n-k), \quad \forall n \geq k.}$$

Le problème revient à réussir à exprimer $\rho_L(n)$ sous la forme d'une *formule close*. Cette question à propos des suites linéaires récurrentes est en toute généralité difficile à résoudre. On dispose du résultat suivant que nous donnons ici sans démonstration.

Proposition V.3.7. *Soit $k \geq 1$.Si une suite $(u_n)_{n\in\mathbb{N}}$ satisfait une relation de récurrence linéaire à coefficients constants de la forme*

$$u_n = c_1\, u_{n-1} + \cdots + c_k\, u_{n-k},\ \forall n \geq k$$

et si $\alpha_1, \ldots, \alpha_r$ sont les racines de multiplicité $m_1, \ldots, m_r$ du polynôme caractéristique de la récurrence

$$X^k - c_1\, X^{k-1} - \cdots - c_k,$$

alors il existe des polynômes P_i de degré strictement inférieur à m_i, $i \in \{1, \ldots, r\}$, tels que

$$u_n = P_1(n)\, \alpha_1^n + \cdots + P_r(n)\, \alpha_r^n.$$

En particulier, les polynômes $P_1, \ldots, P_r$ sont entièrement déterminés par les conditions initiales $u_0, \ldots, u_{k-1}$.

Ainsi, ce théorème nous montre que rechercher une forme close pour $\rho_L(n)$ revient à rechercher les racines d'un polynôme de degré k.

Exemple V.3.8. Poursuivons l'exemple V.3.3. Le polynôme caractéristique de la matrice d'adjacence est donné par

$$\det \begin{pmatrix} 1-\lambda & 1 & 0 \\ 1 & -\lambda & 1 \\ 0 & 0 & 2-\lambda \end{pmatrix} = (2-\lambda)(\lambda^2 - \lambda - 1).$$

Ainsi, puisque

$$(2-\lambda)(\lambda^2 - \lambda - 1) = -\lambda^3 + 3\lambda^2 - \lambda - 2,$$

en vertu du théorème de Cayley-Hamilton, on a

$$M^3 = 3M^2 - M - 2I$$

et donc pour tout $n \geq 3$,

$$M^n = 3M^{n-1} - M^{n-2} - 2M^{n-3}.$$

En conséquence du corollaire V.3.5, on a

$$\rho_L(n) = 3\,\rho_L(n-1) - \rho_L(n-2) - 2\,\rho_L(n-3), \quad \forall n \geq 3.$$

De plus,

$$\rho_L(0) = 1,\ \rho_L(1) = 2 \text{ et } \rho_L(2) = 3$$

car $\varepsilon, a, b, ab, ba, bb$ appartiennent à L. Pour déterminer une formule close pour $\rho_L(n)$, nous factorisons tout d'abord le polynôme caractéristique de la relation de récurrence,

$$X^3 - 3X^2 + X + 2 = (X-2)(X - \frac{1+\sqrt{5}}{2})(X - \frac{1-\sqrt{5}}{2}).$$

En vertu de la proposition V.3.7, puisque les trois racines du polynôme sont simples, il existe trois constantes A, B, C telles que

$$\rho_L(n) = A2^n + B\left(\frac{1+\sqrt{5}}{2}\right)^n + C\left(\frac{1-\sqrt{5}}{2}\right)^n,\ \forall n \geq 3.$$

Au vu des conditions initiales, on a le système suivant

$$\begin{cases} 1 = A + B + C \\ 2 = 2A + B\left(\frac{1+\sqrt{5}}{2}\right) + C\left(\frac{1-\sqrt{5}}{2}\right) \\ 3 = 4A + B\left(\frac{1+\sqrt{5}}{2}\right)^2 + C\left(\frac{1-\sqrt{5}}{2}\right)^2 \end{cases}$$

et on trouve

$$A = 0,\ B = \frac{5+3\sqrt{5}}{10} \text{ et } C = \frac{5-3\sqrt{5}}{10}.$$

Par conséquent,

$$\rho_L(n) = \frac{5+3\sqrt{5}}{10}\left(\frac{1+\sqrt{5}}{2}\right)^n + \frac{5-3\sqrt{5}}{10}\left(\frac{1-\sqrt{5}}{2}\right)^n. \tag{4}$$

Remarque V.3.9. La présence d'un puits, ou plus généralement d'un état non coaccessible (i.e., depuis lequel on ne peut atteindre aucun état final), n'a pas d'influence sur le nombre de mots de longueur n présents dans le langage. Ainsi, il est commode dans les exercices de considérer un automate "émondé" privé de tels états. On pourrait ainsi reprendre l'exercice précédent en ne considérant dans l'automate de la figure V.7 que les états 1 et 2.

Une autre méthode fort utile dans le cadre des équations linéaires récurrentes consiste à utiliser la notion de série génératrice. Ainsi, si $(u_n)_{n\in\mathbb{N}}$ est une suite, on note symboliquement

$$F_u(X) = \sum_{k\geq 0} u_k X^k$$

la *série génératrice* de cette suite. Il s'agit d'une manière commode de coder les éléments de $(u_n)_{n\in\mathbb{N}}$. On peut définir la somme et le produit de deux séries formelles pour munir l'ensemble des séries d'une structure d'anneau.

Proposition V.3.10. *Si $(u_n)_{n\in\mathbb{N}}$ satisfait l'équation linéaire récurrente homogène de degré k*

$$\forall n \geq 0, \quad u_{n+k} = \sum_{i=1}^{k} a_i \, u_{n+k-i}$$

avec comme conditions initiales $u_0 = b_0$, ..., $u_{k-1} = b_{k-1}$, alors la série génératrice F_u est la fraction rationnelle

$$F_u(X) = \frac{\sum_{i=0}^{k-1} b_i \, X^i - \sum_{i+j<k} a_i b_j \, X^{i+j}}{1 - \sum_{i=1}^{k} a_i \, X^i}$$

Démonstration. Il vient

$$\begin{aligned}
F_u(X) &= \sum_{n\geq 0} u_n \, X^n \\
&= \sum_{n\geq 0} u_{n+k} \, X^{n+k} + \sum_{i=0}^{k-1} u_i \, X^i \\
&= \sum_{n\geq 0} \left(\sum_{i=1}^{k} a_i \, u_{n+k-i} \right) X^{n+k} + \sum_{i=0}^{k-1} b_i \, X^i \\
&= \sum_{i=1}^{k} a_i \, X^i \sum_{n\geq 0} u_{n+k-i} \, X^{n+k-i} + \sum_{i=0}^{k-1} b_i \, X^i \\
&= \sum_{i=1}^{k} a_i \, X^i \left(F_u(X) - \sum_{j=0}^{k-i-1} u_j \, X^j \right) + \sum_{i=0}^{k-1} b_i \, X^i
\end{aligned}$$

On a utilisé ci-dessus le fait qu'il s'agit de sommations formelles et qu'il n'y a donc aucune objection à permuter les différents symboles sommatoires. Par conséquent, on obtient

$$\left(1 - \sum_{i=1}^{k} a_i \, X^i \right) F_u(X) = \sum_{i=0}^{k-1} b_i \, X^i - \sum_{i=1}^{k} \sum_{j=0}^{k-i-1} a_i u_j \, X^{i+j}$$

d'où la conclusion car

$$\sum_{i=1}^{k} \sum_{j=0}^{k-i-1} a_i u_j \, X^{i+j} = \sum_{i+j<k} a_i b_j \, X^{i+j}.$$

■

Pour obtenir une expression de $(u_n)_{n\in\mathbb{N}}$, il suffit de décomposer F_u en fractions simples puis de développer celles-ci en série de puissances. Une fois cela fait, il ne reste plus qu'à identifier les coefficients correspondants.

Nous allons illustrer cette technique sur un exemple.

Exemple V.3.11. Résolvons l'exemple V.3.3 en utilisant les séries génératrices. Nous savons déjà que

$$\rho_L(n) = 3\,\rho_L(n-1) - \rho_L(n-2) - 2\,\rho_L(n-3), \quad \forall n \geq 3.$$

Considérons la série génératrice

$$F(X) = \sum_{n\geq 0} \rho_L(n)\, X^n.$$

On a

$$\begin{aligned} F(X) &= \sum_{n\geq 3} \rho_L(n)\, X^n + \rho_L(2)\, X^2 + \rho_L(1)\, X^1 + \rho_L(0)\, X^0 \\ &= \sum_{n\geq 3} [3\,\rho_L(n-1) - \rho_L(n-2) - 2\,\rho_L(n-3)]\, X^n + 3X^2 + 2X + 1 \\ &= 3X\,[F(X) - \rho_L(0) - \rho_L(1)X] - X^2\,[F(X) - \rho_L(0)] - 2X^3\,F(X) \\ &\quad +3X^2 + 2X + 1 \\ &= (3X - X^2 - 2X^3)F(X) - 2X^2 - X + 1 \end{aligned}$$

et donc

$$F(X) = -\frac{2X^2 + X - 1}{2X^3 + X^2 - 3X + 1} = -\frac{(2X-1)(X+1)}{(2X-1)(X+\frac{1+\sqrt{5}}{2})(X+\frac{1-\sqrt{5}}{2})}.$$

Si on développe $F(X)$ en fraction rationnelles, on obtient

$$F(X) = \frac{\alpha}{X + \frac{1+\sqrt{5}}{2}} + \frac{\beta}{X + \frac{1-\sqrt{5}}{2}}$$

et

$$\begin{cases} \alpha + \beta = -1 \\ \alpha(1-\sqrt{5}) + \beta(1+\sqrt{5}) = -2. \end{cases}$$

De là, on tire

$$\alpha = -\frac{5-\sqrt{5}}{10}, \quad \beta = -\frac{5+\sqrt{5}}{10}.$$

Pour le développement en série de puissances, il est utile de rappeler les relations suivantes

$$\boxed{\frac{1}{1-\gamma\,X} = \sum_{k\geq 0} \gamma^k\, X^k.}$$

et

$$\boxed{\frac{1}{(1-\gamma\,X)^2} = \frac{1}{\gamma} D_x \frac{1}{1-\gamma\,X} = \sum_{k\geq 1} k\,\gamma^{k-1}\, X^{k-1}}$$

où D_x est une dérivation formelle[4]. D'une manière générale, $\frac{1}{(1-\gamma\,X)^p}$ est proportionnel à $D_x^{p-1} \frac{1}{1-\gamma\,X}$ et il suffit donc de dériver formellement $\sum_{k\geq 0} \gamma^k\, X^k$.

[4]Les détails et les justifications sortent du cadre de ce cours.

Ainsi, sur notre exemple, il ne reste plus qu'à développer $F(X)$ en série de puissances en utilisant la première relation pour obtenir

$$\frac{\alpha}{X+\frac{1+\sqrt{5}}{2}} = \alpha\frac{2}{1+\sqrt{5}}\,\frac{1}{1+\frac{2}{1+\sqrt{5}}X} = \alpha\frac{2}{1+\sqrt{5}}\sum_{k\geq 0}(-\frac{2}{1+\sqrt{5}})^k X^k,$$

et

$$\frac{\beta}{X+\frac{1-\sqrt{5}}{2}} = \beta\frac{2}{1-\sqrt{5}}\,\frac{1}{1+\frac{2}{1-\sqrt{5}}X} = \beta\frac{2}{1-\sqrt{5}}\sum_{k\geq 0}(-\frac{2}{1-\sqrt{5}})^k X^k.$$

En identifiant les coefficients, on trouve la formule close recherchée :

$$\rho_L(n) = \underbrace{-\frac{5-\sqrt{5}}{10}\frac{2}{1+\sqrt{5}}}_{\frac{5-3\sqrt{5}}{10}}\underbrace{(-\frac{2}{1+\sqrt{5}})}_{\frac{1-\sqrt{5}}{2}}{}^n \underbrace{-\frac{5+\sqrt{5}}{10}\frac{2}{1-\sqrt{5}}}_{\frac{5+3\sqrt{5}}{10}}\underbrace{(\frac{2}{\sqrt{5}-1})}_{\frac{1+\sqrt{5}}{2}}{}^n.$$

On retrouve bien la solution donnée en (4) à des réécritures algébriques près.

Pour conclure cette section, on dispose encore du résultat suivant.

Proposition V.3.12. *Soit L un langage régulier accepté par un AFD accessible $\mathcal{A}$. Le polynôme minimum de la matrice d'adjacence de $\mathcal{A}$ est divisible par le polynôme minimum de la matrice d'adjacence de l'automate minimal $\mathcal{A}_L$ de L.*

Démonstration. Soit M (resp. N) la matrice d'adjacence de l'AFD accessible $\mathcal{A} = (Q, q_0, F, \Sigma, \delta)$ (resp. de $\mathcal{A}_L = (Q_L, q_{0,L}, F_L, \Sigma, \delta_L)$). Au vu du théorème IV.3.8, pour tous $p, q \in Q_L$ et tout mot w de longueur k tel que $\delta_L(p, w) = q$, il existe une application $\Phi : Q \to Q_L$ et des états p' et q' de Q tels que

$$\Phi(p') = p,\ \Phi(q') = q \text{ et } \delta_L(\Phi(p'), w) = \Phi(\delta(p', w)) = \Phi(q') = q.$$

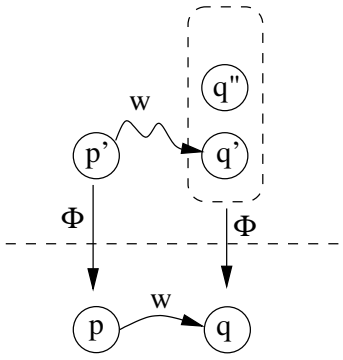


FIGURE V.9. Projection $\Phi : Q \to Q_L$ sur $\mathcal{A}_L$.

Ainsi, il est clair que

$$[N^k]_{p,q} = \sum_{q' \in \Phi^{-1}(q)} [M^k]_{p',q'}.$$

En effet, puisque $\mathcal{A}$ est déterministe, on ne doit prendre en compte qu'un seul $p' \in \Phi^{-1}(p)$ car sinon, on compterait un même mot plus d'une fois.

Par conséquent, si M annule un polynôme[5], N l'annule aussi. En particulier, M annule son polynôme minimum donc N annule le polynôme minimum de M. Pour conclure, il suffit de se rappeler que le polynôme minimum de N divise tout polynôme annulé par N.

■

4. Monoïde syntaxique

Lorsqu'on étudie les langages formels, certaines de leurs propriétés peuvent s'exprimer en termes algébriques en introduisant la notion de monoïde syntaxique. Le but de cette section est de fournir quelques rudiments concernant cet outil puissant[6].

Définition V.4.1. Soit $L \subseteq \Sigma^*$ un langage (régulier ou non). On définit sur Σ^* la relation suivante. Soient $u, v \in \Sigma^*$. On a

$$u \equiv_L v \Leftrightarrow (\forall x, y \in \Sigma^* : xuy \in L \Leftrightarrow xvy \in L).$$

Il est facile de vérifier qu'il s'agit d'une relation d'équivalence sur Σ^* et même d'une congruence (à droite et à gauche), i.e., pour tout $\sigma \in \Sigma$,

$$u \equiv_L v \Rightarrow (u\sigma \equiv_L v\sigma \text{ et } \sigma u \equiv_L \sigma v).$$

On parle souvent de la *congruence syntaxique* $\equiv_L$ et on dit que u et v sont *syntaxiquement équivalents*.

Remarque V.4.2. Nous avons montré que $\equiv_L$ est une congruence à gauche et à droite. Mais en toute généralité, une *congruence* doit respecter la propriété suivante: si $x \equiv_L x'$ et si $y \equiv_L y'$, alors $xy \equiv_L x'y'$, c'est-à-dire qu'elle doit bien se comporter par rapport au produit envisagé, à savoir ici, la concaténation. Et ceci est bien le cas car pour tous $\alpha, \beta \in \Sigma^*$, il vient

$$\alpha xy\beta \in L \Leftrightarrow \alpha x'y\beta \in L \Leftrightarrow \alpha x'y'\beta \in L.$$

Dans cette section, on notera simplement $[w]$ la classe d'équivalence pour $\equiv_L$ étant convenu que la relation $\equiv_L$ est sous-entendue. Bien évidemment, $[w]$ est un ensemble de mots, donc un langage.

[5]Soit $P(z) = \sum_{i=0}^{n} a_i z^i$ tel que $P(M) = 0$. En particulier, pour tous p', q', $\sum_{i=0}^{n} a_i (M^i)_{p',q'} = 0$ et donc $\sum_{q' \in \Phi^{-1}(q)} \sum_{i=0}^{n} a_i (M^i)_{p',q'} = 0$. En permutant les sommes, on obtient $\sum_{i=0}^{n} a_i \sum_{q' \in \Phi^{-1}(q)} (M^i)_{p',q'} = 0$ puis $\sum_{i=0}^{n} a_i (N^i)_{p,q} = 0$ et donc $P(N) = 0$.

[6]En effet, on peut par exemple montrer qu'un langage peut s'exprimer à partir d'ensembles finis en utilisant un nombre fini d'opérations d'union, de concaténation, d'intersection et de complémentation (on parle alors à juste titre de langage sans étoile, ou "star-free") si et seulement si son monoïde syntaxique ne contient que des sous-groupes triviaux. Ce résultat de nature algébrique est dû à M.P. Schützenberger.

Exemple V.4.3. Soit L, le langage formé des mots sur $\{a, b\}$ ne contenant pas deux bb consécutifs. On remarque tout d'abord que

$$xay \in L \Leftrightarrow (x \in L \text{ et } y \in L).$$

De là, on en tire que la classe de a pour la congruence syntaxique $\equiv_L$ est de la forme

$$[a] = \{awa \mid w \in L\} \cup \{a\}.$$

En particulier, $\varepsilon \notin [a]$.

Nous allons voir qu'on peut munir l'ensemble quotient $\Sigma^*/\equiv_L$, i.e., l'ensemble des classes d'équivalence pour $\equiv_L$, d'une structure de monoïde.

Définition V.4.4. Soit l'opération

$$\circ : \Sigma^*/\equiv_L \times \Sigma^*/\equiv_L \to \Sigma^*/\equiv_L : ([x], [y]) \mapsto [x] \circ [y]$$

définie par

$$[x] \circ [y] = [z] \text{ si } [x] \cdot [y] \subseteq [z]$$

où $\cdot$ représente l'opération de concaténation de langages[7]. L'application $\circ$ est bien définie car au vu de la remarque V.4.2, la définition ne dépend pas du représentant choisi.

Remarque V.4.5. Il est évident que

$$[x] \circ [y] = [xy].$$

Remarque V.4.6. On remarque qu'effectivement, la concaténation de deux classes $[x] \cdot [y]$ est formé de mots équivalents mais qu'en général, il s'agit d'un sous-ensemble strict de la classe d'équivalence $[xy]$.

En considérant à nouveau le langage L formé des mots sur $\{a, b\}$ ne contenant pas deux bb consécutifs, on a

$$[a] \circ [a] = [aa] = [a].$$

Cependant, le mot aba (ou même a) appartient bien à $[a]$ mais ne peut pas se factoriser sous la forme

$$aba = xy \text{ avec } x, y \in [a].$$

Ceci montre bien que $[a] \cdot [a] \subsetneq [a]$.

Proposition V.4.7. *Muni de l'opération $\circ$, l'ensemble quotient $\Sigma^*/\equiv_L$ possède une structure de monoïde.*

Démonstration. Le neutre est $[\varepsilon]$, i.e., pour tout $x \in \Sigma^*$, on a

$$[x] \circ [\varepsilon] = [x].$$

De plus, l'opération $\circ$ est associative, i.e., pour tous $x, y, z \in \Sigma^*$,

$$([x] \circ [y]) \circ [z] = [x] \circ ([y] \circ [z]).$$

[7]Opération tout à fait naturelle, puisqu'une classe d'équivalence pour $\equiv_L$ est, comme nous l'avons déjà remarqué, un ensemble de mots. Ainsi, on définit une nouvelle opération $\circ$, à partir d'une ancienne, la concaténation.

Cela résulte de la remarque V.4.5.

■

Définition V.4.8. Le monoïde $(\Sigma^*/\equiv_L, \circ)$ est le *monoïde syntaxique* de L. On note simplement π le *morphisme canonique*

$$\pi : \Sigma^* \to \Sigma^*/\equiv_L : w \mapsto [w].$$

Le résultat suivant fournit un moyen explicite pour rechercher le monoïde syntaxique d'un langage.

Théorème V.4.9. *Soient $L \subseteq \Sigma^*$ un langage et w, w' deux mots sur Σ. On a $w \equiv_L w'$ si et seulement si pour tout état q de l'automate minimal de L, $\delta_L(q, w) = \delta_L(q, w')$.*

Démonstration. Supposons qu'il existe dans l'automate minimal de L, un état tel que

$$\delta_L(q, w) \neq \delta_L(q, w').$$

Puisque l'automate minimal est réduit (cf. proposition IV.3.10), il existe un mot $z \in \Sigma^*$ tel que $\delta_L(\delta_L(q, w), z)$ soit final et $\delta_L(\delta_L(q, w'), z)$ ne le soit pas (ou réciproquement, mais par souci de simplification, nous supposerons être dans un tel cas de figure). De plus, l'automate minimal est accessible. Cela signifie qu'il existe un mot $x \in \Sigma^*$ tel que $\delta_L(q_{0,L}, x) = q$. Schématiquement, nous avons la situation suivante reprise en figure V.10. Par

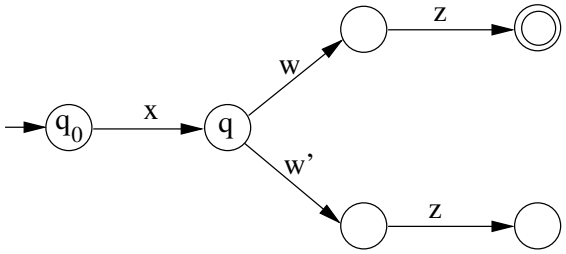


FIGURE V.10. Situation dans l'automate minimal.

conséquent, $xwz \in L$ et $xw'z \notin L$. Ainsi, les deux mots w et w' ne sont pas syntaxiquement équivalents.

Passons à la réciproque. Si pour tout état q de $\mathcal{A}_L$, on a $\delta_L(q, w) = \delta_L(q, w')$, alors pour tout mot $x \in \Sigma^*$,

$$\delta_L(q_{0,L}, xw) = \delta(q_{0,L}, xw')$$

et dès lors, puisque l'automate minimal est déterministe, pour tout $y \in \Sigma^*$, on a

$$\delta_L(q_{0,L}, xwy) = \delta_L(q_{0,L}, xw'y).$$

Schématiquement, on a la situation représentée à la figure V.11 Ainsi, pour tous $x, y \in \Sigma^*$,

$$xwy \in L \Leftrightarrow xw'y \in L.$$

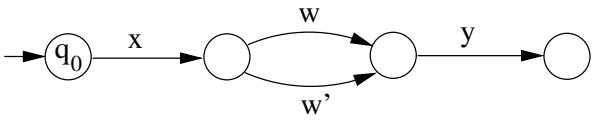


FIGURE V.11. Situation dans l'automate minimal.

■

Soit $\mathcal{A}_L = (Q_L, q_{0,L}, F_L, \sigma, \delta_L)$ l'automate minimal d'un langage L. A tout mot $w \in \Sigma$, il correspond une unique fonction $f_w : Q_L \to Q_L$ définie par

$$f_w : Q_L \to Q_L : q \mapsto \delta_L(q, w).$$

Pour un langage L donné, on note

$$\mathcal{H}_L = \{f_w \mid w \in \Sigma^*\}.$$

Muni de l'opération de composition de fonctions, $\mathcal{H}_L$ est un monoïde ayant id pour neutre. On a $f_{ww'} = f_{w'} \circ f_w$ car pour tout q,

$$f_{ww'}(q) = q.ww' = (q.w).w' = f_{w'}(f_w(q)).$$

Corollaire V.4.10. *L'application*

$$R : \Sigma^*/\equiv_L \to \mathcal{H}_L : [w] \mapsto f_w$$

est un isomorphisme de monoïdes.

Démonstration. Cela résulte directement du théorème précédent. En effet, deux mots w et w' sont syntaxiquement équivalents si et seulement si ils ont la même action sur tous les états de Q_L, c'est-à-dire, si $f_w = f_{w'}$.

■

Le théorème V.4.9 a été énoncé pour un langage L arbitraire. Dans le cas d'un langage régulier, on obtient un monoïde syntaxique fini.

Corollaire V.4.11. *Un langage L est régulier si et seulement si son monoïde syntaxique est fini.*

Démonstration. Si l'automate minimal $\mathcal{A}_L$ de L est fini, l'ensemble Q_L des états de $\mathcal{A}_L$ possède un nombre fini n d'éléments. Le nombre de fonctions de Q dans Q est au plus n^n et par conséquent, le monoïde syntaxique de L possède au plus n^n éléments. Pour la réciproque, au vu du théorème V.4.9, si $\delta_L(q_0, w) \neq \delta_L(q_0, w')$, alors $w \not\equiv_L w'$. Par conséquent, le nombre d'états de l'automate minimal de L est majoré par le nombre de classes du monoïde syntaxique de L. De là, si $\Sigma^*/\equiv_L$ est fini, alors l'automate minimal de L est fini et le langage L est régulier.

■

Le corollaire V.4.10 permet de calculer la table du monoïde syntaxique d'un langage régulier L.

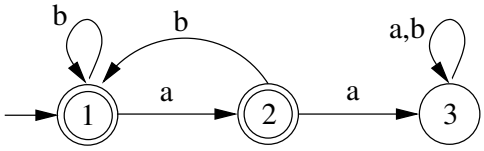


FIGURE V.12. AFD acceptant les mots ne contenant pas aa.

Exemple V.4.12. Considérons une fois encore l'automate minimal du langage formé des mots ne contenant pas deux a consécutifs. Les fonctions de $\mathcal{H}_L$ sont données par

q	$f_\varepsilon(q)$	$f_a(q)$	$f_b(q)$	$f_{aa}(q)$	$f_{ab}(q)$	$f_{ba}(q)$
1	1	2	1	3	1	2
2	2	3	1	3	3	2
3	3	3	3	3	3	3

Cet ensemble aurait pu contenir au plus $3^3 = 27$ fonctions. Pour vérifier qu'il n'y a pas d'autres applications dans $\mathcal{H}_L$, on peut construire de proche en proche un graphe fini de la manière suivante. Si $Q_L = \{1, \ldots, n\}$, alors on initialise la construction avec un unique sommet correspondant au n-uple d'états $(1, \ldots, n)$. Ensuite, on applique les fonctions f_σ, $\sigma \in \Sigma$, à chaque état nouvellement créé. Si $(q_1, \ldots, q_n)$ est un état du graphe, alors on trace un arc de label σ joignant ce sommet au sommet $(f_\sigma(q_1), \ldots, f_\sigma(q_n))$. La procédure s'arrête lorsque plus aucun nouvel état n'est créé. L'application de cette procédure donne le graphe de la figure V.13.

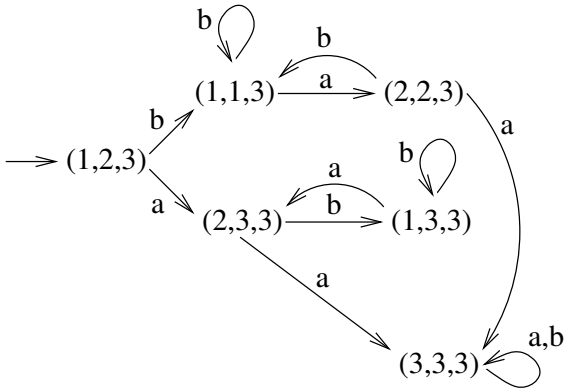


FIGURE V.13. Graphe associé à $\mathcal{H}_L$.

Deux mots w et w' sont syntaxiquement équivalents si la lecture de ces deux mots depuis l'état initial aboutit dans un même état. En effet, par construction du graphe, cela signifie que $f_w = f_{w'}$ et donc que les deux mots ont tous deux la même action sur les états de l'automate minimal. Par exemple, $abba \equiv_L a$ et $bb \equiv_L b$.

Si on note f_w simplement w, on obtient la table du monoïde $\mathcal{H}_L$ muni de l'opération de composition :

	ε	a	b	aa	ab	ba
ε	ε	a	b	aa	ab	ba
a	a	aa	ab	aa	aa	a
b	b	ba	b	aa	b	ba
aa	aa	aa	aa	aa	aa	aa
ab	ab	a	ab	aa	ab	a
ba	ba	aa	b	aa	aa	ba

Au vu de l'isomorphisme donné dans le corollaire V.4.10, il s'agit également de la table du monoïde syntaxique de L pour laquelle w représente alors $[w]$.

Nous allons considérer un second exemple. Ceci s'avérera particulièrement utile pour illustrer les résultats de la section suivante.

Exemple V.4.13. Considérons le langage L formé des mots comprenant un nombre pair de a et de b. L'automate minimal de L est représenté à la figure V.14. En effectuant les mêmes développements que dans l'exemple

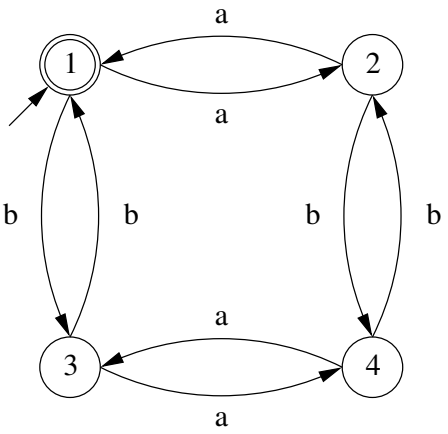


FIGURE V.14. Automate minimal de $L = \{w :| \ |w|_a \equiv |w|_b \equiv 0 \pmod 2\}$.

précédent, on obtient la table de multiplication du monoïde syntaxique de L :

	ε	a	b	ab
ε	ε	a	b	ab
a	a	ε	ab	b
b	b	ab	ε	a
ab	ab	b	a	ε

On s'aperçoit que chaque élément est idempotent et que le monoïde syntaxique de L est en fait un groupe (puisque chaque élément possède un inverse, à savoir lui-même). Ce groupe est isomorphe à un sous-groupe de $\mathcal{S}_4$ des

permutations à 4 éléments constitué des permutations suivantes,

$$\begin{pmatrix}1&2&3&4\\1&2&3&4\end{pmatrix}, \begin{pmatrix}1&2&3&4\\2&1&4&3\end{pmatrix}, \begin{pmatrix}1&2&3&4\\3&4&1&2\end{pmatrix}, \begin{pmatrix}1&2&3&4\\4&3&2&1\end{pmatrix}.$$

5. Langages sans étoile

Cette section met en lumière une application du monoïde syntaxique. En effet, l'étude de ce dernier permet de déterminer aisément si un langage est ou non "sans étoile". Commençons donc par définir ce que l'on entend par langage sans étoile.

Définition V.5.1. Un langage régulier $L \subseteq \Sigma^*$ est dit *sans étoile* s'il peut être obtenu à partir de langages finis (ou vides) en appliquant un nombre fini de fois des opérations d'union, d'intersection, de concaténation et de complémentation (par rapport à Σ^*).

En résumé, on s'interdit d'utiliser l'étoile de Kleene.

Remarque V.5.2. Il serait facile de définir les *expressions régulières sans étoile* permettant de générer exactement les langages réguliers sans étoile. Il suffit pour cela d'adapter les règles de construction données à la définition I.3.1.

Exemple V.5.3. Soit $\Sigma = \{a, b\}$. Par exemple, Σ^* est sans étoile car il s'obtient comme

$$\Sigma^* \setminus \emptyset.$$

Le langage L des mots sur Σ ne contenant pas le facteur aa est aussi sans étoile. En effet,

$$L = \Sigma^* \setminus (\Sigma^* aa \Sigma^*)$$

et nous avons vu que Σ^* était lui-même sans étoile. Enfin, le langage $(ab)^*$ est aussi sans étoile car

$$(ab)^* = \Sigma^* \setminus (b\Sigma^* + \Sigma^* a + \Sigma^* aa\Sigma^* + \Sigma^* bb\Sigma^*).$$

Comme le montre ce dernier exemple, il peut être assez difficile de déterminer si un langage donner peut ou non être mis sous une forme "sans étoile". En particulier, comment pouvons-nous démontrer qu'un langage régulier donné n'est pas sans étoile ?

Nous allons voir que la théorie du monoïde syntaxique permet de donner un algorithme efficace pour répondre à cette question.

Les résultats suivants sont d'application dans tout semi-groupe fini. Rappelons qu'un *semi-groupe* est un ensemble muni d'une opération (binaire, interne et partout définie) associative[8]. Dans un semi-groupe S, un élément e est qualifié de *neutre* si

$$\forall s \in S,\ s \cdot e = s = e \cdot s.$$

[8]Un monoïde est un semi-groupe possédant un neutre.

De même, un élément e est qualifié de *zéro* si

$$\forall s \in S,\ s \cdot e = e = e \cdot s.$$

Proposition V.5.4. *Si un semi-groupe possède un neutre (resp. un zéro), alors celui-ci est unique.*

Démonstration. C'est trivial.

■

Théorème V.5.5. *Soit $(S, \cdot)$ un semi-groupe fini. Pour tout $a \in S$, il existe des entiers positifs m et r tels que $a, \ldots, a^m, \ldots, a^{m+r-1}$ soient distincts et mais tels que $a^{m+r} = a^m$. De plus, la restriction de l'opération $\cdot$ à l'ensemble $\mathcal{C}_a = \{a^m, \ldots, a^{m+r-1}\}$ forme un groupe cyclique d'ordre r.*

Démonstration. Puisque S est fini, parmi $a, a^2, \ldots, a^{\#S+1}$ au moins deux éléments sont égaux. Soient $m, r \geq 1$ tels que $a^m = a^{m+r}$ soit la première répétition d'un même élément dans la liste donnée ci-dessus. En particulier, $a, \ldots, a^{m-1}, a^m, \ldots, a^{m+r-1}$ sont deux à deux distincts.

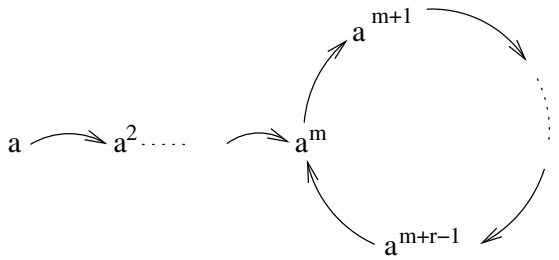


FIGURE V.15. Indice et période.

Il est clair que $a^{m+t} = a^{m+s}$ si et seulement si $t \equiv s \pmod{r}$. Ainsi, pour tous $i, j \geq 0$,

$$a^{m+i} \cdot a^{m+j} = a^{m+k} \quad \text{où} \quad k \equiv m + i + j \pmod{r}.$$

Ceci montre que le produit de deux éléments de $\mathcal{C}_a$ appartient encore à $\mathcal{C}_a$ (i.e., $\mathcal{C}_a$ forme un sous-semi-groupe).

Il nous reste à vérifier que $\mathcal{C}_a$ est en fait un groupe cyclique en montrant qu'il est isomorphe à $(\mathbb{Z}_r, +)$. Soit

$$\varphi : \mathcal{C}_a \to \mathbb{Z}_r : a^{m+i} \mapsto m + i \pmod{r}.$$

Puisque $m, m+1, \ldots, m+r-1$ sont r entiers consécutifs, il est clair que φ est une bijection. Il reste à vérifier qu'il s'agit d'un homomorphisme. D'une part,

$$\varphi(a^{m+i} \cdot a^{m+j}) = \varphi(a^{2m+i+j}) = 2m + i + j \pmod{r}$$

et d'autre part,

$$\varphi(a^{m+i}) + \varphi(a^{m+j}) = m + i + m + j \pmod{r}.$$

■

Définition V.5.6. Soient S un semi-groupe et $a \in S$. L'entier m (resp. r) dont il est question dans le théorème V.5.5 est appelé l'*indice* (resp. la *période*) de l'élément a.

Exemple V.5.7. Prenons comme semi-groupe S, le monoïde syntaxique obtenu dans l'exemple V.4.12. Si, une fois encore, on s'autorise à noter $[w]$ simplement w, on trouve les périodes suivantes pour les éléments du monoïde,

			indice m	période r
ε	$\varepsilon^2 = \varepsilon$		1	1
a	$a^2 = aa$	$a^3 = aa$	2	1
b	$b^2 = b$		1	1
aa	$aa^2 = aa$		1	1
ab	$ab^2 = ab$		1	1
ba	$ba^2 = ba$		1	1

En guise de comparaison, considérons cette fois, le monoïde syntaxique donné dans l'exemple V.4.13. On trouve,

			indice m	période r
ε	$\varepsilon^2 = \varepsilon$		1	1
a	$a^2 = \varepsilon$	$a^3 = a$	1	2
b	$b^2 = \varepsilon$	$b^3 = b$	1	2
ab	$ab^2 = \varepsilon$	$ab^3 = ab$	1	2

On voit donc dans le premier exemple que tous les éléments sont de période 1, ce qui n'est pas le cas du second exemple.

Avant d'énoncer le résultat suivant, rappelons qu'un groupe est *trivial* s'il est restreint au seul neutre. Si S est un semi-groupe sans neutre (i.e., si S est un semi-groupe qui n'est pas un monoïde), on introduit le monoïde S^1 où $S^1 = S \cup \{1\}$ avec 1, un nouvel élément n'appartenant pas S. On étend l'opération de S comme suit,

$$\forall s \in S, 1 \cdot s = s = s \cdot 1.$$

Si S est un monoïde, on pose par convention $S^1 = S$.

Théorème V.5.8. *Soit $(S, \cdot)$ un semi-groupe fini. Les conditions suivantes sont équivalentes.*

(i) *Les sous-groupes de S maximaux (pour l'inclusion) sont triviaux.*
(ii) *Tout élément de S est de période* 1.
(iii) *Il existe $n > 0$ tel que pour tout $a \in S$, $a^n = a^{n+1}$.*

Démonstration. $(i) \Rightarrow (ii)$. Supposons que les seuls sous-groupes de S sont triviaux. Avec les notations du théorème précédent, pour tout $a \in S$, $\mathcal{C}_a$ est trivial. Ceci signifie en particulier que la période de a vaut 1.

$(ii) \Rightarrow (iii)$. Supposons à présent que tout élément de S est de période 1 et notons $\#S = n$. Soit $a \in S$. Nous allons montrer que $a^n = a^{n+1}$. Parmi $a, a^2, \ldots, a^{n+1}$, on trouve au moins deux fois le même élément. Par

conséquent, l'indice i de a est au plus n. Or par hypothèse, a est de période 1. Par conséquent, on obtient

$$a^i = a^{i+1} = \cdots = a^n = a^{n+1} \quad \text{avec } i \leq n.$$

$(iii) \Rightarrow (i)$. Soient G un sous-groupe de S et x, y deux éléments quelconques de G (pas nécessairement distincts). Il existe $a, b, c, d \in G$ tels que

Pensez à la table de Cayley d'un groupe.

$$ax = y, \quad xb = y, \quad cy = x, \quad yd = x.$$

De là, on tire $x = cy = cxb$ et donc $x = c^n x b^n$. Enfin, on obtient

$$y = xb = c^n x b^{n+1} = c^n x b^n = x$$

et le groupe est donc trivial puisqu'il est restreint à un seul élément.

■

Définition V.5.9. Un semi-groupe satisfaisant les propriétés du théorème précédent est qualifié d'*apériodique*.

Exemple V.5.10. Au vu de l'exemple V.5.7, le monoïde syntaxique de l'exemple V.4.12 est apériodique. On pourrait vérifier qu'il ne contient que des sous-groupes triviaux. En effet, la restriction du produit aux ensembles

$$\{\varepsilon\},\ \{b\},\ \{aa\},\ \{ab\},\ \{ba\}$$

en font des groupes restreints à un unique élément idempotent. Il est clair que si s est un élément idempotent d'un semi-groupe S, alors $\{s\}$ est un sous-groupe (trivial) de S.

Par contre, le monoïde syntaxique de l'exemple V.4.13 n'est pas apériodique. En effet, nous avions déjà remarqué qu'il s'agissait d'un groupe. En outre, la restriction du produit à l'ensemble $\{\varepsilon, a\}$ est aussi un groupe (non trivial).

Le théorème précédent nous fournit un algorithme pour décider si un semi-groupe fini est apériodique.

Test du caractère apériodique d'un semi-groupe.

(1) Choisir un élément quelconque $a \in S$ et calculer l'ensemble a^+ de ses puissances successives.

(2) Trois cas peuvent se présenter:

- La période de a n'est pas 1. L'algorithme s'achève, S n'est *pas apériodique*.
- La période de a est 1 et $S = a^+$. L'algorithme s'achève, S est *apériodique*.
- La période de a est 1 et $S \neq a^+$. Remplacer S par $S \setminus a^+$ et répéter **(1)**.

Nous pouvons à présent énoncer le théorème de Schützenberger caractérisant les langages sans étoiles.

Théorème V.5.11 (Schützenberger). *Un langage régulier est sans étoile si et seulement son monoïde syntaxique est apériodique.*

La preuve de ce résultat sort du cadre introductif de ce cours. On pourra par exemple consulter l'ouvrage de Lawson ou de Perrin (cf. bibliographie).

Corollaire V.5.12. *Déterminer si un langage régulier est sans étoile est un problème décidable algorithmiquement.*

Démonstration. C'est immédiat. On dispose d'un algorithme pour tester si un semi-groupe est apériodique et le monoïde syntaxique d'un langage régulier peut être calculé algorithmiquement.

■

Exemple V.5.13. Au vu de l'exemple V.5.7, le langage formé des mots sur $\{a, b\}$ ne contenant pas le facteur aa est sans étoile. Par contre le langage formé des mots contenant un nombre pair de a et un nombre pair de b ne l'est pas. (Comparez ces deux exemples avec le résultat annoncé dans l'exercice V.6.16.)

6. Exercices

6.1. Transduction.

Exercice V.6.1. Supposons que les positions des lettres d'un mot soient comptées de gauche à droite. Ainsi,

$$w = w_1 \cdots w_n,\ w_i \in \Sigma$$

pour un mot w de longueur n. Construire un transducteur $\mathcal{T}$ qui transforme chaque a se trouvant en position de la forme $3i$ (resp. $3i+1$, $3i+2$) en abc (resp. bac, aac) et chaque b se trouvant en position de la forme $3i$ (resp. $3i+1$, $3i+2$) en bca (resp. bac, bcb), $i \in \mathbb{N}$. Donner une expression régulière du langage

$$\mathcal{T}(a^*b^*).$$

6.2. Problèmes de dénombrement.

Exercice V.6.2. Soit $L \subset \Sigma^*$ un langage. On dénote par $\rho_L(n)$, le nombre de mots de longueur n appartenant à L. Si \$ est une lettre n'appartenant pas à Σ, vérifier que

$$\#[(\{\$\} \sqcup\!\sqcup L) \cap (\Sigma \cup \{\$\})^n] = n\,\rho_L(n-1), \quad \forall n \geq 1.$$

Utiliser ce résultat pour construire un langage régulier L tel que

$$\rho_L(n) = n^2.$$

Même question avec cette fois, $\rho_L(n) = n^3$.

Exercice V.6.3. On considère le langage L formé des mots sur $\{a, b\}$ ayant un nombre impair de b.

- Quel est l'automate minimal de L ?
- Donner la matrice d'adjacence de cet automate.
- En déduire une relation de récurrence linéaire pour $\rho_L(n)$.

- Trouver une formule close pour $\rho_L(n)$.

Exercice V.6.4. On considère le langage L formé des mots sur $\{a, b, c\}$ ne commençant pas par c et ne contenant pas le facteur ac.

- Quel est l'automate minimal de L ?
- Soit la série génératrice
$$F(X) = \sum_{n \geq 0} \rho_L(n)\, X^n.$$
Montrer que
$$F(X) = \frac{1 - X}{X^2 - 3X + 1}.$$
- En déduire une formule close pour $\rho_L(n)$.

Exercice V.6.5. On considère le langage $L = a^*b^*$.

- Quel est l'automate minimal de L ?
- Donner la matrice d'adjacence de cet automate.
- En déduire une relation de récurrence linéaire pour $\rho_L(n)$.
- Montrer que la série génératrice de $\rho_L(n)$ est de la forme
$$F(X) = \frac{1}{(1 - X)^2}$$
- En développant en série de puissances, en conclure que $\rho_L(n) = n + 1$.

Exercice V.6.6. On considère l'alphabet $\Sigma = \{a, b, c\}$ et le langage régulier sur Σ formé des mots ne contenant pas le facteur "aa". Ce langage est accepté par l'automate fini déterministe $\mathcal{A} = (\{1, 2, 3\}, 1, \{1, 2\}, \Sigma, \delta)$ où la fonction de transition $\delta : \{1, 2, 3\} \times \Sigma \to \{1, 2, 3\}$ est donnée par

δ	a	b	c
1	2	1	1
2	3	1	1
3	3	3	3.

- Donner une relation de récurrence linéaire pour la suite $\rho_L(n) = \#(L \cap \Sigma^n)$ comptant le nombre de mots de longueur n dans L.
- Par une méthode au choix, en déduire une formule close pour $\rho_L(n)$.

6.3. Monoïde syntaxique et langages sans étoile.

Exercice V.6.7. Soit L un langage. Démontrer que L est une union de classes d'équivalence pour la congruence syntaxique $\equiv_L$.

Exercice V.6.8. Soit $L = a^*b^*$. Donner un représentant de chacune des classes d'équivalence du monoïde syntaxique de L. On choisira, si possible, un représentant de longueur minimale dans chaque classe. Construire la table de multiplication de ce monoïde. Le monoïde syntaxique est-il apériodique?

Exercice V.6.9. Même question avec le langage L formé des mots sur $\{a, b\}$ comprenant un nombre pair de a. S'agit-il d'un langage sans étoile ?

Exercice V.6.10. Même question avec le langage L formé des mots acceptés par l'automate dessiné à la figure V.16

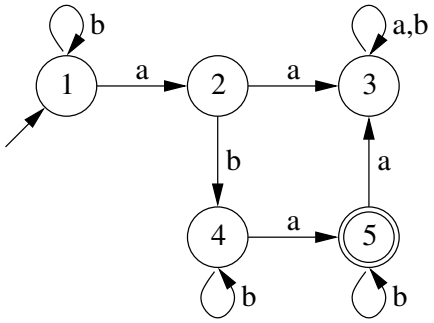


FIGURE V.16. Un AFD dont on recherche le monoïde syntaxique du langage associé.

Exercice V.6.11. Même question avec le langage L formé des mots sur $\{a, b\}$ qui sont formés exclusivement de b en nombre impair ou qui comprennent un nombre de a qui est multiple strictement positif de 3. Montrer que le monoïde syntaxique se décompose en deux sous-groupes cycliques dont on donnera à chaque fois un générateur.

Exercice V.6.12. On considère le langage accepté par l'automate de la figure V.17. Après avoir vérifier que cet automate était minimal, montrer

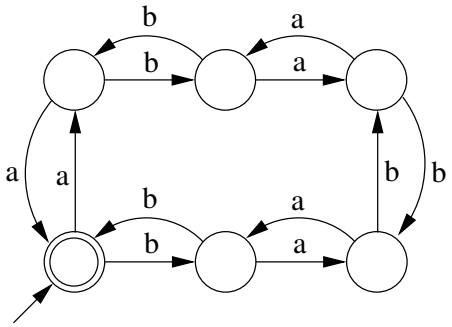


FIGURE V.17. Un AFD dont on recherche le monoïde syntaxique.

que le monoïde syntaxique de ce langage est isomorphe à $\mathcal{S}_3$ (le groupe des permutations de $\{1, 2, 3\}$).

Exercice V.6.13. Pour les automates repérésentés à la figure V.18, vérifier qu'ils sont minimaux. Calculer la table de multiplication du monoïde syntaxique et déterminer dans chaque cas s'il s'agit d'un langage sans étoile.

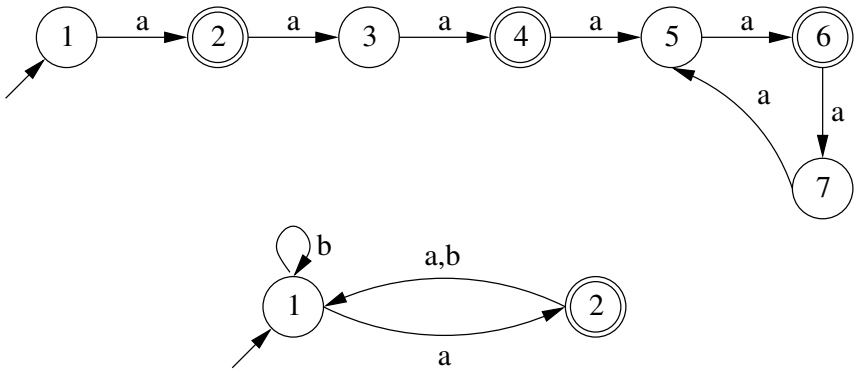


FIGURE V.18. Deux AFD.

Exercice V.6.14. Pour le langage accepté par l'automate de la figure V.19, démontrer que

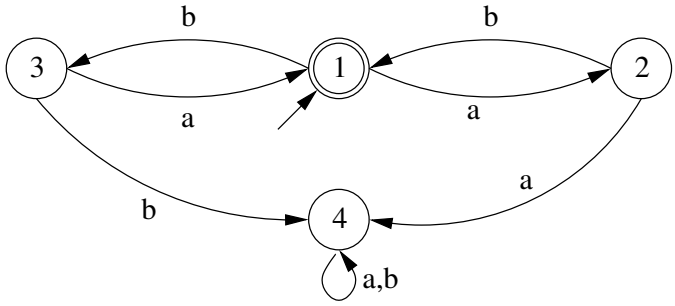


FIGURE V.19. Un AFD.

$aba \equiv a,\ bab \equiv b,\ b^3 \equiv a^3,\ a^4 \equiv a^3,\ a^3b \equiv a^3,\ a^2ba \equiv a^2,\ ab^3 \equiv a^3,\ ba^3 \equiv a^3,$
$ba^2b \equiv ab^2a,\ b^2ab \equiv b^2,\ a^2b^2a \equiv a^2b,\ a^2b^3 \equiv a^3,\ ab^2a^2 \equiv ba^2,\ ab^2ab \equiv ab^2$
et

$$b^2a^3 \equiv a^3,\ b^2a^2b \equiv b^2a.$$

En déduire que a^3 est un zéro et que ces 16 relations peuvent se simplifer en

$$a^3 \equiv b^3,\ aba \equiv a,\ bab \equiv b,\ ba^2b \equiv ab^2a,\ a^2b^2a \equiv a^2b,\ ab^2a^2 \equiv ba^2$$

pour décrire complètement le monoïde syntaxique. Ainsi, ces 6 relations donnent une représentation bien plus succinte que la table de multiplication du monoïde.

Définition V.6.15. Un AFD $\mathcal{A}$ est *sans permutation* s'il n'existe aucun mot w réalisant une permutation non triviale d'un sous ensemble d'états de $\mathcal{A}$, i.e., s'il n'existe pas de mot w et de sous-ensemble d'états $\{q_1, \ldots, q_r\}$ tels que

$$q_1.w = q_{\nu_1},\ \ldots, q_r.w = q_{\nu_r}$$

où ν est une permutation de $\{1, \ldots, r\}$ distincte de l'identité.

Exercice V.6.16. Démontrer qu'un langage régulier est sans étoile si et seulement si son automate minimal est sans permutation.

CHAPITRE VI

Introduction aux langages algébriques

Les chapitres précédents nous ont donnés un aperçu assez complet des langages réguliers et de leurs principales propriétés. En particulier, nous avons constaté, et ce à plusieurs reprises, que des langages comme $\{a^n b^n \mid n \in \mathbb{N}\}$, pourtant "relativement simples" d'un point de vue syntaxique, n'étaient pas réguliers. Dans les pages qui suivent, nous allons présenter une famille de langages qui sont générés par des méthodes plus riches que les expressions régulières. Plus précisément, nous allons introduire la notion de grammaire hors contexte. Un langage généré par une telle grammaire sera dit algébrique (ou hors contexte). Historiquement, ces langages ont été introduits[1] par N. Chomsky dans le but initial de formaliser les propriétés grammaticales de langues naturelles comme l'anglais ou le français. Il s'est avéré par la suite que ces notions étaient particulièrement bien adaptées à la syntaxe des langages de programmation.

1. Premières définitions

Commençons par un exemple introductif présentant le concept de grammaire.

Exemple VI.1.1. Pour construire une phrase grammaticalement correcte en français, on peut procéder comme suit

PHRASE	$\rightarrow$	SUJET VERBE COMPLEMENT
SUJET	$\rightarrow$	LUDOVIC | MICHEL | NICOLAS | THIERRY
VERBE	$\rightarrow$	VOIT | MANGE | PORTE
COMPLEMENT	$\rightarrow$	ARTICLE NOM ADJECTIF
ARTICLE	$\rightarrow$	UN | LE
NOM	$\rightarrow$	LIVRE | PLAT | WAGON
ADJECTIF	$\rightarrow$	BLEU | ROUGE | VERT

Ainsi, sans vouloir formaliser plus que nécessaire, avec les règles données ci-dessus, on peut construire au moyen de substitutions successives des phrases comme

NICOLAS PORTE UN LIVRE VERT

ou

MICHEL MANGE LE WAGON BLEU.

[1] N. Chomsky, On certain formal properties of grammars, *Inform. and Control*, 137–167, 1959.

On peut formaliser cet exemple de la manière suivante.

Définition VI.1.2. Soient V et Σ deux alphabets finis (supposés disjoints). Une *grammaire hors contexte*, ou *grammaire algébrique*, est la donnée d'un 4-uple

$$G = (V, \Sigma, P, S)$$

où $P \subset V \times (V \cup \Sigma)^*$ est un ensemble fini, appelé l'ensemble des *règles de dérivation* (ou *productions*) de G et $S \in V$ est le *symbole initial* de G. Les éléments de l'alphabet V sont appelés *variables* (ou *symboles non terminaux*) et les éléments de l'alphabet Σ sont les *symboles terminaux*. Nous prendrons généralement la convention de représenter les symboles non terminaux par des lettres majuscules et les symboles terminaux par des minuscules.

Soient $A \in V$ une variable, $w \in (V \cup \Sigma)^*$ un mot et $(A, w) \in P$ une règle de dérivation. On dit que A (resp. w) est le *premier* (resp. *second*) *membre* de la production (A, w). Si $A \in V$ est une variable et $(A, w_1), \ldots, (A, w_n) \in P$ sont des productions ayant A pour premier membre et où $w_1, \ldots, w_n \in (V \cup \Sigma)^*$, alors on note

$$A \to w_1 \,|\, w_2 \,|\, \cdots \,|\, w_n.$$

Si w peut s'écrire xAy avec $A \in V$ et $x, y \in (V \cup \Sigma)^*$, alors on note

$$w \Rightarrow_G z$$

lorsque $z = xuy$ avec $(A, u) \in P$. On dit alors que z est obtenu grâce à une *dérivation* de longueur 1. En d'autres termes, on a remplacé dans w une occurence d'un non terminal A par le second membre u d'une production $A \to u$ de G ayant A comme premier membre. Si G est sous-entendu, on s'autorise à écrire simplement $\Rightarrow$ au lieu de $\Rightarrow_G$. On note $\Rightarrow^*$ la fermeture réflexive et transitive de $\Rightarrow$. Ainsi, $w \Rightarrow^* z$ si $z = w$ ou s'il existe des mots $w_1, \ldots, w_n \in (V \cup \Sigma)^*$, $n \geq 0$, tels que

$$w \Rightarrow w_1 \Rightarrow w_2 \Rightarrow \cdots \Rightarrow w_n \Rightarrow z.$$

Dans ce dernier cas, on dit que z est obtenu à partir de w par une *dérivation* de longueur $n + 1$. Enfin, le *langage généré par* G est l'ensemble des mots de Σ qui s'obtiennent par dérivation à partir du symbole initial S, i.e.,

$$L(G) = \{w \in \Sigma^* \mid S \Rightarrow^* w\}.$$

Un langage $L \subset \Sigma^*$ est *algébrique* ou *hors contexte* s'il existe une grammaire hors contexte $G = (V, \Sigma, P, S)$ telle que $L = L(G)$. Enfin, deux grammaires G et H sont *équivalentes* si elles génèrent le même langage, i.e., si $L(G) = L(H)$.

Exemple VI.1.3. Soit la grammaire hors contexte $G = (V, \Sigma, P, S)$ où $V = \{S, A\}$, $\Sigma = \{a, b\}$, et les productions de G données par

$$\begin{array}{rcl} S & \to & AA \\ A & \to & AAA \mid bA \mid Ab \mid a\ . \end{array}$$

Le mot *ababaa* appartient à $L(G)$ car

$$\begin{aligned}\underline{S} &\Rightarrow \underline{A}A \Rightarrow a\underline{A} \Rightarrow a\underline{A}AA \\ &\Rightarrow ab\underline{A}AA \Rightarrow aba\underline{A}A \Rightarrow abab\underline{A}A \Rightarrow ababa\underline{A} \Rightarrow ababaa.\end{aligned}$$

A chaque étape, nous avons souligné le symbole non terminal substitué. Ainsi, le mot *ababaa* est obtenu à partir de S par une dérivation de longueur 8. La suite des règles appliquées donnant lieu à un mot donné n'est pas nécessairement unique. En effet, on peut générer le mot *ababaa* à partir du symbole initial S de diverses façons :

S	$\Rightarrow$	$\underline{A}A$	S	$\Rightarrow$	$A\underline{A}$	S	$\Rightarrow$	$\underline{A}A$
		$\underline{A}AAA$			$\underline{A}a$			$a\underline{A}$
		$a\underline{A}AA$			$AA\underline{A}a$			$aAA\underline{A}$
		$ab\underline{A}AA$			$AAb\underline{A}a$			$a\underline{A}Aa$
		$aba\underline{A}A$			$A\underline{A}baa$			$abA\underline{A}a$
		$abab\underline{A}A$			$Ab\underline{A}baa$			$ab\underline{A}bAa$
		$ababa\underline{A}$			$\underline{A}ababaa$			$abab\underline{A}a$
		$ababaa$			$ababaa$			$ababaa$

Au vu de cet exemple, nous introduisons la notion de dérivation la plus à gauche.

Définition VI.1.4. Soient $G = (V, \Sigma, P, S)$ une grammaire hors contexte, $w \in L(G)$ et

$$S \Rightarrow x_1\,\underline{A_1}\,y_1 \Rightarrow x_2\,\underline{A_2}\,y_2 \Rightarrow \cdots \Rightarrow x_n\,\underline{A_n}\,y_n \Rightarrow w$$

une dérivation de longueur $n+1$ telle que $x_i \in \Sigma^*$, $A_i \in V$ et $y_i \in (V \cup \Sigma)^*$, pour tout $i \in \{1, \ldots, n\}$. Alors, cette dérivation est une *dérivation à gauche*. Cela signifie qu'à chaque étape, on a substitué la variable la plus à gauche. On définit de manière semblable une *dérivation à droite*. Comme le montre une fois encore l'exemple précédent, pour une grammaire G fixée, un mot appartenant à $L(G)$ peut avoir plus d'une dérivation à gauche[2]. Il est aussi clair que si un mot appartient à $L(G)$, il possède au moins une dérivation à gauche. Si tout mot de $L(G)$ possède exactement une dérivation à gauche, alors la grammaire G est qualifiée de *non ambiguë*. Un langage algébrique est *non ambigu* s'il existe une grammaire non ambiguë qui le génère. Nous verrons à la section 3 en quoi le caractère non ambigu est important d'un point de vue pratique[3].

[2]On peut montrer que le nombre de dérivations à gauche d'un mot de $L(G)$ est égal au nombre de dérivations à droite permettant d'obtenir ce même mot. Ainsi, il est équivalent de définir une notion, comme le caractère non ambigu, sur le nombre de dérivations à gauche ou à droite.

[3]Pour plus d'information à propos de l'utilisation des grammaires dans l'écriture d'un compilateur, voir par exemple la page `http://dinosaur.compilertools.net/` où l'on présente les outils `Lex`, `Yacc`, `Flex` et `Bison`

Considérons encore deux autres exemples de grammaires.

Exemple VI.1.5. La grammaire ci-dessous génère exactement le langage $\{a^n b^n \mid n \in \mathbb{N}\}$. Considérons $G = (V, \Sigma, P, S)$ où $V = \{S\}$, $\Sigma = \{a, b\}$, et les productions de G données par

$$S \;\to\; aSb \mid \varepsilon.$$

Ce langage est trivialement non ambigu. En effet, pour chaque mot w du langage $L(G)$ il existe une seule suite de règles de G permettant d'obtenir w à partir de S.

Exemple VI.1.6 (Langage de Dyck). On considère l'alphabet

$$\Sigma = \{a_1, \overline{a_1}, \ldots, a_n, \overline{a_n}\},$$

la grammaire $G = (V, \Sigma, P, S)$ où $V = \{S, T\}$ et les productions de P données par

$$\begin{aligned} S &\to ST \mid \varepsilon \\ T &\to a_1\ S\ \overline{a_1} \mid \cdots \mid a_n\ S\ \overline{a_n}\ . \end{aligned}$$

Le langage généré par la grammaire G s'appelle le *n-ième langage de Dyck* et se note $\mathfrak{D}_n$. Il est facile de voir qu'il s'agit exactement du langage formé des mots "bien parenthèsés" lorsqu'on dispose de n niveaux de parenthèsage (la j-ième parenthèse ouvrante étant symbolisée par a_j et la j-ième parenthèse fermante correspondante par $\overline{a_j}$). En guise d'illustration, considérons l'alphabet $\Sigma = \{(\,), [\,]\}$ formé de parenthèses et de crochets et les productions

$$\begin{aligned} S &\to ST \mid \varepsilon \\ T &\to (\ S\ ) \mid [\ S\ ]. \end{aligned}$$

On obtient par exemple les mots suivants

$$\underline{S} \Rightarrow S\underline{T} \Rightarrow \underline{S}(S) \Rightarrow (\underline{S}) \Rightarrow (\,),$$

$$\begin{aligned} \underline{S} &\Rightarrow S\underline{T} \Rightarrow \underline{S}(S) \Rightarrow ST(\underline{S}) \Rightarrow ST(\,) \\ &\Rightarrow S\underline{T}(\,) \Rightarrow S[S](\,) \Rightarrow S[\underline{S}](\,) \Rightarrow \underline{S}[\,](\,) \Rightarrow [\,](\,), \end{aligned}$$

$$\begin{aligned} \underline{S} &\Rightarrow S\underline{T} \Rightarrow S(\underline{S}) \Rightarrow S(S\underline{T}) \Rightarrow S(S(\underline{S})) \\ &\Rightarrow S(S(\underline{S}T)) \Rightarrow S(S(S\underline{T}T)) \Rightarrow^* (((\,)(\,))). \end{aligned}$$

Noter que, dans un langage de Dyck, il n'y a pas de préséance sur l'ordre des différentes parenthèses. Ainsi, les mots $[(\,)]$ et $([\,])$ sont tous deux valides.

Dans le cas de l'alphabet $\Sigma = \{a, \overline{a}\}$, on peut encore montrer qu'un mot w appartient au premier langage de Dyck $\mathfrak{D}_1$ si et seulement si les deux conditions suivantes sont satisfaites

- pour tout $i \in \{1, \ldots, n\}$, $|w|_a = |w|_{\overline{a}}$
- pour tout préfixe u de w, $|u|_a \geq |u|_{\overline{a}}$.

2. Arbres d'analyse

Dans cette section, nous allons montrer qu'à une dérivation correspond un arbre, appelé arbre d'analyse, et réciproquement, à tout arbre d'analyse correspond au moins une dérivation. Nous supposerons[4] qu'aucun second membre des productions des grammaires rencontrées n'est égal à ε.

Pour rappel, en théorie des graphes, un arbre est un graphe connexe sans cycle. Par commodité, nous allons préférer une définition récursive des arbres. Soit E, un ensemble fini dont les éléments sont appelés *noeuds*. Les arbres de hauteur 0 sont les éléments e de E. On les note $(e, \emptyset)$ et e est la racine de l'arbre. Si $e \in E$ et $\mathcal{A}_1, \ldots, \mathcal{A}_n$ sont des arbres respectivement de hauteur h_i et de racine e_i, $i = 1, \ldots, n$, alors, en connectant e aux différents e_i, on définit $(e, (\mathcal{A}_1, \ldots, \mathcal{A}_n))$ comme étant un arbre de racine e, de hauteur $1 + \sup_i h_i$ et de sous-arbres $\mathcal{A}_1, \ldots, \mathcal{A}_n$. Dans cette définition, le n-uple $(\mathcal{A}_1, \ldots, \mathcal{A}_n)$ est ordonné. Ainsi, si μ est une permutation distincte de l'identité, $(e, (\mathcal{A}_1, \ldots, \mathcal{A}_n)) \neq (e, (\mathcal{A}_{\mu(1)}, \ldots, \mathcal{A}_{\mu(n)}))$. On dit que les noeuds $e_1, \ldots, e_n$ sont les *fils* de e (ou que e est le *père* des e_i). Si $f \in E$ appartient à un des sous-arbres $\mathcal{A}_i$, alors f est un *descendant* de e (ou e est un *ancêtre* de f). En particulier, la racine d'un arbre de hauteur 0 n'a pas de fils (ce qui explique la notation $(e, \emptyset)$). Un arbre de racine e ayant trois sous-arbres $\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3$ est représenté schématiquement à la figure VI.1. La hauteur de

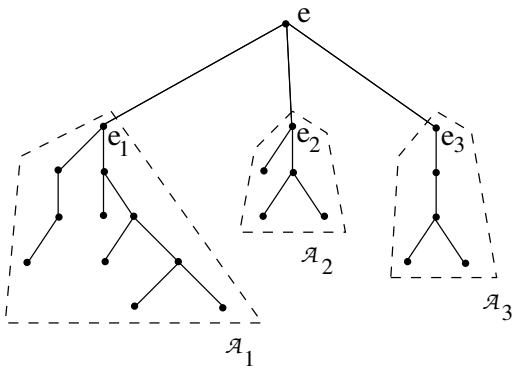


FIGURE VI.1. Un arbre.

l'arbre $\mathcal{A}$ (resp. $\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3$) est 5 (resp. $4, 2, 3$).

Définition VI.2.1. Soit $G = (V, \Sigma, P, S)$ une grammaire hors contexte. Pour tout $A \in V \cup \Sigma$, $(A, \emptyset)$ est un *arbre d'analyse* de G. Si $A \to A_1 \cdots A_n$ est une production de G, $A_i \in V \cup \Sigma$, et si $\mathcal{A}_1, \ldots, \mathcal{A}_n$ sont des arbres d'analyse de G de racine respective $A_1, \ldots, A_n$, alors $(A, (\mathcal{A}_1, \ldots, \mathcal{A}_n))$ est encore un arbre d'analyse de G. Cette définition est récursive et permet de construire de proche en proche[5] les arbres d'analyse de G.

[4] Nous verrons plus tard qu'il est toujours possible de se ramener à une telle situation.

[5] par hauteur croissante.

Exemple VI.2.2. Poursuivons l'exemple VI.1.3. Voici quelques arbres d'analyse de G représentés à la figure VI.2.

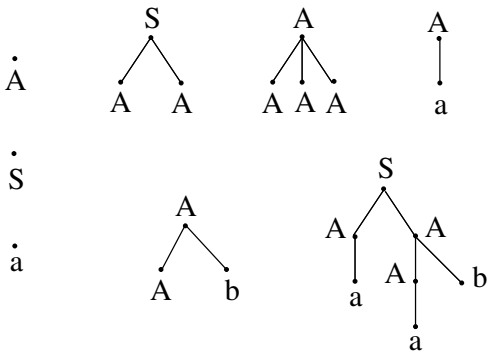


FIGURE VI.2. Des arbres d'analyse.

Définition VI.2.3. Soit $\mathcal{A}$ un arbre d'analyse de G. Le *fruit* de $\mathcal{A}$, noté $\mathcal{F}(\mathcal{A})$, est un mot défini récursivement. Si l'arbre est de hauteur nulle, i.e., si $\mathcal{A} = (B, \emptyset)$, $B \in V \cup \Sigma$, alors $\mathcal{F}(\mathcal{A}) = B$. Sinon, il existe des arbres d'analyse $\mathcal{A}_1, \ldots, \mathcal{A}_n$ tels que $\mathcal{A} = (B, (\mathcal{A}_1, \ldots, \mathcal{A}_n))$. Dans ce cas, on pose

$$\mathcal{F}(\mathcal{A}) = \mathcal{F}(\mathcal{A}_1) \cdots \mathcal{F}(\mathcal{A}_n).$$

L'opération envisagée ici est bien évidemment la concaténation des fruits respectifs des différents sous-arbres.

Exemple VI.2.4. Si on reprend les arbres d'analyse de G représentés à la figure VI.2, les fruits de ces arbres sont respectivement

$$A, S, a, AA, AAA, a, Ab, aab.$$

Il est clair qu'à une dérivation correspond un arbre d'analyse. On construit cet arbre de proche en proche à partir de l'arbre d'analyse $(S, \emptyset)$. A chaque fois qu'une variable est substituée, on greffe le sous-arbre correspondant à la règle qui a été appliquée. Considérons un exemple.

Exemple VI.2.5. Poursuivons l'exemple VI.1.3. Nous avions obtenu la dérivation suivante du mot *ababaa*.

$$\begin{aligned} \underline{S} &\Rightarrow \underline{A}A \Rightarrow a\underline{A} \Rightarrow a\underline{A}AA \\ &\Rightarrow ab\underline{A}AA \Rightarrow aba\underline{A}A \Rightarrow abab\underline{A}A \Rightarrow ababa\underline{A} \Rightarrow ababaa. \end{aligned}$$

Pour chacune des productions considérées, on obtient de proche en proche l'arbre d'analyse représenté à la figure VI.3. Lorsqu'une production est appliquée à une variable donnée, on ajoute le sous-arbre correspondant à l'arbre d'analyse déjà obtenu. A chaque étape, le fruit de l'arbre est modifié en conséquence pour obtenir finalement un arbre de racine S et de fruit *ababaa*.

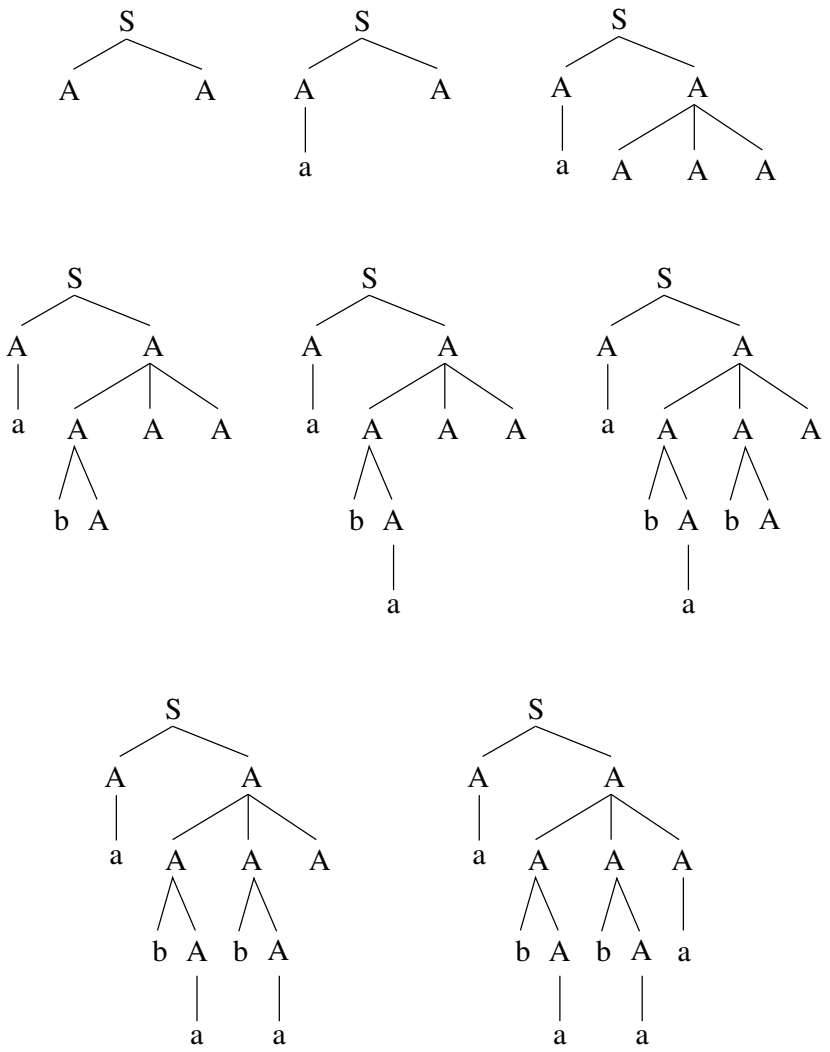


FIGURE VI.3. Arbres d'analyse provenant de dérivations.

Réciproquement, à un arbre d'analyse de G de sommet S et de fruit w appartenant à Σ^*, il correspond[6] au moins une suite de règles produisant w à partir de S. Dans cet arbre, lorsque deux symboles non terminaux se trouvent au même niveau[7], il n'est pas possible de savoir quelle règle de dérivation est appliquée avant quelle autre. Par conséquent, il n'y a pas unicité dans l'ordre d'application des règles de la grammaire. Par exemple, le dernier arbre de dérivation obtenu à la figure VI.3 et ayant *ababaa* pour

[6]Cette correspondance existe pour tout arbre d'analyse, pas seulement ceux de racine S et de fruit terminal. En effet, à tout arbre de racine $A \in V$ et fruit $w \in (V \cup \Sigma)^*$, il correspond une suite de règles produisant w à partir de A.

[7]Dans un arbre, le *niveau* d'un noeud est la longueur du chemin menant de la racine à ce noeud.

fruit correspond également à la suite de règles

$$\begin{aligned} \underline{S} &\Rightarrow A\underline{A} \Rightarrow AA\underline{A}A \Rightarrow AAbA\underline{A} \Rightarrow AAb\underline{A}a \\ &\Rightarrow \underline{A}Abaa \Rightarrow a\underline{A}baa \Rightarrow ab\underline{A}baa \Rightarrow ababaa. \end{aligned}$$

Néanmoins, à un arbre d'analyse donné, il correspond une seule dérivation à gauche. Cela revient à parcourir l'arbre (de manière récursive) comme suit

- Si l'arbre est réduit à la racine $\longrightarrow$ fin du parcours.
- Sinon, $\mathcal{A} = (B, (\mathcal{A}_1, \ldots, \mathcal{A}_n))$ et parcourir, dans l'ordre, les sous-arbres $\mathcal{A}_1, \ldots, \mathcal{A}_n$.

Le parcours $\mathfrak{P}$ dans l'arbre fournit la suite de règles à appliquer pour obtenir la dérivation à gauche.

Exemple VI.2.6. Considérons l'arbre représenté à la figure VI.4. A cet

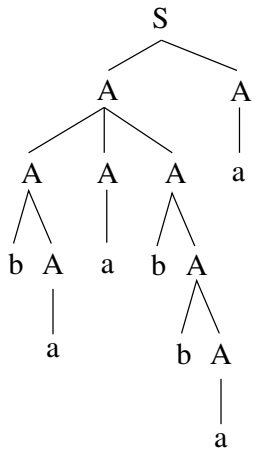


FIGURE VI.4. Un arbre d'analyse.

arbre correspond l'unique dérivation à gauche

$$\begin{aligned} S &\Rightarrow \underline{A}A \Rightarrow \underline{A}AAA \Rightarrow b\underline{A}AAA \Rightarrow ba\underline{A}AA \Rightarrow baa\underline{A}A \Rightarrow baab\underline{A}A \\ &\Rightarrow baabb\underline{A}A \Rightarrow babba\underline{A} \Rightarrow babbaa. \end{aligned}$$

Remarque VI.2.7. Une adaptation immédiate permet d'obtenir la dérivation à droite associée à un arbre. Si $\mathcal{A} = (B, (\mathcal{A}_1, \ldots, \mathcal{A}_n))$ n'est pas réduit à une racine, parcourir, dans l'ordre, les sous-arbres $\mathcal{A}_n, \ldots, \mathcal{A}_1$.

3. Une illustration de l'ambiguité

Considérons la grammaire $G = (N, \Sigma, P, E)$ où $N = \{E, N, C\}$ (on utilise ici les lettres E, N et C comme dans Expression, Nombre et Chiffre),

$$\Sigma = \{+, -, *, /, (,), 0, 1, \ldots, 9\}$$

et où les règles sont

$$\begin{array}{lcl} E & \to & (E) \mid E+E \mid E-E \mid E*E \mid E/E \mid \mathrm{N} \\ N & \to & C \mid NC \\ C & \to & 0 \mid 1 \mid \cdots \mid 9. \end{array}$$

Cette grammaire génère des expressions arithmétiques élémentaires (on s'autorise de plus, pour ne pas alourdir l'exposé, à écrire des nombres commençant par 0).

Considérons le mot "$1-2+3$" appartenant au langage généré par cette grammaire. Ce mot est obtenu par la dérivation à gauche

$$\begin{array}{rcl} \underline{E} & \Rightarrow & \underline{E}+E \Rightarrow \underline{E}-E+E \Rightarrow \underline{N}-E+E \Rightarrow \underline{C}-E+E \Rightarrow 1-\underline{E}+E \\ & \Rightarrow & 1-\underline{N}+E \Rightarrow 1-\underline{C}+E \Rightarrow 1-2+\underline{E} \Rightarrow 1-2+\underline{N} \Rightarrow 1-2+\underline{C} \\ & \Rightarrow & 1-2+3. \end{array}$$

A cette dérivation correspond l'arbre d'analyse représenté à la figure VI.5. Le mot "$1-2+3$" est aussi obtenu par la dérivation à gauche

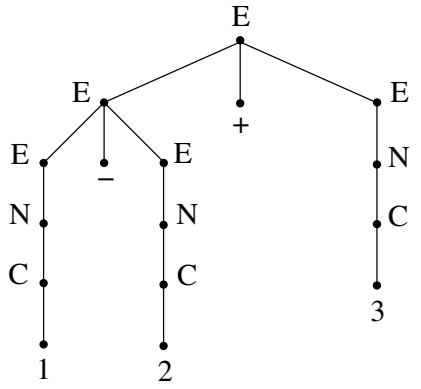


FIGURE VI.5. Un arbre d'analyse pour $1-2+3$.

$$\begin{array}{rcl} \underline{E} & \Rightarrow & \underline{E}-E \Rightarrow \underline{N}-E \Rightarrow \underline{C}-E \Rightarrow 1-\underline{E} \Rightarrow 1-\underline{E}+E \\ & \Rightarrow & 1-\underline{N}+E \Rightarrow 1-\underline{C}+E \Rightarrow 1-2+\underline{E} \Rightarrow 1-2+\underline{N} \Rightarrow 1-2+\underline{C} \\ & \Rightarrow & 1-2+3. \end{array}$$

A cette dérivation correspond l'arbre d'analyse représenté à la figure VI.6. Lorsqu'on dispose d'un arbre d'analyse[8] (que ce soit celui de la figure VI.5 ou celui de la figure VI.6), le parcours récursif $\mathfrak{P}$ de cet arbre où l'on considère à chaque fois, en premier lieu, le sous-arbre le plus à gauche, permet d'évaluer

[8]En général, lors de la phase de compilation d'un programme, ou dans le cas plus simple qui nous intéresse ici, l'interprétation d'une formule, la première étape confiée à l'*analyseur* est de déterminer un arbre d'analyse. Une fois l'arbre d'analyse connu, on peut spécifier le sens à donner aux différents noeuds. La sémantique est particulièrement simple dans le cadre décrit ici puisqu'il ne s'agit que d'expressions arithmétiques. Pour un programme à compiler, on pourrait imaginer devoir réaliser l'allocation de mémoire, l'adressage de variables, etc...

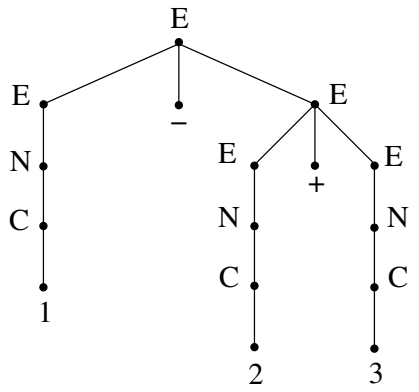


FIGURE VI.6. Un arbre d'analyse pour $1 - 2 + 3$.

les expressions envisagées. Si on considère l'arbre de la figure VI.5, le sous-arbre de gauche a pour valeur $1 - 2$, celui de droite 3 et donc, la valeur assignée à l'arbre est

$$(1 - 2) + 3 = 2.$$

Par contre, si on considère à présent l'arbre de la figure VI.6, le sous-arbre de gauche a pour valeur 1 et celui de droite $2+3$. Dès lors, la valeur assignée est cette fois

$$1 - (2 + 3) = -4.$$

Remarquons qu'il s'agit une fois encore d'une dérivation à gauche,

$$\underline{E} \;\Rightarrow\; \underline{E} - E \Rightarrow^* 1 - \underline{E} \Rightarrow 1 - E + E \Rightarrow^* 1 - 2 + 3.$$

Ainsi, suivant l'arbre choisi, les groupements de termes sont réalisés en partant de la gauche ou de la droite et la valeur assignée n'est pas nécessairement la valeur attendue.

Si les opérateurs n'ont pas la même préséance, la grammaire proposée peut regrouper un opérateur de faible préséance avant un opérateur de préséance plus élevée. En effet, considérons le mot "2+3∗5". A ce mot, il correspond les arbres d'analyse représentés à la figure VI.7 . Ainsi, l'évaluation de l'arbre de gauche fournit la valeur $(2 + 3) * 5$ alors que pour l'arbre de droite, on trouve $2 + (3 * 5)$.

Cet exemple montre bien le problème que pose en pratique l'utilisation d'une grammaire ambiguë. En effet, le compilateur ou l'interpréteur[9] n'a pas les moyens de déterminer quel arbre d'analyse permet d'assigner une valeur correcte à l'expression envisagée. Ainsi, lors de la spécification d'un compilateur, il faut veiller à employer une grammaire non ambiguë.

Revenons sur le problème des expressions arithmétiques. L'écueil principal de la grammaire présentée ci-dessus est qu'elle ne tient pas compte de

[9]Le rôle d'un compilateur est de transformer un code "source" en un autre code. Par exemple, transformer un texte codant un programme écrit en C en un code machine intelligible par le processeur ou le système d'exploitation ou encore, transformer un texte comprenant des instructions LaTeX en un fichier ".dvi" affichable à l'écran.

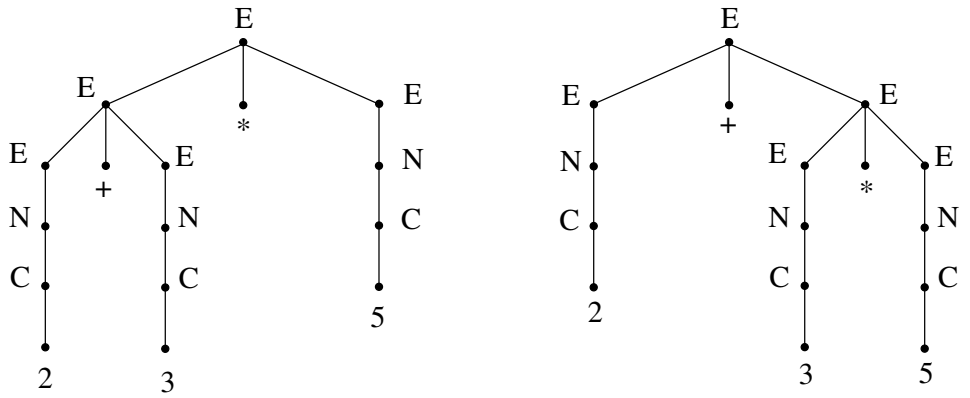


FIGURE VI.7. Deux arbres d'analyse pour $2 + 3 * 5$.

l'ordre de préséance des opérations à effectuer. Pour y remédier et obtenir une grammaire hors contexte non ambiguë, nous proposons (sans preuve) la grammaire suivante. Les symboles non terminaux sont E, T, F, N, C où T et F sont employés pour rappeler les mots Terme et Facteur. Les règles sont

$$\begin{array}{rcl} E & \to & E + T \mid E - T \mid T \\ T & \to & T * F \mid T/F \mid F \\ F & \to & (E) \mid N \\ N & \to & C \mid NC \\ C & \to & 0 \mid 1 \mid \cdots \mid 9. \end{array}$$

La distinction faite ici entre expressions, termes et facteurs force le groupement correct des opérateurs à différents niveaux de préséance. La figure VI.8 reprend les arbres d'analyse des expressions $1 - 2 + 3$ et $2 + 3 * 5$.

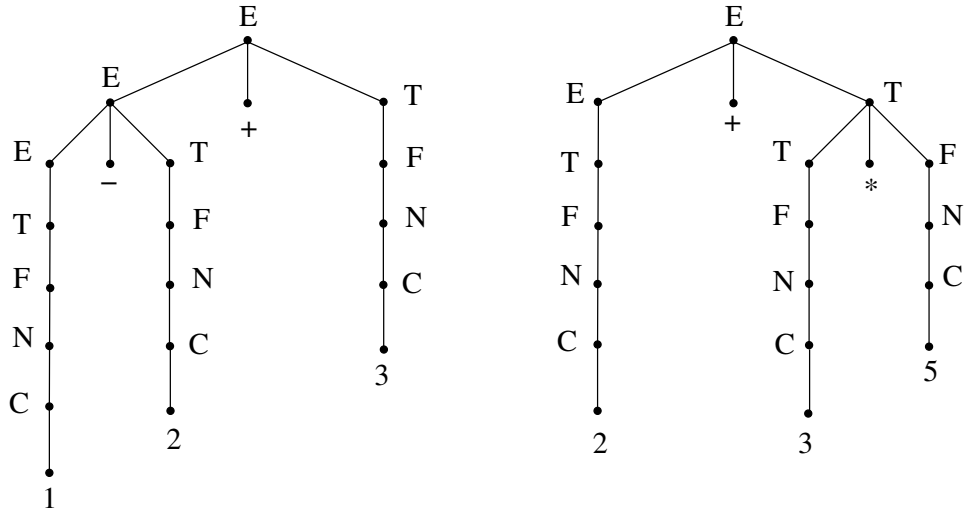


FIGURE VI.8. Arbres d'analyse de "$1 - 2 + 3$" et "$2 + 3 * 5$" pour une grammaire non ambiguë.

Signalons pour conclure, qu'on peut aisément enrichir cette dernière grammaire d'autres opérateurs comme l'exponentiation ou encore les fonctions trigonométriques, etc...

4. Grammaires et langages réguliers

Le but de cette section est de montrer que l'ensemble des langages réguliers est un sous-ensemble (strict[10]) de l'ensemble des langages algébriques.

Pour ce faire, nous allons utiliser la proposition I.3.6 en montrant que la famille des langages algébriques contient le langage vide, les langages $\{\sigma\}$, $\sigma \in \Sigma$, et est stable pour l'union, la concaténation et l'étoile de Kleene.

Remarque VI.4.1. La grammaire $G = (\{S\}, \Sigma, P, S)$ où l'unique règle est $S \to \sigma$, $\sigma \in \Sigma$, génère le langage $\{\sigma\}$. De même, si $P = \emptyset$, le langage généré est $\emptyset$.

Proposition VI.4.2. *L'ensemble des langages algébriques est stable pour l'union.*

Démonstration. Soit $G_1 = (V_1, \Sigma, P_1, S_1)$ (resp. $G_2 = (V_2, \Sigma, P_2, S_2)$) une grammaire générant L_1 (resp. L_2). On peut supposer que $V_1 \cap V_2 = \emptyset$ et que $S \notin V_1 \cup V_2$. La grammaire $G = (\{S\} \cup V_1 \cup V_2, \Sigma, P, S)$ où P contient $P_1 \cup P_2$ et la règle

$$S \to S_1 \,|\, S_2,$$

génère exactement $L_1 \cup L_2$. Les justifications sont laissées au lecteur à titre d'exercice.

■

Proposition VI.4.3. *L'ensemble des langages algébriques est stable pour la concaténation.*

Démonstration. Avec les mêmes notations que dans la preuve précédente, il suffit de considérer la règle supplémentaire

$$S \to S_1 S_2$$

pour générer le langage $L_1 L_2$.

■

Proposition VI.4.4. *L'ensemble des langages algébriques est stable pour l'étoile de Kleene.*

Démonstration. Encore une fois en utilisant les mêmes notations, il suffit de considérer la grammaire $G = (\{S\} \cup V_1, \Sigma, P, S)$ où P contient P_1 et la règle supplémentaire

$$S \to S S_1 \,|\, \varepsilon$$

pour générer le langage L_1^*.

■

[10] Nous savons déjà que $\{a^n b^n \mid n \in \mathbb{N}\}$ est algébrique et non régulier.

Corollaire VI.4.5. *L'ensemble des langages réguliers sur un alphabet fini est un sous-ensemble de l'ensemble des langages algébriques.*

Démonstration. Cela résulte directement de la remarque VI.4.1, des trois propositions précédentes et de la proposition I.3.6.

■

Il est possible de particulariser les grammaires hors contexte en spécifiant la forme des seconds membres de leurs productions. On peut même spécifier des grammaires générant exactement les langages réguliers. De telles grammaires sont appelées grammaires régulières.

Définition VI.4.6. Une grammaire hors contexte $G = (V, \Sigma, P, S)$ est *régulière* (*à gauche*) si toute production de G possède une des trois formes suivantes :

- $A \to a$
- $A \to Ba$
- $A \to \varepsilon$

où A, B appartiennent à V et a à Σ. De manière équivalente, on se convainc facilement[11] qu'une grammaire est régulière à gauche si les seconds membres de ses productions appartiennent tous à $\Sigma^* \cup V\Sigma^*$.

Une grammaire hors contexte est *régulière* (*à droite*) si toute production de G possède une des trois formes suivantes :

- $A \to a$
- $A \to aB$
- $A \to \varepsilon$.

De même, une grammaire est régulière à droite si les seconds membres de ses productions appartiennent tous à $\Sigma^* \cup \Sigma^* V$.

Exemple VI.4.7. Soit la grammaire $G = (V, \Sigma, P, S)$ où $V = \{S, A, B\}$, $\Sigma = \{a, b\}$ et où les productions sont

$$\begin{array}{rcl} S & \to & aB \mid \varepsilon \\ B & \to & bS \mid bA \\ A & \to & aA \mid \varepsilon. \end{array}$$

Il est facile de voir que le langage généré par G est exactement

$$\{\varepsilon\} \cup (ab)^* ab (a)^*$$

qui est régulier.

Nous donnons à titre indicatif et sans démonstration le résultat suivant.

Proposition VI.4.8. *Un langage est régulier si et seulement si il est généré par une grammaire régulière à gauche (resp. à droite).*

[11] Par exemple, on peut remplacer une règle de la forme $A \to Bab$ par les règles $A \to B'b$ et $B' \to Ba$.

Remarque VI.4.9. Signalons que les grammaires régulières sont des cas particuliers de grammaire dont les seconds membres des productions appartiennent tous à $\Sigma^* \cup \Sigma^* V \Sigma^*$, i.e., les seconds membres contiennent au plus une variable. Les grammaires possédant une telle propriété sont dites *linéaires*.

5. A propos de la hiérarchie de Chomsky

Dans ce cours, nous nous limitons volontairement à l'étude des langages réguliers et des langages algébriques. A titre indicatif, nous présentons d'autres types de grammaires plus générales permettant d'obtenir de nouvelles classes plus larges de langages. Ces différents types ayant été introduits par Noam Chomsky, il est de coutume de parler de la hiérarchie de Chomsky.

Définition VI.5.1. Une grammaire $G = (V, \Sigma, P, S)$ de type **0**, ou *grammaire non restrictive*, est la forme la plus générale de grammaire. Les alphabets V et Σ et le symbole initial S sont définis comme dans le cas des grammaires hors contexte. Une production de la forme $u \to v$ précise qu'une occurence du mot $u \neq \varepsilon$ peut être remplacée par v, avec $u, v \in (V \cup \Sigma)^*$.

Remarque VI.5.2. Les grammaires hors contexte sont donc des cas particuliers de grammaire non restrictive. En effet, dans une grammaire hors contexte, le premier membre des règles est réduit à des mots d'une lettre sur l'alphabet V.

Exemple VI.5.3. La grammaire non restrictive $G = (V, \Sigma, P, S)$ telle que $V = \{S, A, C\}$, $\Sigma = \{a, b, c\}$ et dont les règles sont données par

$$\begin{array}{rcl} S & \to & aAbc \mid \varepsilon \\ A & \to & aAbC \mid \varepsilon \\ Cb & \to & bC \\ Cc & \to & cc \end{array}$$

génère le langage $\{a^n b^n c^n \mid n \in \mathbb{N}\}$. En effet,

$$\begin{array}{rcl} S & \Rightarrow & aAbc \Rightarrow aaAbCbc \Rightarrow^* a(a)^i A(bC)^i bc \Rightarrow^* a(a)^i (bC)^i bc = (a)^{i+1} b (Cb)^i c \\ & \Rightarrow^* & (a)^{i+1}(b)^{i+1} C^i c \Rightarrow^* (a)^{i+1}(b)^{i+1}(c)^{i+1}. \end{array}$$

Remarque VI.5.4. On peut montrer qu'un langage L est généré par une grammaire non restrictive si et seulement si L est récursivement énumérable[12] (i.e., accepté par une machine de Turing).

Dans la hiérarchie de Chomsky, entre les grammaires hors contexte et les grammaires non restrictives, il existe encore un type de grammaire.

Définition VI.5.5. Une grammaire non restrictive $G = (V, \Sigma, P, S)$ est de type **1**, aussi appelée *grammaire dépendant du contexte* [*Context-sensitive grammar*], si toutes les productions $u \to v$ de G satisfont

[12] cf. le cours de calculabilité.

- $u, v \in (V \cup \Sigma)^* \setminus \{\varepsilon\}$
- $|u| \leq |v|$.

Si une grammaire satisfait cette dernière condition, on parle parfois de *grammaire non contractante* ou *monotone* car la longueur des mots produits croît à chaque application d'une nouvelle règle. On autorise de plus une unique règle de la forme $S \to \varepsilon$.

Remarque VI.5.6. Une définition équivalente de grammaire dépendant du contexte $G = (V, \Sigma, P, S)$ est de spécifier les productions de P sous la forme

$$\alpha N \beta \to \alpha v \beta$$

où $\alpha, \beta \in (V \cup \Sigma)^*$, $N \in V$, $v \in (V \cup \Sigma)^* \setminus \{\varepsilon\}$. De cette façon, on met mieux en évidence le contexte dans lequel se trouve la variable N qui peut avoir des effets différents suivant les éléments qui l'entourent. On pourrait par exemple imaginer deux règles distinctes

$$\alpha_1 N \beta_1 \to \alpha_1 v_1 \beta_1 \text{ et } \alpha_2 N \beta_2 \to \alpha_2 v_2 \beta_2$$

avec $v_1 \neq v_2$ si $(\alpha_1, \beta_1) \neq (\alpha_2, \beta_2)$.

Exemple VI.5.7. La grammaire présentée dans l'exemple VI.5.3 n'est pas monotone à cause des productions $S \to \varepsilon$ et $A \to \varepsilon$. Il est facile de vérifier que la grammaire suivante génère encore le même langage,

$$\begin{array}{rcl} S & \to & aAbc \mid abc \\ A & \to & aAbC \mid abC \\ Cb & \to & bC \\ Cc & \to & cc. \end{array}$$

Cette dernière est bien une grammaire monotone dépendant du contexte.

Remarque VI.5.8. Une fois encore, on peut montrer que tout langage généré par une grammaire dépendant du contexte est récursif[13] (i.e., décidé par une machine de Turing). Plus précisément, les langages générés par une grammaire dépendant du contexte sont exactement les langages décidés par les machines de Turing dont la mémoire disponible est bornée de manière linéaire par la taille des données. En d'autres termes, on ne s'autorise pas un ruban de longueur arbitraire mais à chaque exécution, la longueur du ruban disponible est proportionnelle à la taille des données fournies à la machine de Turing.

Le tableau suivant récapitule les divers faits énoncés dans cette section.

[13] cf. le cours de calculabilité.

	générateur	langage	accepteur
0	grammaire non restrictive	récursivement énumérable	machine de Turing
1	grammaire dépendant du contexte	dépendant du contexte	machine de Turing à mémoire linéaire
2	grammaire hors contexte	hors contexte	automates à pile
3	expression régulière	régulier	AFD

Les automates à pile, accepteurs des langages algébriques, seront présentés dans une prochaine section.

Remarque VI.5.9. Au vu du tableau précédent, on dispose des inclusions suivantes

$$\boxed{\text{Reg} \subset \text{Lin} \subset \text{Alg} \subset \text{DP} \subset \text{RE}}$$

où les différentes abréviations désignent respectivement l'ensemble des langages réguliers, linéaires (cf. remarque VI.4.9), algébriques, dépendants du contexte et récursivement énumérables.

Remarque VI.5.10. Le lecteur attentif pourrait émettre une objection quant à la définition de grammaire dépendant du contexte où l'on interdit la production du symbole ε, alors que cette restriction n'est pas présente pour les grammaires hors contexte (qui sont cependant un cas particulier de grammaires de type 1). En fait, comme nous allons le voir dans la section suivante, on peut aussi se débarasser des règles $A \to \varepsilon$ dans les grammaires hors contexte. De plus, si une grammaire dépendant du contexte doit effectivement pouvoir générer le mot vide, on se permet d'utiliser une unique règle $S \to \varepsilon$.

6. Formes normales

Lorsqu'on s'intéresse à un langage algébrique L donné, la grammaire générant L n'est pas nécessairement unique. Ainsi, on peut désirer avoir à sa disposition une grammaire dont les règles possèdent une forme particulière. Lorsqu'on impose certaines restrictions sur les seconds membres des productions, on parle de grammaire mise sous *forme normale*. Nous montrons dans cette section que de telles simplifications sont toujours possibles. Rappelons que deux grammaires sont équivalentes si elles génèrent le même langage.

6.1. Elimination des règles $A \to \varepsilon$.

Exemple VI.6.1. Dans une dérivation produisant un mot terminal, il se peut qu'apparaissent des variables ne générant aucun symbole terminal. Ces variables sont éliminées grâce à des règles de la forme $A \to \varepsilon$ que nous appelerons *ε-production*. Un tel phénomène fait grossir inutilement la longueur des mots intermédiaires produits. Par exemple, considérons les règles

$$\begin{aligned} S &\to SaB \mid aB \\ B &\to bB \mid \varepsilon. \end{aligned}$$

La dérivation à gauche générant le mot aaa génère trois B qui seront chacun éliminés par l'application de la règle $B \to \varepsilon$. Ainsi, on a

$$S \Rightarrow SaB \Rightarrow SaBaB \Rightarrow aBaBaB \Rightarrow aaBaB \Rightarrow aaaB \Rightarrow aaa.$$

Définition VI.6.2. On appelle variable *effaçable* toute variable A telle que

$$A \Rightarrow^* \varepsilon.$$

Si une grammaire ne contient aucune variable effaçable, alors $u \Rightarrow v$ entraîne $|u| \leq |v|$. On est dès lors en présence d'une grammaire monotone (appliquer une règle ne peut faire diminuer la longueur du mot obtenu).

Nous présentons maintenant un algorithme[14] permettant de détecter les variables effaçables. Posons

$$E_0 = \{A \in V \mid A \to \varepsilon \in P\}.$$

Si $E_0 = \emptyset$, l'algorithme s'achève et la grammaire ne possède aucune variable effaçable. Sinon, pour $i \geq 0$, on définit

$$E_{i+1} = E_i \cup \{A \in V \mid \exists w \in E_i^* : A \to w \in P\}.$$

Puisque V est fini, la suite des E_i se stabilise. Il est clair que le plus grand E_i apparaissant dans cette suite est l'ensemble des variables effaçables. Une condition d'arrêt pour l'algorithme revient à tester l'égalité de E_i et E_{i+1}.

Proposition VI.6.3. *Soit $G = (V, \Sigma, P, S)$ une grammaire hors contexte. Il existe une grammaire $G' = (V', \Sigma', P', S')$ que l'on peut construire effectivement*[15] *telle que*

- *G et G' sont équivalentes,*
- *S' n'apparaît dans aucun second membre des productions de G',*
- *si ε appartient à $L(G) = L(G')$, alors la seule variable effaçable est S'. Sinon, G' ne contient aucune variable effaçable.*

Démonstration. Si S apparaît dans un second membre des productions de G, on introduit une nouvelle variable S'. On définit V' comme étant $V \cup \{S'\}$ et pour définir P', on ajoute aux règles de P, la règle $S' \to S$. De cette manière, le nouveau symbole initial S' n'apparaît dans aucun second membre des productions. De plus, si $S \Rightarrow_G^* w$, alors $S' \Rightarrow_{G'} S \Rightarrow_{G'}^* w$.

Au vu de l'algorithme précédent, on sait déterminer de manière effective l'ensemble E des variables effaçables de G'. Toute règle de G' de la forme

$$A \to w_1 A_1 w_2 A_2 \cdots w_n A_n w_{n+1}$$

[14] ayant une approche "bottom-up".

[15] Cela signifie qu'il ne s'agit pas d'un théorème existentiel mais bien d'un théorème constructif. La preuve fournit une démarche, un algorithme, permettant d'obtenir la grammaire proposée.

avec $A \in V$, $A_1, \ldots, A_n \in E$, $w_1, \ldots, w_{n+1} \in ((V \cup \Sigma) \setminus E)^*$ est remplacée par les règles

$$A \to w_1 x_1 w_2 x_2 \cdots w_n x_n w_{n+1}$$

où chaque x_i peut prendre la valeur A_i ou ε. Une règle est donc remplacée par au plus 2^n nouvelles règles. Il est clair que cette modification n'altère pas le langage généré puisqu'on a éventuellement enlevé, des seconds membres des productions, des variables effaçables. (Remarquons qu'on ne peut pas simplement supprimer ces variables car une variable effaçable peut être utilisée dans la production d'un mot terminal.)

La dernière étape revient à supprimer (de façon récursive) les règles de la forme $A \to \varepsilon$.

- Supprimer les ε-productions, $A \to \varepsilon$, modifier P' en conséquence,
- si une variable A apparaît uniquement comme premier membre d'une ε-production, l'effacer des seconds membres des autres productions. Cette étape pouvant créer de nouvelles ε-productions, répéter si nécessaire le point précédent.

A la fin de cette procédure, si S' appartenait au départ à E, il faut encore ajouter la règle $S' \to \varepsilon$ pour que la grammaire obtenue puisse également générer ε.

■

Exemple VI.6.4. Soit la grammaire dont les règles sont

$$\begin{array}{rcl} S & \to & ACA \\ A & \to & aAaD \mid B \mid C \\ B & \to & bB \mid b \\ C & \to & cS \mid \varepsilon \\ D & \to & \varepsilon. \end{array}$$

Puisque S apparaît dans la production $C \to cS$, la première étape consiste à introduire une nouvelle variable S' et les règles deviennent

$$\begin{array}{rcl} S' & \to & S \\ S & \to & ACA \\ A & \to & aAaD \mid B \mid C \\ B & \to & bB \mid b \\ C & \to & cS \mid \varepsilon \\ D & \to & \varepsilon. \end{array}$$

Appliquons l'algorithme de recherche des variables effaçables. On trouve

$E_0 = \{C, D\}$, $E_1 = \{A, C, D\}$, $E_2 = \{S, A, C, D\}$ et $E_3 = \{S', S, A, C, D\} = E$.

En suivant la preuve précédente, on remplace les règles comme suit

$$\begin{array}{rcl} S' & \to & S \mid \varepsilon \\ S & \to & ACA \mid CA \mid AA \mid AC \mid A \mid C \mid \varepsilon \\ A & \to & aAaD \mid B \mid C \mid aAa \mid aaD \mid aa \mid \varepsilon \\ B & \to & bB \mid b \\ C & \to & cS \mid c \mid \varepsilon \\ D & \to & \varepsilon. \end{array}$$

Il ne reste plus qu'à éliminer les ε-productions. En particulier, puisque D est le premier membre de l'unique règle $D \to \varepsilon$, on peut supprimer D de tous les seconds membres. On a

$$\begin{array}{rcl} S' & \to & S \\ S & \to & ACA \mid CA \mid AA \mid AC \mid A \mid C \\ A & \to & aAa \mid B \mid C \mid aa \\ B & \to & bB \mid b \\ C & \to & cS \mid c. \end{array}$$

Cette dernière grammaire génère le même langage que la grammaire de départ à l'exception du mot vide (en effet, $\varepsilon \in E$). Pour obtenir une grammaire équivalente, il suffit d'ajouter la règle $S' \to \varepsilon$.

Ainsi, on peut toujours se ramener à une grammaire "essentiellement" monotone. L'adjectif *essentiellement* stipule qu'on autorise l'unique ε-production $S \to \varepsilon$ permettant de générer le mot vide et que le symbole initial S n'apparaît dans aucun second membre des règles.

6.2. Elimination des règles $A \to B$.

Remarque VI.6.5. Une règle de la forme $A \to B$, $A, B \in V$, revient simplement à renommer une variable A en B. On dira d'une telle règle qu'il s'agit d'une 1-*production*. Dans le cas $A \to A$, on parle de 1-*production circulaire*.

Définition VI.6.6. Soient $A, A_1, \ldots, A_n, B$ des variables d'une grammaire G. Une dérivation de la forme

$$A \Rightarrow A_1 \Rightarrow \cdots \Rightarrow A_n \Rightarrow B,$$

où chaque production est une 1-production, est une *chaîne*.

Soit A une variable étant le premier membre d'une 1-production. L'algorithme suivant[16] permet de déterminer toutes les variables apparaissant dans une chaîne débutant en A. Posons $C_0 = \{A\}$ et $C_{-1} = \emptyset$. Pour $i \geq 0$,

$$C_{i+1} = C_i \cup \{C \in V \mid \exists B \in C_i \setminus C_{i-1} : B \to C \in P\}.$$

La procédure s'arrête lorsque $C_i = C_{i+1}$. On notera ce dernier ensemble $\mathcal{C}(A)$. Une fois encore, puisque V est fini, l'algorithme s'achève toujours.

Proposition VI.6.7. *Soit $G = (V, \Sigma, P, S)$ une grammaire essentiellement monotone. Il existe une grammaire équivalente G' ne contenant aucune* 1-*production. De plus, cette grammaire peut être obtenue de manière effective.*

[16] On utilise ici une approche "top-down".

Démonstration. Cela découle immédiatement de la constatation faite à la remarque VI.6.5. Pour toute variable A qui est le premier membre d'une 1-production, les règles de la nouvelle grammaire G' qui ont pour premier membre A sont de la forme $A \to w$ où

- $w \notin V$,
- $\exists B \in \mathcal{C}(A) : B \to w \in P$.

De cette manière, il est clair qu'on élimine les 1-productions sans pour autant modifier le langage généré. (Attirons l'attention du lecteur sur le fait que $A \in \mathcal{C}(A)$. Ainsi, si $A \to w \in P$ avec $w \notin V$, alors $A \to w$ est encore une règle de G'.)

Si A est une variable n'apparaissant dans aucun premier membre des 1-productions de G, les règles correspondantes de G et de G' sont identiques.

Les variables, l'alphabet des symboles terminaux et le symbole initial de G' coïncident avec ceux de G.

■

Exemple VI.6.8. Poursuivons l'exemple VI.6.4. Nous avions obtenu la grammaire essentiellement monotone

$$\begin{array}{rcl} S' & \to & S \mid \varepsilon \\ S & \to & ACA \mid CA \mid AA \mid AC \mid A \mid C \\ A & \to & aAa \mid B \mid C \mid aa \\ B & \to & bB \mid b \\ C & \to & cS \mid c. \end{array}$$

Les 1-productions sont $S' \to S$, $S \to A$, $S \to C$, $A \to B$ et $A \to C$. Ainsi, on trouve

$$\mathcal{C}(S') = \{S', S, A, B, C\},\ \mathcal{C}(S) = \{S, A, B, C\} \text{ et } \mathcal{C}(A) = \{A, B, C\}.$$

Par conséquent, la nouvelle grammaire est

$$\begin{array}{rcl} S' & \to & \underbrace{\varepsilon}_{S'} \mid \underbrace{ACA \mid CA \mid AA \mid CA}_{S} \mid \underbrace{aAa \mid aa}_{A} \mid \underbrace{bB \mid b}_{B} \mid \underbrace{cS \mid c}_{C} \\ S & \to & \underbrace{ACA \mid CA \mid AA \mid CA}_{S} \mid \underbrace{aAa \mid aa}_{A} \mid \underbrace{bB \mid b}_{B} \mid \underbrace{cS \mid c}_{C} \\ A & \to & \underbrace{aAa \mid aa}_{A} \mid \underbrace{bB \mid b}_{B} \mid \underbrace{cS \mid c}_{C} \\ B & \to & bB \mid b \\ C & \to & cS \mid c. \end{array}$$

On a simplement appliqué la méthode fournie dans la preuve précédente. Par exemple, puisque $A \to B$, A est le premier membre d'une 1-production. Ainsi, on doit considérer les seconds membres de toutes les productions des éléments de $\mathcal{C}(A)$ et qui ne sont pas restreints à une unique variable. Les accolades permettent de rappeler de quelle variable de $\mathcal{C}(A)$ proviennent les règles qui ont été ajoutées.

Remarque VI.6.9. Il est assez aisé de se convaincre que l'application de la procédure décrite ci-dessus à une grammaire essentiellement monotone fournit encore une grammaire essentiellement monotone.

6.3. Elimination des symboles inutiles.

Définition VI.6.10. Soit $G = (V, \Sigma, P, S)$ une grammaire hors contexte. Un symbole $x \in V \cup \Sigma$ est *utile* si il existe une dérivation

$$S \Rightarrow_G^* uxv \Rightarrow_G^* w$$

avec $u, v \in (V \cup \Sigma)^*$ et $w \in \Sigma^*$. Dans le cas contraire, x est dit *inutile*. En d'autres termes, un symbole terminal est utile s'il apparaît dans un mot du langage généré et une variable est utile si elle contribue à une dérivation permettant d'obtenir, à partir du symbole initial, un mot de Σ^*.

Pour éliminer les symboles inutiles, nous allons procéder en deux parties. Tout d'abord, nous détectons les variables permettant de générer des mots formés de symboles terminaux. L'algorithme est semblable à celui déterminant les symboles effaçables. Posons

$$T_0 = \{A \in V \mid \exists w \in \Sigma^* : A \to w \in P\}.$$

Pour $i \geq 0$, on définit

$$T_{i+1} = T_i \cup \{A \in V \mid \exists w \in (T_i \cup \Sigma)^* : A \to w \in P\}.$$

Puisque V est fini, la suite des T_i se stabilise. Soit T, l'ensemble des variables permettant d'obtenir un mot sur Σ. Si A n'appartient pas à T, alors A est inutile.

Proposition VI.6.11. *Soit $G = (V, \Sigma, P, S)$ une grammaire. Avec les notations introduites précédemment, il existe une grammaire équivalente G' ne contenant que les variables de T. De plus, cette grammaire peut être obtenue de manière effective.*

Démonstration. Il suffit de supprimer les règles faisant intervenir une variable de $V \setminus T$. Ainsi, l'ensemble des variables de G' est T, l'ensemble des productions de G' est

$$\{A \to w \in P \mid A \in T, w \in (T \cup \Sigma)^*\}$$

et l'alphabet des symboles terminaux de G' est l'ensemble des symboles terminaux apparaissant dans les seconds membres des productions de G'.

■

Exemple VI.6.12. Soit la grammaire dont les règles sont

$$\begin{array}{rcl} S & \to & AC \mid BS \mid B \\ A & \to & aA \mid aF \\ B & \to & CF \mid b \\ C & \to & cC \mid D \\ D & \to & aD \mid BD \mid C \\ E & \to & aA \mid BSA \\ F & \to & bB \mid b. \end{array}$$

En appliquant l'algorithme précédent, on trouve

$$T_0 = \{B, F\},\ T_1 = \{S, A, B, F\},\ T_2 = \{S, A, B, E, F\} = T.$$

En éliminant les règles faisant intervenir C ou D, on obtient

$$\begin{array}{rcl} S & \to & BS \mid B \\ A & \to & aA \mid aF \\ B & \to & b \\ E & \to & aA \mid BSA \\ F & \to & bB \mid b. \end{array}$$

Cette première étape n'est pas suffisante pour éliminer complètement les symboles inutiles. En effet, si on considère l'exemple suivant

$$\begin{array}{rcl} S & \to & A \\ A & \to & Aa \mid bA \mid b \\ B & \to & b, \end{array}$$

bien que la variable B appartienne à T, elle ne joue aucun rôle dans les dérivations obtenues depuis S. En effet, B n'est pas accessible depuis S. Cela signifie que $S \not\Rightarrow^* uBv$, $u, v \in (V \cup \Sigma)^*$, et donc B ne contribue à aucune dérivation à partir de S. La seconde étape permettant d'éliminer les symboles inutiles consiste à conserver uniquement les symboles accessibles.

Définition VI.6.13. Soit $G = (V, \Sigma, P, S)$ une grammaire hors contexte. Une variable A est *accessible* si

$$S \Rightarrow^* uAv$$

avec $u, v \in (V \cup \Sigma)^*$. Sinon, A est *inaccessible*.

La détection des variables accessibles est comparable à la recherche des chaînes. Soient $A_0 = \{S\}$ et $A_{-1} = \emptyset$. Pour $i \geq 0$,

$$A_{i+1} = A_i \cup \{B \in V \mid \exists C \in A_i \setminus A_{i-1}, u, v \in (V \cup \Sigma)^* : C \to uBv\}.$$

La procédure s'arrête lorsque $A_i = A_{i+1}$ et l'ensemble obtenu est clairement l'ensemble des variables accessibles.

Proposition VI.6.14. *Soit $G = (V, \Sigma, P, S)$ une grammaire. Il existe une grammaire équivalente G' ne contenant aucun symbole inutile. De plus, cette grammaire peut être obtenue de manière effective.*

Démonstration. On applique tout d'abord à G la proposition VI.6.11 pour obtenir une grammaire ne contenant que des variables permettant de

produire des symboles terminaux. On applique ensuite l'algorithme précédent pour en déterminer les variables accessibles depuis le symbole initial. Tout règle faisant intervenir une variable non accessible peut être supprimée.

■

Exemple VI.6.15. Poursuivons l'exemple VI.6.12. On a

$$A_0 = \{S\} \quad \text{et} \quad A_1 = \{S, B\}.$$

En supprimant les symboles inaccessibles, on obtient

$$\begin{array}{rcl} S & \to & BS \mid B \\ B & \to & b. \end{array}$$

Remarque VI.6.16. On ne peut inverser impunément l'ordre des deux algorithmes. Il faut tout d'abord rechercher l'ensemble T des variables permettant d'obtenir des symboles terminaux et ensuite éliminer les symboles inaccessibles. En effet, considérons la grammaire simpliste

$$\begin{array}{rcl} S & \to & a \mid AB \\ A & \to & b \\ B & \to & B. \end{array}$$

L'ensemble T est $\{S, A\}$, ainsi une première réduction donne

$$\begin{array}{rcl} S & \to & a \\ A & \to & b. \end{array}$$

Puisque A est inaccessible, il reste uniquement $S \to a$.

Par contre, si on recherche d'abord les symboles accessibles, on trouve $A_1 = \{S, A, B\}$. La grammaire de départ reste inchangée puisqu'aucun symbole n'est inaccessible. Si on recherche ensuite les éléments de T, on trouve $T = \{S, A\}$ et donc on obtient

$$\begin{array}{rcl} S & \to & a \\ A & \to & b \end{array}$$

qui n'a pas la forme voulue. En effet, dans cet ordre, l'élimination des variables n'appartenant pas à T peut créer de nouvelles variables inaccessibles.

6.4. Forme normale de Chomsky.

Définition VI.6.17. Une grammaire hors contexte $G = (V, \Sigma, P, S)$ est sous *forme normale de Chomsky* si les règles de G sont toutes de l'une des formes suivantes :

- $A \to BC$ où $A \in V$, $B, C \in V \setminus \{S\}$,
- $A \to a$ où $A \in V$, $a \in \Sigma$,
- $S \to \varepsilon$.

L'intérêt pratique de la mise sous forme de Chomsky est que les arbres d'analyse correspondants seront des arbres binaires[17] (i.e., le nombre de fils d'un noeud est au plus deux). Nous verrons aussi à la section suivante que

[17] cf. un cours d'introduction à l'algorithmique pour le traitement systématique des arbres binaires et des algorithmes associés.

disposer d'une forme normale pour les grammaires permet de simplifier les développements de certaines preuves.

Théorème VI.6.18. *Soit $G = (V, \Sigma, P, S)$ une grammaire hors contexte. On peut construire de manière effective une grammaire équivalente G' mise sous forme normale de Chomsky.*

Démonstration. Au vu des sous-sections précédentes, on peut supposer que G est essentiellement monotone et qu'elle ne contient aucune 1-production ni symbole inutile. Ainsi, une règle de la grammaire G est de l'une des formes suivantes :

- $S \to \varepsilon$,
- $A \to a$ où $A \in V$, $a \in \Sigma$,
- $A \to w$ où $A \in V$, $w \in ((V \cup \Sigma) \setminus \{S\})^*$ et $|w| \geq 2$.

Les deux premiers types de règles satisfont la forme de Chomsky. Il nous reste à montrer comment remplacer le troisième type de règles. Soit la règle $A \to w$ où

$$w = w_1 A_1 w_2 \cdots w_n A_n w_{n+1}, \quad \text{avec } w_i \in \Sigma^*, A_i \in V \setminus \{S\}.$$

Si $w_i \neq \varepsilon$, on notera $w_i = w_{i,1} \cdots w_{i,\ell_i}$ avec $w_{i,j} \in \Sigma$. Sans changer le langage généré, on peut remplacer la règle $A \to w$ par les règles

$$\begin{aligned} A &\to W_{1,1} \cdots W_{1,\ell_1} A_1 W_{2,1} \cdots W_{n,\ell_n} A_n W_{n+1,1} \cdots W_{n+1,\ell_{n+1}} \\ W_{1,1} &\to w_{1,1} \\ &\vdots \\ W_{n+1,\ell_{n+1}} &\to w_{n+1,\ell_{n+1}} \end{aligned}$$

où les W_i sont de nouvelles variables. Il reste simplement à modifier la première de ces nouvelles règles pour avoir une grammaire mise sous forme de Chomsky. Si une règle est de la forme

$$A \to A_1 \cdots A_n, \ n \geq 3,$$

en faisant intervenir de nouvelles variables, on peut la remplacer par les règles

$$\begin{aligned} A &\to A_1 B_1 \\ B_1 &\to A_2 B_2 \\ &\vdots \\ B_{n-2} &\to A_{n-1} A_n. \end{aligned}$$

■

Exemple VI.6.19. Considérons la grammaire dont les règles sont

$$\begin{aligned} S &\to SaB \mid aB \\ B &\to bB \mid \varepsilon. \end{aligned}$$

Pour pouvoir obtenir la forme normale de Chomsky, nous remplaçons d'abord cette grammaire par une grammaire équivalente dont le symbole initial n'apparaît dans aucun second membre et qui est essentiellement monotone

$$\begin{array}{rcl} S' & \to & S \\ S & \to & SaB \mid aB \mid Sa \mid a \\ B & \to & bB \mid b. \end{array}$$

Ensuite, on supprime les 1-productions et on obtient

$$\begin{array}{rcl} S' & \to & SaB \mid aB \mid Sa \mid a \\ S & \to & SaB \mid aB \mid Sa \mid a \\ B & \to & bB \mid b \end{array}$$

et on remarque qu'aucun symbole n'est inutile. En introduisant de nouvelles variables, on a tout d'abord

$$\begin{array}{rcl} S' & \to & SAB \mid AB \mid SA \mid a \\ S & \to & SAB \mid AB \mid SA \mid a \\ B & \to & B'B \mid b \\ A & \to & a \\ B' & \to & b. \end{array}$$

Enfin, pour obtenir des règles de longueur au plus deux, on a

$$\begin{array}{rcl} S' & \to & ST \mid AB \mid SA \mid a \\ S & \to & ST \mid AB \mid SA \mid a \\ B & \to & B'B \mid b \\ A & \to & a \\ B' & \to & b \\ T & \to & AB. \end{array}$$

6.5. Forme normale de Greibach.

Définition VI.6.20. Une grammaire hors contexte $G = (V, \Sigma, P, S)$ est *sous forme normale de Greibach* si les règles de G sont toutes de l'une des formes suivantes :

- $A \to aA_1 \cdots A_n$ avec $A \in V$, $a \in \Sigma$, $A_i \in V \setminus \{S\}$,
- $A \to a$ avec $A \in V$ et $a \in \Sigma$,
- $S \to \varepsilon$.

L'intérêt pratique de la mise sous forme normale de Greibach est qu'à chaque dérivation, on détermine un préfixe de plus en plus long formé uniquement de symboles terminaux. Cela permet de construire plus aisément des analyseurs permettant de retrouver l'arbre d'analyse associé à un mot généré. Dans ce texte introductif, nous énonçons le résultat suivant.

Théorème VI.6.21. [18] *Soit $G = (V, \Sigma, P, S)$ une grammaire hors contexte. On peut construire de manière effective une grammaire équivalente G' mise sous forme normale de Greibach.*

[18] Le lecteur intéressé trouvera par exemple plus de détails dans T. A. Sudkamp, *Languages and Machines, An introduction to the Theory of Computer Science*, 2e édition, Addison-Wesley, (1998), pp. 140–147.

7. Lemme de la pompe

On dispose d'un analogue du lemme de la pompe dans le cadre des langages hors contexte.

Proposition VI.7.1 (Lemme de la pompe – Théorème de Bar-Hillel). *Soit L un langage hors contexte. Il existe $p \in \mathbb{N} \setminus \{0\}$ tel que tout mot $z \in L$ de longueur $|z| \geq p$ peut s'écrire $z = uvwxy$, $u, v, w, x, y \in \Sigma^*$ avec $|vwx| < p$, $vx \neq \varepsilon$ et pour tout $n \in \mathbb{N}$,*

$$uv^n wx^n y \in L.$$

Pour obtenir ce résultat, nous allons tirer parti de la mise sous forme normale de Chomsky. Néanmoins, on pourrait obtenir un résultat analogue sans recourir à cette simplification.

Lemme VI.7.2. *Soit $G = (V, \Sigma, P, S)$ une grammaire hors contexte mise sous forme normale de Chomsky. Si la dérivation $A \Rightarrow^* w$, $w \in \Sigma^*$, a un arbre d'analyse de hauteur n, alors $|w| \leq 2^{n-1}$.*

Démonstration. Puisque la grammaire est mise sous forme normale de Chomsky, les seuls arbres d'analyse[19] de hauteur 1 dont le fruit est formé de symboles terminaux sont de la forme donnée à la figure VI.9 et leurs fruits

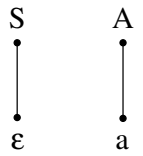


FIGURE VI.9. Arbres d'analyse de hauteur 1 pour une grammaire mise sous forme normale de Chomsky.

sont de longueur au plus 1. Supposons à présent le résultat satisfait pour les arbres de hauteur au plus n et vérifions-le pour les arbres de hauteur $n+1$. Pour obtenir un arbre de hauteur $n+1$, on applique nécessairement une première règle de la forme $A \to BC$. Les sous-arbres de racine B et C sont chacun de hauteur au plus n. Par hypothèse de récurrence, leurs fruits respectifs sont de longueur au plus 2^{n-1}. Le fruit de l'arbre, obtenu par concaténation des fruits des deux sous-arbres, a donc une longueur majorée par $2.2^{n-1} = 2^n$.

■

Par contraposition, ce résultat se réexprime comme suit.

Corollaire VI.7.3. *Soit $G = (V, \Sigma, P, S)$ une grammaire hors contexte mise sous forme normale de Chomsky. Si $S \Rightarrow^* w$ avec $w \in \Sigma^*$ et $|w| > 2^{n-1}$*

[19] A la section 2, on a supposé que les seconds membres des productions n'étaient jamais égaux à ε. Ici, nous autorisons une telle situation. Il est clair que cette généralisation ne modifie en rien les développements de la section 2.

(donc en particulier, avec $|w| \geq 2^n$), alors l'arbre d'analyse associé à cette dérivation est de hauteur au moins $n+1$.

Nous en arrivons à présent à la preuve du lemme de la pompe.

Démonstration. Sans restriction, nous pouvons supposer que la grammaire est mise sous forme normale de Chomsky. Posons $\#V = m$. Soit z un mot généré par G de longueur au moins $2^m =: p$. Au vu du corollaire précédent, on dispose d'un arbre d'analyse de fruit z et de hauteur au moins $m+1$. Ainsi, cet arbre contient un chemin de longueur au moins $m+1$ débutant en S et aboutissant en un symbole terminal. Une illustration de cette situation est donnée à la figure VI.10. Ce chemin de longueur $m+1$

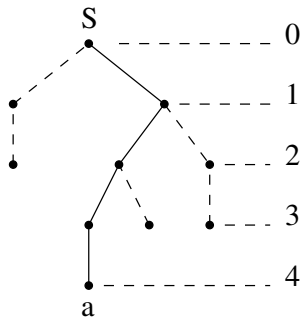


FIGURE VI.10. Un arbre d'analyse pour une grammaire sous forme de Chomsky.

passe par $m+2$ sommets de l'arbre dont $m+1$ sont des variables. Or $\#V = m$. Par conséquent, ce chemin contient au moins deux fois la même variable A. Dans ce chemin, nous considérons les 2 occurrences de A, *le plus bas possible* dans l'arbre (i.e., le plus loin de la racine). Schématiquement, dans l'arbre d'analyse de z, on a la situation représentée à la figure VI.11. Ainsi, la grammaire contient les dérivations

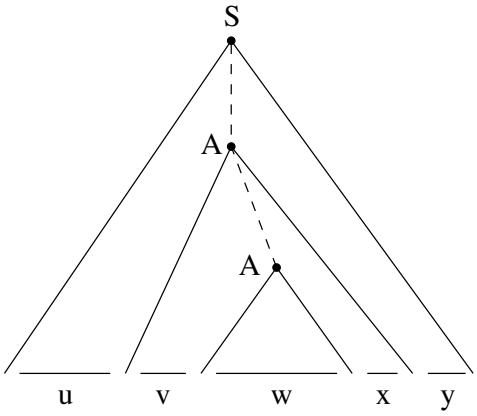


FIGURE VI.11. Un arbre d'analyse avec A apparaissant deux fois.

$$S \Rightarrow^* uAy, \quad A \Rightarrow^* vAx \quad \text{et} \quad A \Rightarrow^* w.$$

Par conséquent, en appliquant n fois la dérivation centrale, on obtient

$$S \Rightarrow^* uAy \Rightarrow^* uvAxy \Rightarrow^* \cdots \Rightarrow^* uv^nAx^ny \Rightarrow^* uv^nwx^ny$$

et les mots uv^nwx^ny appartiennent à $L(G)$ pour tout $n \in \mathbb{N}$.

Pour terminer la démonstration du résultat, nous devons encore vérifier que $|vwx| < p$ et $vx \neq \varepsilon$. Puisque la grammaire est sous forme de Chomsky, la dérivation $A \Rightarrow^* vAx$ doit nécessairement débuter par une production de la forme $A \to BC$. Supposons que la deuxième occurrence de la variable A provienne de B (on dispose d'un raisonnement analogue pour l'autre cas). Ainsi,

$$A \Rightarrow BC \Rightarrow^* vArC \Rightarrow^* vArs = vAx.$$

La variable C ne peut donner ε (en effet, seul $S \to \varepsilon$). On en conclut que $x \neq \varepsilon$ et donc $vx \neq \varepsilon$. Le sous-arbre de racine A et de fruit vwx est de hauteur au plus m (au vu du choix des deux occurrences de A prises le plus bas possible). Par conséquent, $|vwx| \leq 2^{m-1} < p$.

■

Ce résultat peut être utilisé pour montrer que certains langages ne sont pas algébriques.

Exemple VI.7.4. Le langage

$$L = \{a^nb^nc^n \mid n \in \mathbb{N}\}$$

n'est pas algébrique. Procédons par l'absurde. Soit p l'entier donné dans l'énoncé du lemme de la pompe. Le mot $z = a^pb^pc^p$ est de longueur au moins p. Il existe donc des mots u, v, w, x, y tels que

$$a^pb^pc^p = uvwxy$$

avec $|vwx| < p$ et $vx \neq \varepsilon$. Par conséquent, vwx ne peut contenir simultanément des lettres a, b et c. Ceci contredit le fait que uv^nwx^ny doive appartenir au langage pour tout n.

7.1. Théorème de Parikh. Un autre résultat peut parfois s'avérer utile pour vérifier qu'un langage n'est pas algébrique. Nous ne ferons ici que d'énoncer le théorème de Parikh.

Définition VI.7.5. Un sous-ensemble M de $\mathbb{N}^k$ est dit *linéaire* s'il existe $p_0, p_1, \ldots, p_s \in \mathbb{N}^k$ tels que

$$M = \{p_0 + \sum_{i=1}^{s} \lambda_i\, p_i \mid \lambda_1, \ldots, \lambda_s \in \mathbb{N}\} = p_0 + \mathbb{N}.p_1 + \cdots + \mathbb{N}.p_s.$$

On dit que p_0 est la *constante* de M et les p_i's, $i \geq 1$, en sont les *périodes*.

Une union finie d'ensembles linéaires est un ensemble *semi-linéaire*.

Théorème VI.7.6 (Parikh). *Si $L \subset \{\sigma_1, \ldots, \sigma_k\}$ est un langage algébrique, alors $\psi(L)$ est un ensemble semi-linéaire de $\mathbb{N}^k$.*

Remarque VI.7.7. La réciproque de ce résultat est fausse. En effet, nous savons que le langage $L = \{a^n b^n c^n \mid n \in \mathbb{N}\}$ n'est pas algébrique. Par contre,

$$\psi(L) = \mathbb{N}. \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$$

est semi-linéaire.

8. Automates à pile

D'une part, nous avons vu dans les sections précédentes que les grammaires hors contexte étaient utilisées pour générer les langages algébriques. D'une certaine façon, les grammaires sont une généralisation des expressions régulières qui permettent quant à elles de générer les langages réguliers.

D'autre part, nous avons montré que les automates finis acceptent exactement les langages réguliers. L'ensemble des langages réguliers étant un sous-ensemble strict de l'ensemble des langages algébriques, pour espérer trouver l'analogue des automates finis, nous allons étendre les possibilités de ces derniers par l'ajout d'une pile. Un automate fini est, par définition, une machine ne disposant que d'une mémoire finie (le nombre de configurations qui peuvent être mémorisées est égal à son nombre d'états). L'ajout d'une pile[20] permet d'étendre les possibilités de mémorisation, puisque, comme nous allons le voir, la capacité de stockage d'une pile peut être arbitrairement grande.

Une *pile* est un dispositif du type[21] "dernier entré, premier sorti". On

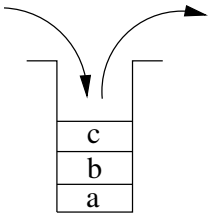


FIGURE VI.12. Représentation d'une pile.

peut la représenter à l'aide d'un mot fini. Par convention, on notera l'alphabet de pile Π. Une pile est donc un mot $p \in \Pi^*$. Les opérations dont on dispose pour une pile sont

- tester si la pile est vide, i.e., déterminer si $p = \varepsilon$,
- empiler,
- dépiler.

[20] Dispositif permettant de mémoriser un nombre arbitraire de symboles appartenant à un alphabet fini.

[21] LIFO : Last In First Out. Penser par exemple à une pile d'assiettes.

Si w est une pile, *empiler* le symbole $\sigma \in \Pi$ est l'opération

$$w \mapsto \sigma w.$$

Si w est une pile non vide, elle est de la forme $\sigma w'$, $\sigma \in \Pi$, $w' \in \Pi^*$, et *dépiler* un symbole est l'opération

$$\sigma w' \mapsto w'.$$

On peut étendre ces notions à des mots. Ainsi, empiler un mot $u = u_1 \cdots u_\ell$ revient à empiler successivement les lettres $u_1, \ldots, u_\ell$. Partant de la pile w, on obtient

$$w \mapsto u_1 w \mapsto u_2 u_1 w \mapsto \ldots \mapsto u_\ell \cdots u_1 w = u^R w.$$

Remarque VI.8.1. Dans la suite, il faudra être attentif à cette situation faisant apparaître le miroir de u (cf., par exemple, la relation de transition donnée dans la proposition VI.8.6).

Définition VI.8.2. Un *automate à pile*[22] est la donnée d'un sextuple $\mathcal{A} = (Q, \Sigma, \Pi, \delta, q_0, F)$ où Q est un ensemble fini d'états, Σ est l'alphabet de l'automate, Π est l'alphabet de pile,

$$\delta \subset Q \times \Sigma^* \times \Pi^* \times Q \times \Pi^*$$

est la relation de transition de $\mathcal{A}$, $q_0 \in Q$ est l'état initial et $F \subset Q$ l'ensemble des états finals. On suppose bien sûr que δ est un ensemble fini. Une *configuration* de l'automate à pile est un triplet $[q, w, p]$ de $Q \times \Sigma^* \times \Pi^*$ dont le rôle est de coder l'état q dans lequel se trouve l'automate, le mot w restant à lire et l'état p de la pile. On passe de la configuration $[q, w, p]$ à la configuration $[q', w', p']$ si il existe une transition de $\mathcal{A}$ telle que

$$w = mw',\ p = xy,\ p' = z^R y \quad \text{et} \quad (q, m, x, q', z) \in \delta.$$

En d'autres termes, on a lu m, on a dépilé x et empilé z. La représentation sagittale correspondante est donnée à la figure VI.13. Les autres conventions

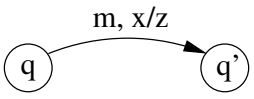


FIGURE VI.13. Une transition d'un automate à pile.

sont analogues à celles utilisées pour représenter les automates finis. On notera dès lors cette transition par

$$[q, w, p] \vdash [q', w', p']$$

[22]Le modèle présenté ici est non déterministe. Le lecteur ayant déjà rencontré la notion d'automate fini pourra s'apercevoir qu'on est en présence d'une relation de transition et non d'une fonction de transition.

et $\vdash^*$ est la fermeture réflexive et transitive de $\vdash$. Un mot w est accepté par l'automate à pile $\mathcal{A}$ si

$$[q_0, w, \varepsilon] \vdash^* [q, \varepsilon, \varepsilon] \quad \text{avec } q \in F.$$

Il est évident que le langage accepté par $\mathcal{A}$ est

$$L(\mathcal{A}) = \{w \in \Sigma^* \mid \exists q \in F : [q_0, w, \varepsilon] \vdash^* [q, \varepsilon, \varepsilon]\}.$$

Deux automates à pile sont *équivalents* s'ils acceptent le même langage.

Exemple VI.8.3. L'automate à pile représenté à la figure VI.14 accepte exactement le langage $\{a^n b^n \mid n \in \mathbb{N}\}$. La pile d'alphabet $\{A\}$ y est utilisée

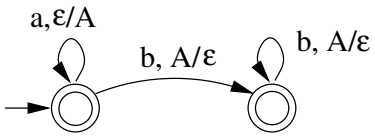


FIGURE VI.14. Un automate à pile acceptant $\{a^n b^n \mid n \in \mathbb{N}\}$.

pour retenir le nombre de a qui ont été lus (à chaque a lu, un A est empilé). De plus, à chaque b rencontré, on dépile un A. Ainsi, on ne peut obtenir une pile vide que si on a lu le même nombre de a que de b. Remarquons également qu'un mot contenant un facteur ba ne peut jamais être accepté par l'automate. De même, on ne saurait lire plus de b que de a , car pour pouvoir lire un b, il faut être en mesure de dépiler un A. Si les états sont notés 1 et 2, on a par exemple la suite de configurations

$$[1, aabb, \varepsilon] \vdash [1, abb, A] \vdash [1, bb, AA] \vdash [2, b, A] \vdash [2, \varepsilon, \varepsilon].$$

Définition VI.8.4. Un automate à pile est *déterministe* s'il existe au plus une transition résultant de chaque configuration. Par exemple, l'automate donné à la figure VI.14 est déterministe.

Définition VI.8.5. Un automate à pile est *atomique* ou *élémentaire* si chaque transition est de l'une des quatre formes suivantes :

- $(p, \varepsilon, \varepsilon, q, \varepsilon)$: changement d'état, sans aucune autre action
- $(p, \sigma, \varepsilon, q, \varepsilon)$: lecture d'une lettre $\sigma \in \Sigma$
- $(p, \varepsilon, \alpha, q, \varepsilon)$: dépilement d'une lettre $\alpha \in \Pi$
- $(p, \varepsilon, \varepsilon, q, \alpha)$: empilement d'une lettre $\alpha \in \Pi$.

Il est clair qu'on peut remplacer un automate à pile par un automate à pile élémentaire en ajoutant de nouveaux états (les constructions sont semblables à celles développées dans le lemme II.2.8).

Proposition VI.8.6. *Soit $G = (V, \Sigma, P, S)$ une grammaire hors contexte. L'automate à pile $\mathcal{A} = (Q, \Sigma, \Pi, \delta, q_0, F)$ où*

- $Q = \{q_0, f\}$,
- $\Pi = V \cup \Sigma$,

- $F = \{f\}$ *et*
- *la relation de transition* δ *est donnée par*

$$\{(q_0, \varepsilon, \varepsilon, f, S)\} \cup \{(f, \varepsilon, A, f, w^R) \mid A \to w \in P\} \cup \{(f, \sigma, \sigma, f, \varepsilon) \mid \sigma \in \Sigma\},$$

accepte exactement le langage $L(G)$.

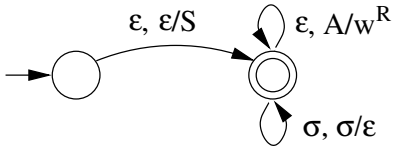


FIGURE VI.15. Automate acceptant $L(G)$.

Démonstration. Montrons tout d'abord que si $u \in \Sigma^*$, $v \in (V \cup \Sigma)^*$ et

$$[f, u, S] \vdash^* [f, \varepsilon, v],$$

alors il existe une dérivation à gauche telle que

$$S \Rightarrow^* uv.$$

Nous prouvons ce résultat par récurrence sur le nombre m de transitions à effectuer pour passer de $[f, u, S]$ à $[f, \varepsilon, v]$. Si $m = 0$, alors $u = \varepsilon$ et $S = v$ et il est clair que $S \Rightarrow^* S$. Supposons le résultat satisfait pour $m = t$ et démontrons-le pour $m = t + 1$. Puisque les transitions de l'automate sont de la forme $(\sigma, \sigma/\varepsilon)$ ou $(\varepsilon, A/w^R)$, il est clair que pour passer de $[f, u, S]$ à $[f, \varepsilon, v]$, il y a au moins une transition de la seconde forme[23]. En particulier, nous allons considérer la dernière fois où l'on applique une transition de ce type dans la suite de configurations menant de $[f, u, S]$ à $[f, \varepsilon, v]$. Ainsi, $[f, u, S] \vdash^* [f, \varepsilon, v]$ se décompose en

$$[f, u, S] \vdash^* \underbrace{[f, r, Av'] \vdash [f, r, wv']}_{\text{dernière application } (\varepsilon, A/w^R)} \vdash^* [f, \varepsilon, v]$$

où r est un suffixe de u, i.e., $u = u'r$, car on a pu appliquer certaines transitions de la forme $(\sigma, \sigma/\varepsilon)$ pour lire un préfixe u' de u et où $wv' = rv$ car pour lire r, puisqu'on applique uniquement des règles de la forme $(\sigma, \sigma/\varepsilon)$, il faut que wv' débute par r. Ainsi,

$$[f, u, S] = [f, u'r, S] \vdash^* [f, \varepsilon\, r, Av'] = [f, r, Av']$$

et donc

$$[f, u', S] \vdash^* [f, \varepsilon, Av'].$$

Ayant dans ce dernier cas au plus t transitions, on peut appliquer l'hypothèse de récurrence : il existe une dérivation à gauche

$$S \Rightarrow^* u'Av'.$$

[23]En effet, on ne saurait appliquer la transition $(\sigma, \sigma/\varepsilon)$ à la configuration $[f, u, S]$ puisqu'il faudrait dépiler σ d'une pile ne contenant que S.

Or u' appartient à Σ^*. Ainsi, si on applique la règle $A \to w$, on conserve une dérivation à gauche et on obtient

$$S \Rightarrow^* u'Av' \Rightarrow u'wv' = u'rv = uv.$$

La réciproque est également vraie. S'il existe une dérivation à gauche $S \Rightarrow^* uv$ avec $u \in \Sigma^*$ et $v \in (V \cup \Sigma)^*$, alors $[f, u, S] \vdash^* [f, \varepsilon, v]$. Cette partie se démontre de manière semblable à la première partie[24].

Il nous faut à présent prouver que $L(\mathcal{A}) = L(G)$. Si w appartient à $L(\mathcal{A})$, cela signifie que $[q_0, w, \varepsilon] \vdash^* [f, \varepsilon, \varepsilon]$. Vu la forme de l'automate, on a

$$[q_0, w, \varepsilon] \vdash [f, w, S] \vdash^* [f, \varepsilon, \varepsilon].$$

Vu la première partie de la démonstration, il existe une dérivation à gauche telle que $S \Rightarrow^* w$ et donc, $L(\mathcal{A}) \subseteq L(G)$. Passons à la réciproque. Si $w \in L(G)$, il existe une dérivation (que l'on peut supposer à gauche) telle que $S \Rightarrow^* w$. Si cette dérivation est de longueur au moins un, on applique une dernière règle de la forme $A \to y$ et on peut écrire

$$S \Rightarrow^* xAz \Rightarrow xyz = w \quad \text{avec } x, z \in \Sigma^*.$$

Par la deuxième partie de la preuve, on a $[f, x, S] \vdash^* [f, \varepsilon, Az]$. Ainsi, dans $\mathcal{A}$, on a

$$[q_0, w = xyz, \varepsilon] \vdash [f, xyz, S] \vdash^* [f, yz, Az].$$

De là, puisque $(\varepsilon, A/y^R)$ est une transition de $\mathcal{A}$, il vient

$$[f, yz, Az] \vdash [f, yz, yz] \vdash^* [f, \varepsilon, \varepsilon]$$

où pour conclure, on applique des transitions $(\sigma, \sigma/\varepsilon)$ pour lire et dépiler yz.

■

Remarque VI.8.7. L'automate donné dans la proposition précédente n'est en général pas déterministe car lorsqu'on se trouve dans l'état f et que la pile a un sommet A, i.e., lorsque l'on se trouve dans une configuration $[f, w, Ap]$ avec $w \in \Sigma^*$ et $p \in \Pi^*$, et si la grammaire G possède deux règles de la forme $A \to w_1$ et $A \to w_2$, alors on peut considérer indifféremment les deux transitions $(\varepsilon, A/w_1^R)$ ou $(\varepsilon, A/w_2^R)$.

Il nous faut à présent montrer que tout langage accepté par un automate à pile est algébrique. Nous allons pour ce faire associer à un automate à pile, une grammaire dont les règles sont obtenues à partir des transitions de l'automate.

Soit $\mathcal{A} = (Q, \Sigma, \Pi, \delta, q_0, F)$ un automate à pile où l'on peut supposer que

$$\delta \subset Q \times (\Sigma \cup \{\varepsilon\}) \times (\Pi \cup \{\varepsilon\}) \times Q \times (\Pi \cup \{\varepsilon\}).$$

Il est clair qu'il ne s'agit pas d'une véritable restriction (on autorise à lire, empiler ou dépiler au plus une lettre; si l'automate n'a pas la forme voulue,

[24] Par récurrence sur la longueur de la dérivation.

on peut par exemple le rendre élémentaire). On peut ajouter de nouvelles transitions à δ sans modifier le langage accepté par l'automate. Ainsi, si

$$(q_i, s, \varepsilon, q_j, p) \in \delta, \quad \text{avec } s \in \Sigma \cup \{\varepsilon\}, p \in \Pi \cup \{\varepsilon\},$$

alors pour tout $A \in \Pi$, on ajoute les transitions

$$(q_i, s, A, q_j, Ap).$$

En effet, cela revient à dépiler A puis à l'empiler à nouveau (et ensuite on empile p). Tout mot accepté par l'automate pour lequel on utilise une transition du nouveau type aurait pu aussi être accepté avec la transition originale correspondante. Dans la suite, nous supposerons donc disposer d'un tel automate modifié que nous noterons encore $\mathcal{A}$.

Construisons une grammaire $G = (V, \Sigma, P, S)$ où l'ensemble V des variables est

$$\{S\} \cup \{\langle q_i, A, q_j\rangle \mid q_i, q_j \in Q, A \in \Pi \cup \{\varepsilon\}\}.$$

Un élément de la forme $\langle q_i, A, q_j\rangle$ va être utilisé pour symboliser une suite de transitions permettant de passer de l'état q_i à l'état q_j en dépilant A. Les règles de P sont de quatre types

- $\mathbf{S} \to \langle \mathbf{q_0}, \varepsilon, \mathbf{q}\rangle$, pour tout $q \in F$,
- pour chaque transition de la forme

  $$(q_i, s, A, q_j, B), \ s \in \Sigma \cup \{\varepsilon\}, A, B \in \Pi \cup \{\varepsilon\},$$

  on considère les règles $\langle \mathbf{q_i}, \mathbf{A}, \mathbf{q}\rangle \to \mathbf{s}\langle \mathbf{q_j}, \mathbf{B}, \mathbf{q}\rangle$, pour tout $q \in Q$,
- pour chaque transition de la forme

  $$(q_i, s, A, q_j, AB), \ s \in \Sigma \cup \{\varepsilon\}, A, B \in \Pi,$$

  on considère les règles $\langle \mathbf{q_i}, \mathbf{A}, \mathbf{q}\rangle \to \mathbf{s}\langle \mathbf{q_j}, \mathbf{B}, \mathbf{q'}\rangle\langle \mathbf{q'}, \mathbf{A}, \mathbf{q}\rangle$, pour tous $q, q' \in Q$,
- $\langle \mathbf{q}, \varepsilon, \mathbf{q}\rangle \to \varepsilon$ pour tout $q \in Q$.

Exemple VI.8.8. Reprenons l'automate à pile introduit dans l'exemple VI.8.3 dont les transitions sont

$$(1, a, \varepsilon, 1, A), \ (1, b, A, 2, \varepsilon), \ (2, b, A, 2, \varepsilon).$$

On y ajoute la transition $(1, a, A, 1, AA)$ et on considère la grammaire dont les règles sont

S	$\to$	$\langle 1,\varepsilon,2\rangle \mid \langle 1,\varepsilon,1\rangle$	
$\langle 1,\varepsilon,1\rangle$	$\to$	$a\ \langle 1,A,1\rangle$	$(1,a,\varepsilon,1,A)$
$\langle 1,\varepsilon,2\rangle$	$\to$	$a\ \langle 1,A,2\rangle$	
$\langle 1,A,1\rangle$	$\to$	$b\ \langle 2,\varepsilon,1\rangle$	$(1,b,A,2,\varepsilon)$
$\langle 1,A,2\rangle$	$\to$	$b\ \langle 2,\varepsilon,2\rangle$	
$\langle 2,A,1\rangle$	$\to$	$b\ \langle 2,\varepsilon,1\rangle$	$(2,b,A,2,\varepsilon)$
$\langle 2,A,2\rangle$	$\to$	$b\ \langle 2,\varepsilon,2\rangle$	
$\langle 1,A,1\rangle$	$\to$	$a\ \langle 1,A,1\rangle\ \langle 1,A,1\rangle$	$(1,a,A,1,AA)$
$\langle 1,A,1\rangle$	$\to$	$a\ \langle 1,A,2\rangle\ \langle 2,A,1\rangle$	
$\langle 1,A,2\rangle$	$\to$	$a\ \langle 1,A,1\rangle\ \langle 1,A,2\rangle$	
$\langle 1,A,2\rangle$	$\to$	$a\ \langle 1,A,2\rangle\ \langle 2,A,2\rangle$	
$\langle 1,\varepsilon,1\rangle$	$\to$	ε	
$\langle 2,\varepsilon,2\rangle$	$\to$	ε	

(On a indiqué à chaque fois, de quelle transition provient la règle.) Le mot $aabb$ est accepté par l'automate car on a la suite de configurations

$$[1,aabb,\varepsilon]\vdash[1,abb,A]\vdash[1,bb,AA]\vdash[2,b,A]\vdash[2,\varepsilon,\varepsilon].$$

Ce mot est aussi généré par la grammaire en considérant la dérivation

$$\begin{aligned} S &\Rightarrow \langle 1,\varepsilon,2\rangle \Rightarrow a\langle 1,A,2\rangle \Rightarrow aa\langle 1,A,2\rangle\langle 2,A,2\rangle \Rightarrow aab\langle 2,\varepsilon,2\rangle\langle 2,A,2\rangle \\ &\Rightarrow aab\langle 2,A,2\rangle \Rightarrow aabb\langle 2,\varepsilon,2\rangle \Rightarrow aabb. \end{aligned}$$

Bien que cet exemple ne constitue en rien une preuve, on remarque que la grammaire permet d'une certaine façon de tenir compte des lettres lues dans l'automate et de retenir l'état de la pile.

Le résultat suivant peut aussi être considéré comme une conséquence du théorème de Chomsky-Schützenberger (théorème VI.10.3). A la différence de ce dernier, la preuve donnée ici est constructive : on associe une grammaire de manière canonique à l'automate considéré.

Proposition VI.8.9. *Tout langage accepté par un automate à pile $\mathcal{A}$ est hors contexte.*

Démonstration. Nous supposons être dans les conditions données ci-dessus (i.e., nous disposons d'un automate à pile $\mathcal{A}$ modifié auquel on a associé une grammaire G). Avec les notations qui précèdent, nous devons montrer que dans $\mathcal{A}$, $[q_0,w,\varepsilon]\vdash^*[q,\varepsilon,\varepsilon]$ avec $q\in F$ si et seulement si il existe une dérivation de la grammaire G telle que $S\Rightarrow^* w$.

Montrons tout d'abord que si $[q_i,w,A]\vdash^*[q_j,\varepsilon,\varepsilon]$ avec $A\in\Pi\cup\{\varepsilon\}$, alors il existe une dérivation de G telle que $\langle q_i,A,q_j\rangle\Rightarrow^* w$. On procède par récurrence sur la longueur de la suite de configurations. Si celle-ci est nulle, $q_i=q_j$, $w=\varepsilon$, $A=\varepsilon$ et pour conclure, on remarque que $\langle q_i,\varepsilon,q_i\rangle\to\varepsilon$ est une règle de G. Supposons le résultat acquis pour une suite de longueur n et vérifions-le pour une suite de longueur $n+1$. Si $[q_i,w,A]\vdash^*[q_j,\varepsilon,\varepsilon]$ avec une suite de $n+1$ transitions, alors on peut décomposer cette suite

en l'application d'une première transition suivie par n autres. On a deux possibilités. Tout d'abord

$$[q_i, w, A] \vdash [q_k, v, B] \vdash^* [q_j, \varepsilon, \varepsilon]$$

si $w = sv$ et $(q_i, s, A, q_k, B) \in \delta$, $s \in \Sigma \cup \{\varepsilon\}$. Par hypothèse de récurrence, on a $\langle q_k, B, q_j\rangle \Rightarrow^* v$. De plus, à la transition (q_i, s, A, q_k, B) correspond notamment la règle $\langle q_i, A, q_j\rangle \to s\langle q_k, B, q_j\rangle$. Ainsi,

$$\langle q_i, A, q_j\rangle \Rightarrow s\langle q_k, B, q_j\rangle \Rightarrow^* sv = w.$$

L'autre situation à envisager est

$$[q_i, w, A] \vdash [q_k, v, BA] \vdash^* [q_\ell, y, A] \vdash^* [q_j, \varepsilon, \varepsilon]$$

où la première transition est de la forme (q_i, s, A, q_k, AB), $s \in \Sigma \cup \{\varepsilon\}$ et $w = sv$, $v = xy$. La grammaire contient la règle $\langle q_i, A, q_j\rangle \to s\langle q_k, B, q_\ell\rangle\langle q_\ell, A, q_j\rangle$. Puisque $[q_k, v, BA] \vdash^* [q_\ell, y, A]$ et que $v = xy$, on en tire que $[q_k, x, B] \vdash^* [q_\ell, \varepsilon, \varepsilon]$. Donc par hypothèse de récurrence, on a

$$\langle q_k, B, q_\ell\rangle \Rightarrow^* x \quad \text{et} \quad \langle q_\ell, A, q_j\rangle \Rightarrow^* y.$$

D'où

$$\langle q_i, A, q_j\rangle \Rightarrow s\langle q_k, B, q_\ell\rangle\langle q_\ell, A, q_j\rangle \Rightarrow^* sxy = w.$$

De là, on en conclut aisément que $L(\mathcal{A}) \subseteq L(G)$.

Montrons à présent que si $\langle q_i, A, q_j\rangle \Rightarrow^* w$, $w \in \Sigma^*$, $A \in \Pi \cup \{\varepsilon\}$, alors $[q_i, w, A] \vdash^* [q_j, \varepsilon, \varepsilon]$. On procède une fois encore par récurrence sur la longueur de la dérivation. S'il s'agit d'une dérivation de longueur un, les seules règles de la grammaire donnant un symbole terminal sont de la forme $\langle q, \varepsilon, q\rangle \to \varepsilon$ et on a bien $[q_i, \varepsilon, \varepsilon] \vdash^* [q_i, \varepsilon, \varepsilon]$. Supposons à présent la propriété satisfaite pour les dérivations de longueur au plus n et démontrons-la pour les dérivations de longueur $n + 1$. Si la première règle appliquée est de la forme $\langle q_i, A, q_j\rangle \to s\langle q_k, B, q_j\rangle$, alors on a

$$\langle q_i, A, q_j\rangle \Rightarrow s\langle q_k, B, q_j\rangle \Rightarrow^* sv = w.$$

Par hypothèse de récurrence, $[q_k, v, B] \vdash^* [q_j, \varepsilon, \varepsilon]$. De plus, par construction de G, la règle $\langle q_i, A, q_j\rangle \to s\langle q_k, B, q_j\rangle$ provient de la transition (q_i, s, A, q_k, B) et $[q_i, sv, A] \vdash [q_k, v, B]$. La seconde possibilité est que la première règle appliquée soit de la forme $\langle q_i, A, q_j\rangle \to s\langle q_k, B, q_m\rangle\langle q_m, A, q_j\rangle$. Dans ce cas, on a

$$\langle q_i, A, q_j\rangle \Rightarrow s\langle q_k, B, q_m\rangle\langle q_m, A, q_j\rangle \Rightarrow^* w$$

avec

$$\langle q_k, B, q_m\rangle \Rightarrow^* x,\ \langle q_m, A, q_j\rangle \Rightarrow^* y \quad \text{et} \quad w = sxy.$$

On conclut en appliquant deux fois l'hypothèse de récurrence. Dès lors, $L(G) \subseteq L(\mathcal{A})$ et ceci termine la preuve.

■

Corollaire VI.8.10. *Un langage est algébrique si et seulement si il est accepté par un automate à pile.*

Démonstration. Cela résulte immédiatement des propositions VI.8.6 et VI.8.9.

■

Remarque VI.8.11. Dans le cadre des langages réguliers, nous avons montré que les ensembles des langages acceptés par automate fini déterministe ou non déterministe coïncident. Ainsi, le caractère non déterministe n'apporte rien du point de vue des langages acceptés (il apporte néanmoins des facilités de construction non négligeables). On peut naturellement se poser la même question dans le cas des langages algébriques. On peut montrer[25] que la classe des langages acceptés par un automate à pile déterministe est un sous-ensemble strict des langages algébriques. Il s'agit des langages préfixes. Un langage $L \subseteq \Sigma^*$ est *préfixe* si

$$\forall u, v \in \Sigma^* : u \in L, uv \in L \Rightarrow v = \varepsilon.$$

Autrement dit, si un mot u est dans L, aucun préfixe propre de u n'appartient à L.

9. Stabilité du caractère algébrique

Nous avons vu précédemment que l'ensemble des langages algébriques était stable pour l'union, la concaténation et l'étoile de Kleene. Nous allons montrer ici que l'intersection de deux langages algébriques n'est en général pas algébrique. Par conséquent, le complémentaire d'un langage algébrique n'est en général pas algébrique. Néanmoins, l'intersection d'un langage algébrique et d'un langage régulier est encore algébrique.

Exemple VI.9.1. Le langage $L = \{a^n b^n \mid n \in \mathbb{N}\}c^*$ est algébrique car il s'obtient comme la concaténation de deux langages algébriques (cf. proposition VI.4.3). De même, le langage $M = a^*\{b^n c^n \mid n \in \mathbb{N}\}$ est aussi algébrique. Il est clair que

$$L \cap M = \{a^n b^n c^n \mid n \in \mathbb{N}\}$$

n'est pas algébrique. Ainsi, cet exemple montre que l'ensemble des langages algébriques n'est pas stable pour l'intersection.

Remarque VI.9.2. Supposons que pour tout langage $L \subset \Sigma^*$, L algébrique entraîne $\Sigma^* \setminus L$ algébrique. Dans ce cas, puisque

$$L \cap M = \Sigma^* \setminus ((\Sigma^* \setminus L) \cup (\Sigma^* \setminus M)),$$

on pourrait en conclure que l'intersection de deux langages algébriques est encore algébrique (en effet, nous savons que l'union de deux langages algébriques est algébrique, cf. proposition VI.4.2). Ainsi, l'ensemble des langages algébriques ne peut pas être stable pour le passage au complémentaire.

[25] Voir par exemple, J.-M. Autebert, *Langages Algébriques*, études et recherche en informatique, Masson, Paris, (1987).

Théorème VI.9.3. *Soient $R \subseteq \Sigma^*$ un langage régulier et $L \subseteq \Sigma^*$ un langage algébrique. Le langage $L \cap R$ est algébrique.*

Démonstration. L'idée de la démonstration consiste, tout comme dans le cas de l'intersection de deux langages réguliers, à construire un automate produit simulant simultanément le comportement d'un automate fini déterministe acceptant R et d'un automate à pile acceptant L. Soient $\mathcal{A} = (Q, q_0, F, \Sigma, \delta)$ et $\mathcal{A}' = (Q', \Sigma, \Pi, \delta', q_0', F')$ deux tels automates où $\mathcal{A}'$ est supposé élémentaire[26]. On considère l'automate à pile

$$\mathcal{P} = (Q \times Q', \Sigma, \Pi, \tau, (q_0, q_0'), F \times F')$$

où la relation de transition τ est donnée par

$$((q_i, q_i'), \sigma, x, (q_j, q_j'), y) \in \tau \text{ si } \delta(q_i, \sigma) = q_j \text{ et } (q_i', \sigma, x, q_j', y) \in \delta'$$

et

$$((q_i, q_i'), \varepsilon, x, (q_i, q_j'), y) \in \tau \text{ si } (q_i', \varepsilon, x, q_j', y) \in \delta'.$$

Il est facile de se convaincre que le langage accepté par $\mathcal{P}$ est exactement $L \cap R$. On conclut en utilisant la proposition VI.8.9.

■

Lemme VI.9.4. *L'ensemble des langages algébriques est stable par morphisme.*

Démonstration. Soient $L \subseteq \Sigma^*$ un langage algébrique généré par $G = (V, \Sigma, P, S)$ et $h : \Sigma \to \Gamma^*$ un morphisme. Nous supposerons de plus que les trois alphabets V, Σ et Γ sont deux à deux disjoints. Considérons la grammaire $G' = (V \cup \Sigma, \Gamma, P', S)$ où

$$P' = P \cup \{\sigma \to h(\sigma) \mid \sigma \in \Sigma\}.$$

Il est clair que cette nouvelle grammaire génère le langage $h(L)$.

■

10. Un théorème de Schützenberger

Nous terminons ce chapitre par une caractérisation des langages acceptés par un automate à pile. Ainsi, ce résultat montre en particulier qu'un langage algébrique peut toujours s'obtenir comme l'image par un morphisme de l'intersection d'un langage régulier et du langage formé de mots transformant la pile vide en elle-même. Expliquons notre propos.

Dans cette section $\mathcal{A} = (Q, \Sigma, \Pi, \delta, q_0, F)$ est un automate à pile fixé une fois pour toutes et supposé élémentaire. Si $\Pi = \{\pi_1, \ldots, \pi_k\}$, on considère l'alphabet

$$\Phi = \Sigma \cup \{e_1, \ldots, e_k, d_1, \ldots, d_k\}$$

où $e_1, \ldots, e_k, d_1, \ldots, d_k$ sont de nouveaux symboles[27]. Si $p \in \Pi^*$ représente l'état de la pile, les éléments de Φ agissent sur p comme suit :

[26] Il faut surtout ne pouvoir lire qu'au plus une lettre de Σ à chaque transition.

[27] "e" comme "empilement" et "d" comme dépilement

- $\forall \sigma \in \Sigma$, $\sigma.p = p$,
- $\forall i \in \{1, \ldots, k\}$, $e_i.p = \pi_i\, p$,
- si $p = \pi_i\, p'$, alors $d_i.p = p'$.

Si $x = x_1 \cdots x_n \in \Phi^*$, on a $x.p = x_1.(\cdots(x_{n-1}.(x_n.p))\cdots)$ pour autant que ces opérations puissent être définies[28]. En particulier, $xy.p = x.(y.p)$, $x, y \in \Phi^*$.

Définition VI.10.1. Avec les notations qui précèdent, on introduit le langage $D_{\mathcal{A}}$ formé des mots qui transforment la pile vide en elle-même, i.e.,

$$D_{\mathcal{A}} = \{x \in \Phi^* \mid x.\varepsilon = \varepsilon\}.$$

Proposition VI.10.2. *Le langage $D_{\mathcal{A}}$ est algébrique et généré par la grammaire dont les règles sont données par*

$$S \to Sd_1Se_1S \,|\, \cdots \,|\, Sd_kSe_kS \,|\, \sigma_1S \,|\, \cdots \,|\, \sigma_mS \,|\, \varepsilon.$$

Démonstration. Soit $w \in D_{\mathcal{A}}$. Montrons que $S \Rightarrow^* w$ par récurrence sur $|w|$. Si $w = \varepsilon$, le résultat est satisfait. Supposons le résultat acquis pour les mots de longueur au plus ℓ et vérifions-le pour les mots de longueur $\ell + 1$. Si $w \in \Sigma^{\ell+1}$, le résultat est immédiat. Il suffit d'appliquer $\ell + 1$ règles $S \to \sigma_t S$. Nous pouvons donc supposer que w contient un symbole e_i (que nous prenons le plus à droite possible dans w), i.e.,

$$w = ue_iz, \quad \text{avec } u \in \Phi^*, z \in \Sigma^*.$$

Il est clair que z ne contient pas de symbole e_j (par choix de e_i), mais il ne peut non plus contenir des symboles d_j (car w ne définirait pas une action valide). De plus, à ce symbole e_i réalisant l'empilement de π_i, il correspond exactement un symbole d_i le dépilant (car $w \in D_{\mathcal{A}}$). Ainsi, on a

$$w = xd_iye_iz, \quad \text{avec } x, y \in \Phi^*.$$

De là, on tire que x, y et z appartiennent à $D_{\mathcal{A}}$. En appliquant trois fois l'hypothèse de récurrence, on a $S \Rightarrow^* x$, $S \Rightarrow^* y$, $S \Rightarrow^* z$. On conclut en remarquant que $S \Rightarrow Sd_iSe_iS \Rightarrow^* xd_iye_iz = w$.

Passons à la réciproque et vérifions que si $S \Rightarrow^* w$, $w \in \Phi^*$, alors w appartient à $D_{\mathcal{A}}$. Pour tout $p \in \Pi^*$, on pose $S.p = p$. De cette façon, on étend l'action sur Π^* de Φ^* à $(\Phi \cup \{S\})^*$. Montrons par récurrence sur la longueur de la dérivation que si $S \Rightarrow^* w$, $w \in (\Phi \cup \{S\})^*$, alors $w.p = p$ pour tout $p \in \Pi^*$. Si la dérivation est de longueur nulle, on a bien $S.p = p$. Supposons le résultat satisfait pour les dérivations de longueur au plus k et vérifions-le pour une dérivation de longueur $k + 1$. Ainsi, en mettant en évidence la dernière règle appliquée, cette dérivation se décompose en

$$S \Rightarrow^* w_1Sw_2 \Rightarrow w.$$

[28] Par exemple, $d_2e_1.p$ n'est jamais défini et ce, quel que soit p. En effet, e_1 empile π_1 et on devrait ensuite dépiler e_2 qui ne se trouve pas au sommet de la pile.

Si la dernière règle appliquée est de la forme $S \to Sd_iSe_iS$, alors $w = w_1Sd_iSe_iSw_2$ et pour tout $p \in \Pi^*$, on a

$$\begin{aligned} w_1Sd_iSe_iSw_2.p &= w_1Sd_iSe_iS.(w_2.p) = w_1Sd_iSe_i.(w_2.p) \\ &= w_1Sd_iS.(\pi_i(w_2.p)) = w_1Sd_i.(\pi_i(w_2.p)) = w_1S.(w_2.p) \\ &= (w_1Sw_2).p = p \end{aligned}$$

où, à la dernière ligne, on a appliqué l'hypothèse de récurrence. Si la dernière règle appliquée est de la forme $S \to \sigma_iS$, alors $w = w_1\sigma_iSw_2$ et pour tout $p \in \Pi^*$,

$$w_1\sigma_iSw_2.p = w_1\sigma_iS.(w_2.p) = w_1\sigma_i.(w_2.p) = w_1.(w_2.p) = (w_1Sw_2).p = p.$$

Enfin, si la dernière règle appliquée est de la forme $S \to \varepsilon$, $w = w_1w_2$ et pour tout $p \in \Pi^*$,

$$w_1w_2.p = (w_1Sw_2).p = p.$$

Ceci conclut la preuve.

■

Théorème VI.10.3 (Thm. de représentation de Chomsky-Schützenberger). *Soit $L \subseteq \Sigma^*$ un langage accepté par un automate à pile $\mathcal{A}$. Il existe un morphisme $h : \Phi \to \Sigma^*$ et un langage régulier $R \subseteq \Phi^*$ tel que*

$$L = h(D_{\mathcal{A}}^R \cap R).$$

En particulier, L est algébrique.

Démonstration. Puisque nous supposons $\mathcal{A}$ élémentaire, on peut voir cet automate comme un automate fini sur l'alphabet Φ. En effet, il suffit de remplacer les transitions par des arcs de label appartenant à Φ :

- $(p, \varepsilon, \varepsilon, q, \varepsilon)$ devient $p \xrightarrow{\varepsilon} q$,
- $(p, \sigma_i, \varepsilon, q, \varepsilon)$ devient $p \xrightarrow{\sigma_i} q$,
- $(p, \varepsilon, \pi_i, q, \varepsilon)$ devient $p \xrightarrow{e_i} q$,
- $(p, \varepsilon, \varepsilon, q, \pi_i)$ devient $p \xrightarrow{d_i} q$.

Par le théorème de Kleene, le langage $R \subseteq \Phi^*$ accepté par cet automate est régulier. Soit $w \in \Sigma^*$, un mot accepté par l'automate à pile $\mathcal{A}$. Cela signifie que

$$[q_0, w, \varepsilon] \vdash^* [q, \varepsilon, \varepsilon], \quad \text{avec } q \in F.$$

A ce mot, il correspond donc un chemin dans $\mathcal{A}$ de label $W \in \Phi^*$ débutant en q_0 et aboutissant en $q \in F$. Puisque, pour être accepté par l'automate, la suite de configurations débute et se finit par une pile vide, il est clair que W^R appartient à $D_{\mathcal{A}}$. De plus, W appartient à R. On retrouve w en appliquant à W le morphisme

$$h : \Phi \to \Sigma^* : \begin{cases} \sigma_i \mapsto \sigma_i \\ e_i \mapsto \varepsilon \\ d_i \mapsto \varepsilon. \end{cases}$$

Ce morphisme est simplement défini pour effacer les lettres de $\Phi \setminus \Sigma$. Réciproquement, si $W \in \Phi^*$ appartient à $D_{\mathcal{A}}^R \cap R$, alors on se convainc aisément que $h(W)$ est un mot accepté par l'automate.

■

Exemple VI.10.4. On a représenté à la figure VI.16 un automate à pile élémentaire acceptant $\{a^n b^n \mid n \in \mathbb{N}\}$. Le langage régulier accepté par cet

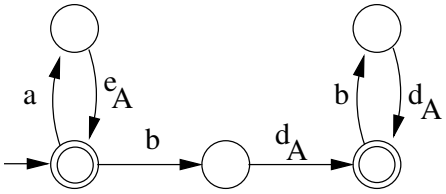


FIGURE VI.16. Automate élémentaire acceptant $\{a^n b^n \mid n \in \mathbb{N}\}$.

automate est

$$R = (a\, e_A)^* (b\, d_A)^*.$$

De plus, $D_{\mathcal{A}} = \{d_A^n e_A^n \mid n \in \mathbb{N}\}$. Ce langage est généré par la grammaire

$$S \to \varepsilon \,|\, d_A S e_A.$$

Par exemple, le mot $a\, e_A\, a\, e_A\, b\, d_A$ appartient à R mais n'appartient pas à $D_{\mathcal{A}}^R$ car le mot miroir $d_A\, b\, e_A\, a\, e_A\, a$ ne transforme pas la pile vide en la pile vide, en effet: $d_A\, b\, e_A\, a\, e_A\, a.\varepsilon = A$. Ainsi, $a\, e_A\, a\, e_A\, b\, d_A$ n'appartient pas à $D_{\mathcal{A}} \cap R$. Par contre, le mot $a\, e_A\, b\, d_A$ appartient à R et à $D_{\mathcal{A}}^R$ car $d_A\, b\, e_A\, a.\varepsilon = \varepsilon$. Si on lui applique le morphisme h, on trouve

$$h(a\, e_A\, b\, d_A) = ab$$

qui est bien un mot du langage.

11. Exercices

Exercice VI.11.1. Déterminer une grammaire hors contexte générant les langages

$$L = \{ww^R \mid w \in \{a, b\}^*\}$$

et

$$M = \{wcw^R \mid w \in \{a, b\}^*\}.$$

Exercice VI.11.2. Décrire une grammaire hors contexte générant les langages

$$\{a^n b^n c^m \mid n, m \in \mathbb{N}\},$$
$$\{a^n b^m c^m \mid n, m \in \mathbb{N}\}$$

et

$$\{a^n b^m \mid m > n \geq 0\}.$$

Exercice VI.11.3. Le langage $\{a^i b^j c^k \mid i \neq j \text{ ou } i \neq k\}$ est-il algébrique ? Justifier votre réponse.

Exercice VI.11.4. Le langage

$$L = \{w \in \{a,b,c\}^* \,:\, |w|_a = |w|_b = |w|_c\}$$

est-il algébrique ? Justifier. Remarquer qu'il s'agit, en particulier, de la clôture commutative du langage $\{a^n b^n c^n \mid n \in \mathbb{N}\}$.

Exercice VI.11.5. Dans le plan muni d'un repère orthonormé, on symbolise un déplacement d'une unité vers la droite (resp. la gauche, le haut, le bas) par la lettre D (resp. G, H, B). Ainsi, une suite de déplacements est représentée par un mot sur $\{D, G, H, B\}$. Caractériser le langage des mots qui correspondent à un déplacement de deux unités vers la droite. Ce langage est-il algébrique ? Justifier.

Exercice VI.11.6. Soit le morphisme f tel que $f(a) = b$ et $f(b) = a$. Le langage

$$L = \{wf(w) \mid w \in \{a,b\}^*\}$$

est-il algébrique ?

Exercice VI.11.7. Le langage formé des mots comprenant deux fois plus de a que de b est-il algébrique ? Même question avec le langage

$$L = \{w \in \{a,b\}^* \,:\, |w|_a = 2|w|_b + 3\}.$$

Exercice VI.11.8. Le langage des palindromes est-il algébrique ? Justifier. Même question en considérant uniquement les palindromes de longueur paire.

Exercice VI.11.9. Donnez la description d'un automate à pile déterministe acceptant le langage

$$\{wcw^R \mid w \in \{a,b\}^*\}.$$

Exercice VI.11.10. Décrire un automate à pile acceptant le langage formé des palindromes sur l'alphabet $\{a,b\}$. Même question en considérant uniquement les palindromes de longueur paire.

Exercice VI.11.11. Quels sont les langages acceptés respectivement par les automates à pile suivants

Exercice VI.11.12. Décrire un automate à pile acceptant le langage L formé des mots sur $\{a,b\}$ pour lesquels il existe un préfixe contenant (strictement) plus de b que de a, i.e.,

$$\exists u, v \in \{a,b\}^* : w = uv \text{ et } |u|_b > |u|_a.$$

Par exemple, *baa*, *abba* et *abbaaa* appartiennent à L mais *aab* et *ababab* n'y appartiennent pas.

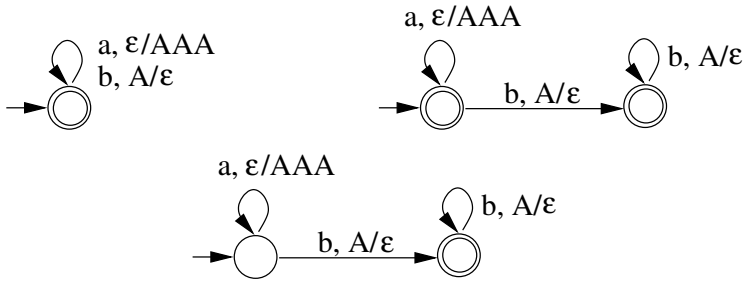


FIGURE VI.17. Automates à pile.

Exercice VI.11.13. Montrer que si l'automate à pile $\mathcal{A} = (Q, \Sigma, \Pi, \delta, q_0, F)$ est élémentaire alors les langages

$$G_{p,q} = \{x \in \Pi^* \mid \exists m \in \Sigma^* : [p, m, x] \vdash^* [q, \varepsilon, \varepsilon]\}, \quad p, q \in Q$$

sont réguliers. (Suggestion : exprimer $t^{-1}.G_{p,q}$ à l'aide des différents langages $G_{r,s}$, $t \in \Pi^*$.)

Exercice VI.11.14. Utiliser le lemme de la pompe pour montrer que les langages suivants ne sont pas algébriques

- $\{a^{k^2} \mid k \in \mathbb{N}\}$
- $\{a^i b^j c^i d^j \mid i, j \geq 0\}$
- L'ensemble des préfixes de longueur finie du mot infini

$$abaabaaab \cdots ba^n ba^{n+1} b \cdots$$

Exercice VI.11.15. Les langages suivants sont-ils algébriques ?

$$L = \{a^i b^{2i} c^j \mid i, j \geq 0\},$$
$$M = \{a^j b^i c^{2i} \mid i, j \geq 0\}$$

et

$$L \cap M.$$

Exercice VI.11.16. Soit le langage

$$L = \{ww \mid w \in \{a, b\}^*\}.$$

Montrer que $\{a, b\}^* \setminus L$ est algébrique mais que L ne l'est pas.

Exercice VI.11.17. Fournir une grammaire hors contexte générant le langage $L = \{a^i b^j c^j d^{2i} \mid i, j \in \mathbb{N}\}$.

Exercice VI.11.18. Fournir une grammaire hors contexte générant le langage formé des mots sur $\{a, b, c\}$ qui commencent par a, se terminent par bac et qui comprennent un nombre pair de c.

Exercice VI.11.19. Mettre sous forme essentiellement monotone la grammaire suivante (élimination des ε-productions)

$$\begin{array}{rcl} S & \to & ACA \\ A & \to & aAa \mid B \mid C \\ B & \to & bB \mid b \\ C & \to & cC \mid \varepsilon. \end{array}$$

Si la grammaire obtenue contient des 1-productions, les éliminer elles aussi pour obtenir une grammaire équivalente.

Exercice VI.11.20. Mettre sous forme essentiellement monotone la grammaire suivante (élimination des ε-productions)

$$\begin{array}{rcl} S & \to & ABC \\ A & \to & aA \mid \varepsilon \\ B & \to & bB \mid \varepsilon \\ C & \to & cC \mid \varepsilon. \end{array}$$

Quel est le langage généré par cette grammaire ?

Exercice VI.11.21. Mettre sous forme normale de Chomsky, la grammaire suivante

$$\begin{array}{rcl} S & \to & aABA \mid aBB \\ A & \to & bA \mid b \\ B & \to & cB \mid c. \end{array}$$

Exercice VI.11.22. Mettre sous forme normale de Chomsky, la grammaire suivante

$$\begin{array}{rcl} S & \to & A \mid ABa \mid AbA \\ A & \to & Aa \mid \varepsilon \\ B & \to & Bb \mid BC \\ C & \to & CB \mid CA \mid bB. \end{array}$$

Exercice VI.11.23. Fournir une grammaire non restrictive de type **0** générant le langage

$$\{ww \mid w \in \{a,b\}^*\}.$$

Même question avec le langage

$$\{w^3 \mid w \in \{a,b\}^*\}.$$

Bibliographie

[1] A. Aho, J. Ullman, *Concepts fondamentaux de l'informatique*, Dunod, Paris, (1993).

[2] A. Aho, R. Sethi, J. Ullman, *Compilers: Principles, Techniques, and Tools*, Addison-Wesley, (1986).

[3] J.-P. Allouche, J. Shallit, *Automatic Sequences, Theory, Applications, Generalizations*, Cambridge University Press, Cambridge, (2003).

[4] J.-P. Allouche, J. Shallit, The ubiquitous Prouhet-Thue-Morse sequence, *Sequences and their applications* (Singapore, 1998), 1–16, Springer Ser. Discrete Math. Theor. Comput. Sci., Springer, London, 1999.

[5] J.-M. Autebert, *Langages algébriques*, études et recherches en informatique, Masson, Paris, (1987).

[6] J.-M. Autebert, J. Berstel, L. Boasson, *Context-Free Languages and Pushdown Automata*, Handbook of Formal Languagues, Vol. **1**, Springer, (1997).

[7] E. Bach, J. Shallit, *Algorithmic number theory, Efficient algorithms*, Foundations of Computing Series, MIT Press, Cambridge, MA, (1996).

[8] J. Berstel, L. Boasson, The set of minimal words of a context-free language is context-free, *J. Comput. System Sci.* **55** (1997), 477–488

[9] J. Berstel, C. Reutenauer, *Les séries rationnelles et leurs langages*, Études et Recherches en Informatique, Masson, Paris, (1984).

[10] J. Berstel, D. Perrin, The origins of combinatorics on words, *European J. Combin.* **28** (2007), 996–1022.

[11] V. Berthé, *Combinatoire des mots*, cours de DEA, Univ. Montpellier II (2006).

[12] V. Bruyère, G. Hansel, C. Michaux, R. Villemaire, Logic and p-recognizable sets of integers, Journées Montoises (Mons, 1992), *Bull. Belg. Math. Soc.* **1** (1994), 191–238.

[13] J. H. Conway, *Regular Algebra and Finite Machines*, Mathematics Series, Chapman and Hall, London, (1971).

[14] Ding-Zhu Du, Ker-I Ko, *Problem solving in Automata, Languages, And Complexity*, John Wiley & Sons, (2001).

[15] S. Eilenberg, *Automata, Languages, and Machines*, Vol A., Academic Press, New York-London, (1974).

[16] S. Eilenberg, *Automata, Languages, and Machines*, Vol B., Academic Press, New York-London, (1976).

[17] B. Khoussainov, A. Nerode, *Automata Theory and its Applications*, Progress in Computer Science and Applied Logic, Vol. **21**, Birkhäuser, Boston, (2003).

[18] M. V. Lawson, *Finite Automata*, Chapman & Hall/CRC Press, (2003).

[19] P. Lecomte, *Algorithmitique et Calculabilité*, notes de cours, Université de Liège, 1996.

[20] P. Lecomte, M. Rigo, Numeration systems on a regular language, *Theory Comput. Syst.* **34** (2001), 27–44

[21] M. Lothaire, *Combinatorics on words*, Cambridge Mathematical Library, Cambridge University Press, Cambridge, 1997.

[22] M. Lothaire, *Algebraic Combinatorics on Words*, Encyclopedia of Mathematics and its Applications **90**, Cambridge University Press, Cambridge, (2002).

[23] A. Mateescu, A. Salomaa, *Formal Languages: an Introduction and a Synopsis*, Handbook of Formal Languagues, vol. **1**, Springer, (1997).

[24] D. Perrin, Finite Automata, *Handbook of Theoretical Computer Science*, J. van Leeuwen Ed., Elsevier, (1990), 3–57.

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

[26] M. Rigo, Automates et numération, *Bull. Soc. Royale Sci. Liège* **74** (2005), 249–262.

[27] J. Sakarovitch, *Élements de théorie des automates*, Vuibert, Paris, (2003).

[28] A. Salomaa, *Theory of Automata*, International series of monograps on pure and applied mathematics, Pergamon Press, Oxford, (1969).

[29] J. Shallit, Numeration systems, linear recurrences, and regular sets, *Information and Computation* **113** (1994), 331–347.

[30] T. A. Sudkamp, *Languages and Machines, An Introduction to the Theory of Computer Science*, second edition, Addison-Wesley, Massachusetts, (1997).

[31] A. Szilard, S. Yu, K. Zhang, J. Shallit, Characterizing regular languages with polynomial densities, MFCS 1992, *Lect. Notes in Comput. Sci.* **629**, 494–503, Springer, (1992).

[32] W. Thomas, Automata on infinite objects, in *Handbook of theoretical computer science*, Vol. B, 133–191, Elsevier, Amsterdam, 1990.

[33] S. Yu, *Regular Languages*, Handbook of Formal Languagues, vol. **1**, Springer, (1997).

[34] P. Wolper, *Introduction à la calculabilité*, InterEditions, Paris, (1991).

Liste des figures

Index

Notations

A

T

U

V